



# **InstallShield 2016 InstallScript**

## **Reference Guide**

# Legal Information

**Book Name:** InstallShield 2016 InstallScript Reference Guide  
**Part Number:** ISP-2300-RG00  
**Product Release Date:** August 2016

## Copyright Notice

Copyright © 2016 Flexera Software LLC. All Rights Reserved.

This publication contains proprietary and confidential information and creative works owned by Flexera Software LLC and its licensors, if any. Any use, copying, publication, distribution, display, modification, or transmission of such publication in whole or in part in any form or by any means without the prior express written permission of Flexera Software LLC is strictly prohibited. Except where expressly provided by Flexera Software LLC in writing, possession of this publication shall not be construed to confer any license or rights under any Flexera Software LLC intellectual property rights, whether by estoppel, implication, or otherwise.

All copies of the technology and related information, if allowed by Flexera Software LLC, must display this notice of copyright and ownership in full.

## Intellectual Property

For a list of trademarks and patents that are owned by Flexera Software, see <http://www.flexerasoftware.com/intellectual-property>. All other brand and product names mentioned in Flexera Software products, product documentation, and marketing materials are the trademarks and registered trademarks of their respective owners.

## Restricted Rights Legend

The Software is commercial computer software. If the user or licensee of the Software is an agency, department, or other entity of the United States Government, the use, duplication, reproduction, release, modification, disclosure, or transfer of the Software, or any related documentation of any kind, including technical data and manuals, is restricted by a license agreement or by the terms of this Agreement in accordance with Federal Acquisition Regulation 12.212 for civilian purposes and Defense Federal Acquisition Regulation Supplement 227.7202 for military purposes. The Software was developed fully at private expense. All other use is prohibited.

# Contents

<b>InstallScript Language Reference</b>	<b>51</b>
<b>Integrated Compiler</b>	<b>52</b>
<b>Command-Line Compiler</b>	<b>53</b>
<b>Setup Scripts</b>	<b>55</b>
InstallScript Limits	55
Structure of a Script	56
Declarations	56
Program Block	56
Function Block	56
Identifiers	57
Syntax Punctuation Rules	57
Writing Comments	57
Using White Space	58
Hungarian Notation	59
Escape Sequences	60
Embedding Quotation Marks	61
Format Specifiers	62
Reserved Words	64
<b>Language Keywords</b>	<b>65</b>
abort	65
BOOL	65
cdecl	65
exit	65
export	66
external	66
for...endfor	66
goto	67
if	67
if Structure with goto	68

if-then Structure .....	68
if-then-else Structure .....	69
Nested if-then-else Structure .....	69
elseif Structure .....	70
method .....	70
property() .....	70
prototype .....	71
repeat...until .....	71
return .....	72
set .....	72
stdcall .....	73
switch...endswitch .....	73
try, catch, and endcatch .....	74
void .....	74
while...endwhile .....	75
Nested while Example .....	75
Flow Control .....	77
<b>Predefined Constants .....</b>	<b>79</b>
AFTER .....	79
ALLCONTENTS .....	80
ALLCONTROLS .....	80
APPEND .....	80
ASKDESTPATH .....	80
ASKOPTIONS .....	80
ASKPATH .....	81
ASKTEXT .....	81
BACK .....	81
BACKBUTTON .....	82
BACKGROUND .....	82
BACKGROUNDCAPTION .....	83
BASEMEMORY .....	83
BEFORE .....	83
BIF_BROWSEFORCOMPUTER .....	84
BIF_BROWSEFORPRINTER .....	84
BIF_DONTGOBELOWDOMAIN .....	84
BIF_EDITBOX .....	84
BIF_RETURNFSANCESTORS .....	85
BIF_RETURNONLYFSDIRS .....	85
BIF_STATUSTEXT .....	85
BILLBOARD .....	85
BITMAPICON .....	86
BK_BLUE .....	86
BK_GREEN .....	86
BK_MAGENTA .....	86
BK_ORANGE .....	86
BK_PINK .....	87

BK_RED.....	87
BK_SMOOTH.....	87
BK_SOLIDBLACK.....	87
BK_SOLIDBLUE.....	87
BK_SOLIDGREEN.....	88
BK_SOLIDMAGENTA.....	88
BK_SOLIDORANGE.....	88
BK_SOLIDPINK.....	88
BK_SOLIDRED.....	88
BK_SOLIDWHITE.....	89
BK_SOLIDYELLOW.....	89
BK_YELLOW.....	89
BLACK.....	89
BLUE.....	89
BOOTUPDRIVE.....	90
BUTTON_CHECKED.....	90
BUTTON_UNCHECKED.....	90
BYTES.....	90
CANCEL.....	91
CANCELBUTTON.....	91
CDROM.....	91
CDROM_DRIVE.....	91
CENTERED.....	91
CHECKBOX.....	92
CHECKBOX95.....	92
CHECKLINE.....	92
CHECKMARK.....	92
CLEAR_FILE_ATTR.....	92
COLORS.....	93
COMMAND.....	93
COMMON.....	93
COMPACT.....	93
COMPARE_DATE.....	93
COMPARE_MD5_SIGNATURE.....	94
COMPARE_SIZE.....	94
COMPARE_VERSION.....	94
COMP_NORMAL.....	94
COMP_UPDATE_DATE.....	94
COMP_UPDATE_SAME.....	95
COMP_UPDATE_VERSION.....	95
CONTINUE.....	95
COPY_ERR_CREATEDIR.....	95
COPY_ERR_MEMORY.....	96
COPY_ERR_NODISKSPACE.....	96
COPY_ERR_OPENINPUT.....	96
COPY_ERR_OPENOUTPUT.....	96

COPY_ERR_TARGETREADONLY .....	96
CPU .....	97
CS_OPTION_FLAG_NO_NEW_INSTALL_HIGHLIGHT .....	97
CS_OPTION_FLAG_NO_STARTSCREEN_PIN .....	97
CS_OPTION_FLAG_PREVENT_PINNING .....	97
CS_OPTION_FLAG_REPLACE_EXISTING .....	98
CS_OPTION_FLAG_RUN_MAXIMIZED .....	98
CS_OPTION_FLAG_RUN_MINIMIZED .....	98
CURRENTROOTKEY .....	98
CUSTOM .....	98
DATA_COMPONENT .....	99
DATA_LIST .....	99
DATA_NUMBER .....	99
DATA_STRING .....	99
DATE .....	100
DEFAULT .....	100
DEFWINDOWMODE .....	100
DELETE .....	100
DELETE_EOF .....	100
DIALOGCACHE .....	101
DIFXAPI_ERROR .....	101
DIFXAPI_INFO .....	101
DIFXAPI_SUCCESS .....	101
DIFXAPI_WARNING .....	101
DIRECTORY .....	102
DIR_WRITEABLE .....	102
DISABLE_ALLUSERBTN .....	102
DISABLE_PERUSERBTN .....	102
DISK .....	103
DISK1FEATURE .....	103
DISK_INFO_QUERY_ALL .....	103
DISK_INFO_QUERY_BYTES_PER_CLUSTER .....	104
DISK_INFO_QUERY_DISK_FREE_SPACE .....	104
DISK_INFO_QUERY_DISK_TOTAL_SPACE .....	104
DISK_INFO_QUERY_DRIVE_TYPE .....	104
DISK_TOTALSPACE .....	104
DISK_TOTALSPACE_EX .....	105
DLG_ASK_OPTIONS .....	105
DLG_ASK_PATH .....	105
DLG_ASK_TEXT .....	105
DLG_ASK_YESNO .....	105
DLG_CENTERED .....	106
DLG_CLOSE .....	106
DLG_DIR_DIRECTORY .....	106
DLG_DIR_DRIVE .....	106
DLG_DIR_FILE .....	106

DLG_ENTER_DISK .....	107
DLG_ERR .....	107
DLG_ERR_ALREADY_EXISTS .....	107
DLG_ERR_ENDDLG .....	107
DLG_INFO_ALTIMAGE .....	107
DLG_INFO_ALTIMAGE_HIDPI .....	108
DLG_INFO_ALTIMAGE_REVERT_IMAGE .....	108
DLG_INFO_ALTIMAGE_VERIFY_BMP .....	108
DLG_INFO_CHECKSELECTION .....	108
DLG_INFO_KUNITS .....	108
DLG_INFO_USEDECIMAL .....	109
DLG_INIT .....	109
DLG_MSG_ALL .....	109
DLG_MSG_INFORMATION .....	109
DLG_MSG_SEVERE .....	109
DLG_MSG_STANDARD .....	110
DLG_MSG_WARNING .....	110
DLG_STATUS .....	110
DLG_USER_CAPTION .....	110
DOINSTALL_OPTION_NOHIDEPROGRESS .....	110
DOINSTALL_OPTION_NOHIDESPLASH .....	111
DOINSTALL_OPTION_NOLANGSWITCH .....	111
DOINSTALL_OPTION_NOSETBATCHINSTALL .....	111
DOTNETFRAMEWORKINSTALLED .....	111
DOTNETSERVICEPACKINSTALLED .....	111
DRIVE .....	112
DRIVE_CDROM .....	112
DRIVE_FIXED .....	112
DRIVE_NO_ROOT_DIR .....	112
DRIVE_RAMDISK .....	112
DRIVE_REMOTE .....	113
DRIVE_REMOVABLE .....	113
DRIVE_UNKNOWN .....	113
DRIVER_PACKAGE_DELETE_FILES .....	113
DRIVER_PACKAGE_FORCE .....	113
DRIVER_PACKAGE_LEGACY_MODE .....	114
DRIVER_PACKAGE_ONLY_IF_DEVICE_PRESENT .....	114
DRIVER_PACKAGE_REPAIR .....	114
DRIVER_PACKAGE_SILENT .....	114
EDITBOX_CHANGE .....	114
EFF_BOXSTRIPE .....	115
EFF_FADE .....	115
EFF_HORZREVEAL .....	115
EFF_HORZSTRIPE .....	115
EFF_NONE .....	115
EFF_REVEAL .....	116

EFF_VERTSTRIPE .....	116
END_OF_FILE .....	116
END_OF_LIST .....	116
ENTERDISK .....	117
EQUALS .....	117
ERROR_ACCESS_DENIED .....	117
ERROR_CIRCULAR_DEPENDENCY .....	118
ERROR_DATABASE_DOES_NOT_EXIST .....	118
ERROR_DEPENDENT_SERVICES_RUNNING .....	118
ERROR_DUP_NAME .....	118
ERROR_FILE_NOT_FOUND .....	119
ERROR_INVALID_HANDLE .....	119
ERROR_INVALID_PARAMETER .....	119
ERROR_INVALID_SERVICE_ACCOUNT .....	119
ERROR_INVALID_SERVICE_CONTROL .....	120
ERROR_PATH_NOT_FOUND .....	120
ERROR_SERVICE_ALREADY_RUNNING .....	120
ERROR_SERVICE_CANNOT_ACCEPT_CTRL .....	120
ERROR_SERVICE_DATABASE_LOCKED .....	121
ERROR_SERVICE_DEPENDENCY_DELETED .....	121
ERROR_SERVICE_DEPENDENCY_FAIL .....	121
ERROR_SERVICE_DISABLED .....	121
ERROR_SERVICE_DOES_NOT_EXIST .....	122
ERROR_SERVICE_EXISTS .....	122
ERROR_SERVICE_LOGON_FAILED .....	122
ERROR_SERVICE_NOT_ACTIVE .....	122
ERROR_SERVICE_NO_THREAD .....	123
ERROR_SERVICE_REQUEST_TIMEOUT .....	123
ERROR_TIMEOUT .....	123
ERR_ABORT .....	123
ERR_BOX_BADPATH .....	124
ERR_BOX_BADTAGFILE .....	124
ERR_BOX_DISKID .....	124
ERR_BOX_DRIVEOPEN .....	124
ERR_IGNORE .....	124
ERR_NO .....	125
ERR_PERFORM_AFTER_REBOOT .....	125
ERR_RETRY .....	125
ERR_YES .....	125
EXCLUDE_SUBDIR .....	125
EXCLUSIVE .....	126
EXISTS .....	126
EXIT .....	126
EXTENDEDMEMORY .....	126
EXTENSION_ONLY .....	127
FALSE .....	127



FEATURE_FIELD_CDROM_FOLDER .....	128
FEATURE_FIELD_DESCRIPTION .....	128
FEATURE_FIELD_DISPLAYNAME .....	128
FEATURE_FIELD_ENCRYPT .....	129
FEATURE_FIELD_FILENEED .....	129
FEATURE_FIELD_FLAGS .....	129
FEATURE_FIELD_FTPLOCATION .....	129
FEATURE_FIELD_GUID .....	130
FEATURE_FIELD_HANDLER_ONINSTALLED .....	130
FEATURE_FIELD_HANDLER_ONINSTALLING .....	130
FEATURE_FIELD_HANDLER_ONUNINSTALLED .....	131
FEATURE_FIELD_HANDLER_ONUNINSTALLING .....	131
FEATURE_FIELD_HTTPLOCATION .....	131
FEATURE_FIELD_IMAGE .....	132
FEATURE_FIELD_MISC .....	132
FEATURE_FIELD_PASSWORD .....	132
FEATURE_FIELD_SELECTED .....	133
FEATURE_FIELD_SIZE .....	133
FEATURE_FIELD_STATUS .....	133
FEATURE_FIELD_VISIBLE .....	134
FEATURE_INFO_ATTRIBUTE .....	134
FEATURE_INFO_COMPONENT_FLAGS .....	134
FEATURE_INFO_COMPsize_HIGH .....	134
FEATURE_INFO_COMPsize_LOW .....	135
FEATURE_INFO_DATE .....	135
FEATURE_INFO_DATE_EX .....	135
FEATURE_INFO_DESTINATION .....	136
FEATURE_INFO_FTPLOCATION .....	136
FEATURE_INFO_HTTPLOCATION .....	136
FEATURE_INFO_LANGUAGE .....	137
FEATURE_INFO_MD5_SIGNATURE .....	137
FEATURE_INFO_MISC .....	137
FEATURE_INFO_ORIGsize_HIGH .....	138
FEATURE_INFO_ORIGsize_LOW .....	138
FEATURE_INFO_OS .....	138
FEATURE_INFO_OVERWRITE .....	138
FEATURE_INFO_PLATFORM_SUITE .....	139
FEATURE_INFO_TIME .....	139
FEATURE_INFO_VERSIONLS .....	139
FEATURE_INFO_VERSIONMS .....	140
FEATURE_INFO_VERSIONSTR .....	140
FEATURE_OPCOST_UNINSTALL_FILE .....	140
FEATURE_OPCOST_UNINSTALL_REGORINI .....	141
FEATURE_OPCOST_UNINSTALL_UNREGFILE .....	141
FEATURE_VALUE_CRITICAL .....	141
FEATURE_VALUE_HIGHLYRECOMMENDED .....	141

FEATURE_VALUE_STANDARD .....	142
File Attributes .....	142
FILE_ADD_FILE .....	142
FILE_ADD_SUBDIRECTORY .....	142
FILE_ALL_ACCESS .....	143
FILE_APPEND_DATA .....	143
FILE_ATTR_ARCHIVED .....	143
FILE_ATTR_HIDDEN .....	143
FILE_ATTR_NORMAL .....	143
FILE_ATTR_READONLY .....	144
FILE_ATTR_SYSTEM .....	144
FILE_ATTRIBUTE .....	144
FILE_BIN_CUR .....	144
FILE_BIN_END .....	144
FILE_BIN_START .....	145
FILE_DATE .....	145
FILE_DELETE_CHILD .....	145
FILE_EXECUTE .....	145
FILE_EXISTS .....	145
FILE_INSTALLED .....	146
FILE_IS_LOCKED .....	146
FILE_LINE_LENGTH .....	146
FILE_LIST_DIRECTORY .....	146
FILE_LOCKED .....	146
FILE_MD5_SIGNATURE .....	147
FILE_MODE_APPEND .....	147
FILE_MODE_APPEND_UNICODE .....	147
FILE_MODE_BINARY .....	147
FILE_MODE_BINARYREADONLY .....	147
FILE_MODE_NORMAL .....	148
FILE_NOT_FOUND .....	148
FILE_NO_VERSION .....	148
FILE_RD_ONLY .....	148
FILE_READ_ATTRIBUTES .....	149
FILE_READ_DATA .....	149
FILE_READ_EA .....	149
FILE_SHARED_COUNT .....	149
FILE_SIZE .....	150
FILE_SIZE_HIGH .....	150
FILE_SIZE_LOW .....	150
FILE_SRC_OLD .....	150
FILE_TIME .....	150
FILE_TRAVERSE .....	151
FILE_WRITE_ATTRIBUTES .....	151
FILE_WRITE_DATA .....	151
FILE_WRITE_EA .....	151

FILE_WRITEABLE .....	151
FILENAME .....	152
FILENAME_ONLY .....	152
FIXED_DRIVE .....	152
FONT_AVAILABLE .....	152
FULL .....	152
FULLSCREEN .....	153
FULLSCREENSIZE .....	153
FULLWINDOWMODE .....	153
FUNCTION_EXPORTED .....	153
GBYTES .....	154
GENERIC_ALL .....	154
GENERIC_EXECUTE .....	154
GENERIC_READ .....	154
GENERIC_WRITE .....	154
GREATER_THAN .....	155
GREEN .....	155
GTFIS_OPTION_DELETE_TEMP_FILE .....	155
GTFIS_OPTION_DONT_CREATE_DIR .....	155
GTFIS_OPTION_DONT_RESOLVE_TEXTSUBS .....	155
GTFIS_OPTION_NONE .....	156
HELP .....	156
HIDE_DISABLED_BTNS .....	156
HKEY_CLASSES_ROOT .....	156
HKEY_CURRENT_USER .....	157
HKEY_LOCAL_MACHINE .....	157
HKEY_USERS .....	157
HKEY_USER_SELECTABLE .....	158
HOURLASS .....	158
HWND_DESKTOP .....	158
HWND_INSTALL .....	158
IDCANCEL .....	158
IDOK .....	159
IDS_IFX_ERROR_INVALID_MEDIA_PASSWORD .....	159
IFX_ONNEXTDISK_PACKAGE_CAPTION .....	159
IFX_ONNEXTDISK_PACKAGE_MSG .....	160
INCLUDE_SUBDIR .....	160
INDVFILESTATUS .....	160
INFORMATION .....	160
IS_PERMISSIONS_OPTION_64BIT_OBJECT .....	161
IS_PERMISSIONS_OPTION_ALLOW_ACCESS .....	161
IS_PERMISSIONS_OPTION_DENY_ACCESS .....	161
IS_PERMISSIONS_OPTION_NO_APPLYDOWN .....	161
IS_PERMISSIONS_TYPE_FILE .....	161
IS_PERMISSIONS_TYPE_FOLDER .....	162
IS_PERMISSIONS_TYPE_REGISTRY .....	162

ISDIFX_OPTION_DONT_ASSOCIATE .....	162
ISDIFX_OPTION_DONT_RESOLVE_TEXTSUBS .....	162
ISDIFX_OPTION_LOG_IN_DRIVER_PACKAGE_PATH .....	162
ISDIFX_OPTION_NO_REPAIR .....	163
ISERR_GEN_FAILURE .....	163
ISERR_SUCCESS .....	163
ISLANG_AFRIKAANS .....	163
ISLANG_AFRIKAANS_STANDARD .....	163
ISLANG_ALBANIAN .....	164
ISLANG_ALBANIAN_STANDARD .....	164
ISLANG_ALL .....	164
ISLANG_ARABIC .....	164
ISLANG_ARABIC_ALGERIA .....	164
ISLANG_ARABIC_BAHRAIN .....	164
ISLANG_ARABIC_EGYPT .....	164
ISLANG_ARABIC_IRAQ .....	164
ISLANG_ARABIC_JORDAN .....	165
ISLANG_ARABIC_KUWAIT .....	165
ISLANG_ARABIC_LEBANON .....	165
ISLANG_ARABIC_LIBYA .....	165
ISLANG_ARABIC_MOROCCO .....	165
ISLANG_ARABIC_OMAN .....	165
ISLANG_ARABIC_QATAR .....	165
ISLANG_ARABIC_SAUDIARABIA .....	165
ISLANG_ARABIC_SYRIA .....	166
ISLANG_ARABIC_TUNISIA .....	166
ISLANG_ARABIC_UAE .....	166
ISLANG_ARABIC_YEMEN .....	166
ISLANG_BASQUE .....	166
ISLANG_BASQUE_STANDARD .....	166
ISLANG_BELARUSIAN .....	166
ISLANG_BELARUSIAN_STANDARD .....	166
ISLANG_BULGARIAN .....	167
ISLANG_BULGARIAN_STANDARD .....	167
ISLANG_CATALAN .....	167
ISLANG_CATALAN_STANDARD .....	167
ISLANG_CHINESE .....	167
ISLANG_CHINESE_HONGKONG .....	167
ISLANG_CHINESE_PRC .....	167
ISLANG_CHINESE_SINGAPORE .....	167
ISLANG_CHINESE_TAIWAN .....	168
ISLANG_CROATIAN .....	168
ISLANG_CROATIAN_STANDARD .....	168
ISLANG_CZECH .....	168
ISLANG_CZECH_STANDARD .....	168
ISLANG_DANISH .....	168

ISLANG_DANISH_STANDARD .....	168
ISLANG_DUTCH .....	168
ISLANG_DUTCH_BELGIAN .....	169
ISLANG_DUTCH_STANDARD .....	169
ISLANG_ENGLISH .....	169
ISLANG_ENGLISH_AUSTRALIAN .....	169
ISLANG_ENGLISH_BELIZE .....	169
ISLANG_ENGLISH_CANADIAN .....	169
ISLANG_ENGLISH_CARIBBEAN .....	169
ISLANG_ENGLISH_IRELAND .....	169
ISLANG_ENGLISH_JAMAICA .....	170
ISLANG_ENGLISH_NEWZEALAND .....	170
ISLANG_ENGLISH_SOUTHAFRICA .....	170
ISLANG_ENGLISH_TRINIDAD .....	170
ISLANG_ENGLISH_UNITEDKINGDOM .....	170
ISLANG_ENGLISH_UNITEDSTATES .....	170
ISLANG_ESTONIAN .....	170
ISLANG_ESTONIAN_STANDARD .....	170
ISLANG_FAEROESE .....	171
ISLANG_FAEROESE_STANDARD .....	171
ISLANG_FARSI .....	171
ISLANG_FARSI_STANDARD .....	171
ISLANG_FINNISH .....	171
ISLANG_FINNISH_STANDARD .....	171
ISLANG_FRENCH .....	171
ISLANG_FRENCH_BELGIAN .....	171
ISLANG_FRENCH_CANADIAN .....	172
ISLANG_FRENCH_LUXEMBOURG .....	172
ISLANG_FRENCH_STANDARD .....	172
ISLANG_FRENCH_SWISS .....	172
ISLANG_GERMAN .....	172
ISLANG_GERMAN_AUSTRIAN .....	172
ISLANG_GERMAN_LIECHTENSTEIN .....	172
ISLANG_GERMAN_LUXEMBOURG .....	172
ISLANG_GERMAN_STANDARD .....	173
ISLANG_GERMAN_SWISS .....	173
ISLANG_GREEK .....	173
ISLANG_GREEK_STANDARD .....	173
ISLANG_HEBREW .....	173
ISLANG_HEBREW_STANDARD .....	173
ISLANG_HUNGARIAN .....	173
ISLANG_HUNGARIAN_STANDARD .....	173
ISLANG_ICELANDIC .....	174
ISLANG_ICELANDIC_STANDARD .....	174
ISLANG_INDONESIAN .....	174
ISLANG_INDONESIAN_STANDARD .....	174

ISLANG_ITALIAN . . . . .	174
ISLANG_ITALIAN_STANDARD . . . . .	174
ISLANG_ITALIAN_SWISS . . . . .	174
ISLANG_JAPANESE . . . . .	174
ISLANG_JAPANESE_STANDARD . . . . .	175
ISLANG_KOREAN . . . . .	175
ISLANG_KOREAN_JOHAB . . . . .	175
ISLANG_KOREAN_STANDARD . . . . .	175
ISLANG_LATVIAN . . . . .	175
ISLANG_LATVIAN_STANDARD . . . . .	175
ISLANG_LITHUANIAN . . . . .	175
ISLANG_LITHUANIAN_STANDARD . . . . .	175
ISLANG_NORWEGIAN . . . . .	176
ISLANG_NORWEGIAN_BOKMAL . . . . .	176
ISLANG_NORWEGIAN_NYNORSK . . . . .	176
ISLANG_POLISH . . . . .	176
ISLANG_POLISH_STANDARD . . . . .	176
ISLANG_PORTUGUESE . . . . .	176
ISLANG_PORTUGUESE_BRAZILIAN . . . . .	176
ISLANG_PORTUGUESE_STANDARD . . . . .	176
ISLANG_ROMANIAN . . . . .	177
ISLANG_ROMANIAN_STANDARD . . . . .	177
ISLANG_RUSSIAN . . . . .	177
ISLANG_RUSSIAN_STANDARD . . . . .	177
ISLANG_SERBIAN_CYRILLIC . . . . .	177
ISLANG_SERBIAN_LATIN . . . . .	177
ISLANG_SLOVAK . . . . .	177
ISLANG_SLOVAK_STANDARD . . . . .	177
ISLANG_SLOVENIAN . . . . .	178
ISLANG_SLOVENIAN_STANDARD . . . . .	178
ISLANG_SPANISH . . . . .	178
ISLANG_SPANISH_ARGENTINA . . . . .	178
ISLANG_SPANISH_BOLIVIA . . . . .	178
ISLANG_SPANISH_CHILE . . . . .	178
ISLANG_SPANISH_COLOMBIA . . . . .	178
ISLANG_SPANISH_COSTARICA . . . . .	178
ISLANG_SPANISH_DOMINICANREPUBLIC . . . . .	179
ISLANG_SPANISH_ECUADOR . . . . .	179
ISLANG_SPANISH_ELSALVADOR . . . . .	179
ISLANG_SPANISH_GUATEMALA . . . . .	179
ISLANG_SPANISH_HONDURAS . . . . .	179
ISLANG_SPANISH_MEXICAN . . . . .	179
ISLANG_SPANISH_MODERNSORT . . . . .	179
ISLANG_SPANISH_NICARAGUA . . . . .	179
ISLANG_SPANISH_PANAMA . . . . .	180
ISLANG_SPANISH_PARAGUAY . . . . .	180

ISLANG_SPANISH_PERU .....	180
ISLANG_SPANISH_PUERTORICO .....	180
ISLANG_SPANISH_TRADITIONALSORT .....	180
ISLANG_SPANISH_URUGUAY .....	180
ISLANG_SPANISH_VENEZUELA .....	180
ISLANG_SWEDISH .....	180
ISLANG_SWEDISH_FINLAND .....	181
ISLANG_SWEDISH_STANDARD .....	181
ISLANG_THAI .....	181
ISLANG_THAI_STANDARD .....	181
ISLANG_TURKISH .....	181
ISLANG_TURKISH_STANDARD .....	181
ISLANG_UKRAINIAN .....	181
ISLANG_UKRAINIAN_STANDARD .....	181
ISLANG_VIETNAMESE .....	182
ISLANG_VIETNAMESE_STANDARD .....	182
ISOSL_ALL .....	182
ISOSL_SUPPORTED .....	182
ISOSL_WIN7_SERVER2008R2 .....	182
ISOSL_WIN8 .....	182
ISOSL_WIN81 .....	183
ISOSL_WIN10 .....	183
ISOSL_WINSERVER2003 .....	183
ISOSL_WINVISTA .....	183
ISOSL_WINVISTA_SERVER2008 .....	183
ISOSL_WINXP .....	184
ISOS_ST_ALL .....	184
ISOS_ST_BACKOFFICE .....	184
ISOS_ST_DATACENTER .....	184
ISOS_ST_ENTERPRISE .....	185
ISOS_ST_PROC_ARCH_32 .....	185
ISOS_ST_PROC_ARCH_AMD64 .....	185
ISOS_ST_PROC_ARCH_IA64 .....	185
ISOS_ST_SERVER .....	186
ISOS_ST_SERVER2003_R2 .....	186
ISOS_ST_SMALLBUSINESS .....	186
ISOS_ST_SMALLBUSINESS_RESTRICTED .....	187
ISOS_ST_TERMINAL .....	187
ISOS_ST_WORKSTATION .....	187
ISOS_ST_XP_HOME .....	187
ISOS_ST_XP_PRO .....	188
ISUS_AGENT_FEATURE .....	188
ISUS_MAIN_FEATURE .....	188
ISUS_TEXTSUB_HOST .....	188
ISUS_TEXTSUB_INTERVAL .....	188
ISUS_TEXTSUB_LANGUAGE .....	189

ISUS_TEXTSUB_LOGO .....	189
ISUS_TEXTSUB_MANAGER .....	189
ISUS_TEXTSUB_VERSION.....	189
ISUS_UPDATEMANAGER_FEATURE.....	189
IS_386.....	190
IS_486.....	190
IS_ALPHA .....	190
IS_CDROM.....	190
IS_EGA .....	190
IS_FIXED.....	191
IS_FOLDER.....	191
IS_ITEM.....	191
IS_PENTIUM.....	191
IS_REMOTE .....	191
IS_REMOVABLE.....	192
IS_SVGA .....	192
IS_UNKNOWN .....	192
IS_UVGA.....	192
IS_VGA .....	192
IS_WINDOWS .....	193
IS_WINDOWS9X.....	193
IS_WINDOWSNT.....	193
IS_XVGA.....	193
KBYTES.....	193
KEY_CREATE_LINK .....	194
KEY_CREATE_SUB_KEY.....	194
KEY_ENUMERATE_SUB_KEYS .....	194
KEY_NOTIFY.....	194
KEY_QUERY_VALUE .....	194
KEY_SET_VALUE .....	195
LAAW_OPTION_CHANGEDIRECTORY.....	195
LAAW_OPTION_FIXUP_PROGRAM .....	195
LAAW_OPTION_HIDDEN.....	195
LAAW_OPTION_MAXIMIZED .....	195
LAAW_OPTION_MINIMIZED.....	196
LAAW_OPTION_NO_CHANGEDIRECTORY.....	196
LAAW_OPTION_NOWAIT .....	196
LAAW_OPTION_SET_BATCH_INSTALL .....	196
LAAW_OPTION_SHOW_HOURLASS.....	197
LAAW_OPTION_USE_CALLBACK .....	197
LAAW_OPTION_USE_SHELLEXECUTE .....	197
LAAW_OPTION_WAIT .....	198
LAAW_OPTION_WAIT_INCL_CHILD.....	198
LANGUAGE_SUPPORTED .....	198
LANGUAGE .....	198
LESS_THAN .....	199



LINE_NUMBER .....	199
LISTBOX_ENTER .....	199
LISTBOX_SELECT .....	199
LISTFIRST .....	199
LISTLAST .....	200
LISTNEXT .....	200
LISTPREV .....	200
LIST_NULL .....	200
LOCKEDFILE .....	200
LOGGING .....	201
LOWER_LEFT .....	201
LOWER_RIGHT .....	201
LWTF_OPTION_APPEND_TO_FILE .....	201
LWTF_OPTION_WRITE_AS_ANSI .....	202
LWTF_OPTION_WRITE_AS_UNICODE .....	202
MAGENTA .....	202
MATH_COPROCESSOR .....	202
MBYTES .....	203
MEDIA_FIELD_ADDREMOVE_NOMODIFY .....	203
MEDIA_FIELD_ADDREMOVE_NOREMOVE .....	203
MEDIA_FIELD_COMPANY_NAME .....	203
MEDIA_FIELD_MEDIA_FLAGS .....	204
MEDIA_FIELD_PREVIOUS_VERSIONS .....	204
MEDIA_FIELD_PRODUCT_COMMENTS .....	204
MEDIA_FIELD_PRODUCT_EXE .....	204
MEDIA_FIELD_PRODUCT_ICON .....	205
MEDIA_FIELD_PRODUCT_NAME .....	205
MEDIA_FIELD_PRODUCT_README .....	205
MEDIA_FIELD_PRODUCT_SUPPORT_CONTACT .....	206
MEDIA_FIELD_PRODUCT_SUPPORT_PHONE .....	206
MEDIA_FIELD_PRODUCT_SUPPORT_URL .....	206
MEDIA_FIELD_PRODUCT_UPDATE_URL .....	206
MEDIA_FIELD_PRODUCT_URL .....	207
MEDIA_FIELD_PRODUCT_VERSION .....	207
MEDIA_FIELD_TARGETDIR .....	207
MEDIA_FLAG_FORMAT_DIFFERENTIAL .....	208
MEDIA_FLAG_FORMAT_PATCH .....	208
MEDIA_FLAG_UPDATEMODE_SUPPORTED .....	208
MEDIA_PASSWORD_KEY .....	208
METAFILE .....	209
MMEDIA_AVI .....	209
MMEDIA_MIDI .....	209
MMEDIA_PLAYASYNCH .....	209
MMEDIA_PLAYCONTINUOUS .....	210
MMEDIA_PLAYSYNCH .....	210
MMEDIA_STOP .....	210

MMEDIA_SWF .....	210
MMEDIA_WAVE .....	210
MODIFY .....	211
NEXT .....	211
NEXTBUTTON .....	212
NO .....	212
NONEXCLUSIVE .....	212
NORMALMODE .....	213
NORMAL_PRIORITY_CLASS .....	213
NOSET .....	213
NOTEXISTS .....	213
NO_SUBDIR .....	213
NULL .....	214
NUMBERLIST .....	214
OFF .....	214
OK .....	214
ON .....	215
ONLYDIR .....	215
OTHER_FAILURE .....	215
OUT_OF_DISK_SPACE .....	215
PARALLEL .....	216
PARTIAL .....	216
PATH .....	216
PATH_EXISTS .....	216
PCRESTORE .....	217
PERSONAL .....	217
READ_CONTROL .....	217
REBOOTED .....	217
RECORDMODE .....	218
RED .....	218
REGDBREMOTEREGCONNECTED .....	218
REGDB_APPPATH .....	218
REGDB_APPPATH_DEFAULT .....	218
REGDB_BINARY .....	219
REGDB_ERR_CONNECTIONEXISTS .....	219
REGDB_ERR_CORRUPTEDREGISTRY .....	219
REGDB_ERR_INITIALIZATION .....	219
REGDB_ERR_INVALIDHANDLE .....	220
REGDB_ERR_INVALIDNAME .....	220
REGDB_KEYPATH_APPPATHS .....	220
REGDB_KEYPATH_DOTNET_10 .....	220
REGDB_KEYPATH_DOTNET_11 .....	220
REGDB_KEYPATH_DOTNET_20 .....	221
REGDB_KEYPATH_DOTNET_30 .....	221
REGDB_KEYPATH_DOTNET_30_SP .....	221
REGDB_KEYPATH_DOTNET_35 .....	222

REGDB_KEYPATH_DOTNET_40_CLIENT .....	222
REGDB_KEYPATH_DOTNET_40_FULL .....	222
REGDB_KEYPATH_ISUNINSTINFO .....	223
REGDB_KEYPATH_RUN .....	223
REGDB_KEYPATH_RUNONCE .....	223
REGDB_KEYPATH_RUNONCEEX .....	223
REGDB_KEYPATH_SHAREDDLLS .....	223
REGDB_KEYPATH_UNINSTALL .....	224
REGDB_KEYPATH_WINCURRVER .....	224
REGDB_KEYPATH_WINCURRVER_AUTO .....	224
REGDB_KEYPATH_WINNTCURRVER .....	224
REGDB_KEYS .....	224
REGDB_NAMES .....	224
REGDB_NUMBER .....	225
REGDB_STRING .....	225
REGDB_STRING_EXPAND .....	225
REGDB_STRING_MULTI .....	225
REGDB_UNINSTALL_COMMENTS .....	226
REGDB_UNINSTALL_CONTACT .....	226
REGDB_UNINSTALL_DISPLAYICON .....	226
REGDB_UNINSTALL_DISPLAY_VERSION .....	227
REGDB_UNINSTALL_HELPLINK .....	227
REGDB_UNINSTALL_HELPTELEPHONE .....	227
REGDB_UNINSTALL_INSTALLDATE .....	227
REGDB_UNINSTALL_INSTALLLOC .....	228
REGDB_UNINSTALL_INSTALLSOURCE .....	228
REGDB_UNINSTALL_LANGUAGE .....	228
REGDB_UNINSTALL_LOGFILE .....	229
REGDB_UNINSTALL_MAINT_OPTION .....	229
REGDB_UNINSTALL_MAJOR_VERSION .....	229
REGDB_UNINSTALL_MAJOR_VERSION_OLD .....	230
REGDB_UNINSTALL_MINOR_VERSION .....	230
REGDB_UNINSTALL_MINOR_VERSION_OLD .....	230
REGDB_UNINSTALL_MODIFYPATH .....	231
REGDB_UNINSTALL_NAME .....	231
REGDB_UNINSTALL_NOMODIFY .....	231
REGDB_UNINSTALL_NOREMOVE .....	231
REGDB_UNINSTALL_NOREPAIR .....	232
REGDB_UNINSTALL_PRODUCTGUID .....	232
REGDB_UNINSTALL_PRODUCTID .....	232
REGDB_UNINSTALL_PUBLISHER .....	232
REGDB_UNINSTALL_README .....	233
REGDB_UNINSTALL_REGCOMPANY .....	233
REGDB_UNINSTALL_REGOWNER .....	233
REGDB_UNINSTALL_STRING .....	233
REGDB_UNINSTALL_SYSTEMCOMPONENT .....	234

REGDB_UNINSTALL_URLINFOABOUT.....	234
REGDB_UNINSTALL_URLUPDATEINFO.....	234
REGDB_UNINSTALL_VERSION.....	235
REGDB_VALUENAME_APPPATH.....	235
REGDB_VALUENAME_APPPATHDEFAULT.....	235
REGDB_VALUENAME_INSTALL.....	235
REGDB_VALUENAME_INSTALLSUCCESS.....	235
REGDB_VALUENAME_SP.....	235
REGDB_VALUENAME_UNINSTALL_COMMENTS.....	236
REGDB_VALUENAME_UNINSTALL_CONTACT.....	236
REGDB_VALUENAME_UNINSTALL_DISPLAYICON.....	236
REGDB_VALUENAME_UNINSTALL_DISPLAYNAME.....	236
REGDB_VALUENAME_UNINSTALL_DISPLAYVERSION.....	236
REGDB_VALUENAME_UNINSTALL_HELPLINK.....	236
REGDB_VALUENAME_UNINSTALL_HELPTELEPHONE.....	236
REGDB_VALUENAME_UNINSTALL_INSTALLDATE.....	237
REGDB_VALUENAME_UNINSTALL_INSTALLLOCATION.....	237
REGDB_VALUENAME_UNINSTALL_INSTALLSOURCE.....	237
REGDB_VALUENAME_UNINSTALL_LANGUAGE.....	237
REGDB_VALUENAME_UNINSTALL_LOGFILE.....	237
REGDB_VALUENAME_UNINSTALL_LOGMODE.....	237
REGDB_VALUENAME_UNINSTALL_MAJORVERSION.....	237
REGDB_VALUENAME_UNINSTALL_MAJORVERSION_OLD.....	238
REGDB_VALUENAME_UNINSTALL_MINORVERSION.....	238
REGDB_VALUENAME_UNINSTALL_MINORVERSION_OLD.....	238
REGDB_VALUENAME_UNINSTALL_MODIFYPATH.....	238
REGDB_VALUENAME_UNINSTALL_NOMODIFY.....	238
REGDB_VALUENAME_UNINSTALL_NOREMOVE.....	238
REGDB_VALUENAME_UNINSTALL_NOREPAIR.....	238
REGDB_VALUENAME_UNINSTALL_PRODUCTGUID.....	239
REGDB_VALUENAME_UNINSTALL_PRODUCTID.....	239
REGDB_VALUENAME_UNINSTALL_PUBLISHER.....	239
REGDB_VALUENAME_UNINSTALL_README.....	239
REGDB_VALUENAME_UNINSTALL_REGCOMPANY.....	239
REGDB_VALUENAME_UNINSTALL_REGOWNER.....	239
REGDB_VALUENAME_UNINSTALL_SYSTEMCOMPONENT.....	239
REGDB_VALUENAME_UNINSTALL_UNINSTALLSTRING.....	240
REGDB_VALUENAME_UNINSTALL_URLINFOABOUT.....	240
REGDB_VALUENAME_UNINSTALL_URLUPDATEINFO.....	240
REGDB_VALUENAME_UNINSTALL_VERSION.....	240
REGDB_VALUENAME_UNINSTALLKEY.....	240
REGDB_VALUENAME_WINCURRVER_REGORGANIZATION.....	240
REGDB_VALUENAME_WINCURRVER_REGOWNER.....	241
REGDB_WINCURRVER_REGORGANIZATION.....	241
REGDB_WINCURRVER_REGOWNER.....	241
REGFONT_OPTION_DEFAULT.....	241

REGFONT_OPTION_DONTBROADCASTFONTCHANGEMSG .....	241
REGFONT_OPTION_DONTUPDATEREGISTRY .....	242
REGISTRYFUNCTIONS_USETEXTSUBS .....	242
REMOTE_DRIVE .....	242
REMOVE .....	242
REMOVEABLE_DRIVE .....	242
REMOVEALL .....	243
REPAIR .....	243
REPLACE .....	243
RESET .....	243
RESTART .....	244
ROOT .....	244
RUN_MAXIMIZED .....	244
RUN_MINIMIZED .....	244
SELECTFOLDER .....	245
SELFREGISTER .....	245
SELFREGISTERBATCH .....	245
SELFREGISTRATIONPROCESS .....	245
SERIAL .....	245
SERVICE_ADAPTER .....	246
SERVICE_ALL_ACCESS .....	246
SERVICE_AUTO_START .....	246
SERVICE_BOOT_START .....	246
SERVICE_CHANGE_CONFIG .....	247
SERVICE_CONTINUE_PENDING .....	247
SERVICE_DEMAND_START .....	247
SERVICE_DISABLED .....	248
SERVICE_ENUMERATE_DEPENDENTS .....	248
SERVICE_ERROR_CRITICAL .....	248
SERVICE_ERROR_IGNORE .....	248
SERVICE_ERROR_NORMAL .....	249
SERVICE_ERROR_SEVERE .....	249
SERVICE_FILE_SYSTEM_DRIVER .....	249
SERVICE_FLAG_DIFX_32 .....	250
SERVICE_FLAG_DIFX_AMD64 .....	250
SERVICE_FLAG_DIFX_IA64 .....	250
SERVICE_FLAG_ISFONTREG .....	250
SERVICE_INTERACTIVE_PROCESS .....	251
SERVICE_INTERROGATE .....	251
SERVICE_ISFONTREG .....	251
SERVICE_ISUPDATE .....	251
SERVICE_KERNEL_DRIVER .....	252
SERVICE_PAUSED .....	252
SERVICE_PAUSE_CONTINUE .....	252
SERVICE_PAUSE_PENDING .....	252
SERVICE_QUERY_CONFIG .....	253

SERVICE_QUERY_STATUS .....	253
SERVICE_RECOGNIZER_DRIVER .....	253
SERVICE_RUNNING .....	254
SERVICE_START .....	254
SERVICE_START_PENDING .....	254
SERVICE_STOP .....	254
SERVICE_STOPPED .....	255
SERVICE_STOP_PENDING .....	255
SERVICE_SYSTEM_START .....	255
SERVICE_USER_DEFINED_CONTROL .....	256
SERVICE_WIN32_OWN_PROCESS .....	256
SERVICE_WIN32_SHARE_PROCESS .....	256
SETUPTYPE .....	256
SETUPTYPE_INFO_DESCRIPTION .....	257
SETUPTYPE_INFO_DISPLAYNAME .....	257
SETUPTYPE_STR_COMPACT .....	257
SETUPTYPE_STR_COMPLETE .....	257
SETUPTYPE_STR_CUSTOM .....	257
SETUPTYPE_STR_TYPICAL .....	258
SETUP_PACKAGE .....	258
SEVERE .....	258
SHAREDFILE .....	258
SILENTMODE .....	259
SKIN_LOADED .....	259
SQL_BATCH_INSTALL .....	259
SQL_BATCH_UNINSTALL .....	259
SQL_BROWSE_ALIAS .....	259
SQL_BROWSE_ALL .....	260
SQL_BROWSE_LOCAL .....	260
SQL_BROWSE_REMOTE .....	260
SQL_ERROR_GET_SCHEMA_VERSION .....	260
SQL_ERROR_SCRIPT_COMMAND_ERROR .....	260
SQL_ERROR_SCRIPT_CONNECTION_NOT_OPEN .....	261
SQL_ERROR_SCRIPT_UNABLE_OPEN_FILE .....	261
SQL_ERROR_SET_SCHEMA_VERSION .....	261
SRCINSTALLDIR .....	261
SRCTARGETDIR .....	262
SSP_PROPERTY_NO_NEW_INSTALL_HIGHLIGHT .....	262
SSP_PROPERTY_NO_STARTSCREEN_PIN .....	262
SSP_PROPERTY_PREVENT_PINNING .....	262
STANDARD_RIGHTS_ALL .....	263
STANDARD_RIGHTS_EXECUTE .....	263
STANDARD_RIGHTS_READ .....	263
STANDARD_RIGHTS_REQUIRED .....	263
STANDARD_RIGHTS_WRITE .....	263
STATUS .....	264

STATUSBAR.....	264
STATUSBBD.....	264
STATUSDLG.....	264
STATUSEX.....	265
STATUSOLD.....	265
STRINGLIST.....	265
STYLE_BOLD.....	265
STYLE_ITALIC.....	265
STYLE_NORMAL.....	266
STYLE_SHADOW.....	266
STYLE_UNDERLINE.....	266
SW_MAXIMIZE.....	266
SW_MINIMIZE.....	267
SW_RESTORE.....	267
SW_SHOW.....	267
SYNCHRONIZE.....	267
SYS_BOOTMACHINE.....	267
SYSTEM_DPI.....	268
SYSTEM_DPI_SCALING.....	268
TBYTES.....	268
TILED.....	268
TIME.....	268
TRUE.....	269
TTFONTFILEINFO_FONTTITLE.....	269
TYPICAL.....	270
UPDATE_SERVICE_INSTALL.....	270
UPDATESERVICECOMPONENT.....	270
UPPER_LEFT.....	270
UPPER_RIGHT.....	270
URL.....	271
USER_ADMINISTRATOR.....	271
USER_INADMINGROUP.....	271
USER_POWERUSER.....	271
USE_LOADED_SKIN.....	272
VALID_PATH.....	272
VERSION_COMPARE_RESULT_NEWER.....	272
VERSION_COMPARE_RESULT_NEWER_NOT_SUPPORTED.....	272
VERSION_COMPARE_RESULT_NOT_INSTALLED.....	273
VERSION_COMPARE_RESULT_OLDER.....	273
VERSION_COMPARE_RESULT_SAME.....	273
VERSION_PREVIOUS_VERSION_DELIMITER.....	273
VER_DLL_NOT_FOUND.....	274
VER_UPDATE_ALWAYS.....	274
VER_UPDATE_COND.....	274
VIDEO.....	274
VIRTUAL_MACHINE_TYPE.....	275

VOLUMELABEL .....	275
WARNING .....	275
WEB_BASED_SETUP .....	275
WELCOME .....	275
WHITE .....	276
WILL_REBOOT .....	276
WINDOWS_SHARED .....	276
WINMAJOR .....	276
WINMINOR .....	276
WOW64FSREDIRECTION .....	277
WRITE_DAC .....	277
WRITE_OWNER .....	277
YELLOW .....	277
YES .....	278
_MAX_PATH .....	278
<b>Predefined Script Variables .....</b>	<b>279</b>
__FILE__ .....	279
__LINE__ .....	279
BASICMSI .....	280
INSTALLSCRIPTMSI .....	280
INSTALLSCRIPTMSIEUI .....	280
ISUS_PRODUCT_CODE .....	281
SERVICE_IS_PARAMS .....	281
SERVICE_IS_STATUS .....	283
SUITE_HOSTED .....	286
<b>Data Types and Predefined Structures .....</b>	<b>289</b>
Arrays .....	294
Constant Data .....	294
Data Structures .....	295
Language IDs .....	298
Pointers .....	299
<b>Variable Data .....</b>	<b>303</b>
Global vs. Local Variables .....	303
String Variables .....	305
String Indexing .....	306
String Size and Autosize .....	306
System Variables .....	307
ADDREMOVE .....	312
ADDREMOVE_COMBINEDBUTTON .....	312
ADDREMOVE_HIDECHANGEOPTION .....	313
ADDREMOVE_HIDEREMOVEOPTION .....	313
ADDREMOVE_STRING_REMOVEONLY .....	313
ADDREMOVE_SYSTEMCOMPONENT .....	313
ALLUSERS .....	314
<i>How an InstallScript Installation Works by Default Depending on ALLUSERS</i> .....	319
ADMINUSER .....	319



BATCH_INSTALL.....	320
CMDLINE.....	320
COMMONFILES.....	321
COMMONFILES64.....	322
DISK1SETUPEXENAME.....	322
DISK1TARGET.....	322
ENABLED_ISERVICES.....	322
ENGINECOMMONDIR.....	322
ENGINEDIR.....	323
ERRORFILENAME.....	323
FOLDER_APPDATA.....	323
FOLDER_APPLICATIONS.....	324
FOLDER_APPLICATIONS64.....	324
FOLDER_COMMON_APPDATA.....	324
FOLDER_DESKTOP.....	324
FOLDER_DOTNET_10.....	325
FOLDER_DOTNET_11.....	325
FOLDER_DOTNET_20.....	325
FOLDER_DOTNET_30.....	325
FOLDER_DOTNET_35.....	325
FOLDER_DOTNET_40.....	325
FOLDER_FONTS.....	326
FOLDER_LOCAL_APPDATA.....	326
FOLDER_PERSONAL.....	326
FOLDER_PROGRAMS.....	327
FOLDER_STARTMENU.....	327
FOLDER_STARTUP.....	327
FOLDER_TEMP.....	327
HKEYCURRENTROOTKEY.....	328
HKEY_USER_SELECTABLE_AUTO.....	328
IFX_COMPANY_NAME.....	328
IFX_DISK1INSTALLED.....	329
IFX_INITIALIZED.....	329
IFX_INSTALLED_DISPLAY_VERSION.....	329
IFX_INSTALLED_VERSION.....	329
IFX_KEYPATH_PRODUCT_INFO.....	329
IFX_MULTI_INSTANCE_SUFFIX.....	330
IFX_PRODUCT_COMMENTS.....	330
IFX_PRODUCT_DISPLAY_NAME.....	330
IFX_PRODUCT_DISPLAY_VERSION.....	330
IFX_PRODUCT_ICON.....	331
IFX_PRODUCT_KEY.....	331
IFX_PRODUCT_NAME.....	331
IFX_PRODUCT_README.....	331
IFX_PRODUCT_REGISTEREDCOMPANY.....	331
IFX_PRODUCT_REGISTEREDOWNER.....	332

IFX_PRODUCT_REGISTEREDSERIALNUM .....	332
IFX_PRODUCT_SUPPORT_CONTACT .....	332
IFX_PRODUCT_SUPPORT_PHONE .....	333
IFX_PRODUCT_SUPPORT_URL .....	333
IFX_PRODUCT_UPDATE_URL .....	333
IFX_PRODUCT_URL .....	333
IFX_PRODUCT_VERSION .....	334
IFX_SETUP_TITLE .....	334
IFX_SUPPORTED_VERSIONS .....	334
INFOFILENAME .....	334
INSTALLDIR .....	334
INSTANCE_GUID .....	335
ISDIFXAPPID .....	335
ISMSI_HANDLE .....	335
IS_NULLSTR_PTR .....	336
ISRES .....	336
ISUSER .....	336
ISVERSION .....	337
LAAW_PARAMETERS .....	337
LAAW_PROCESS_INFORMATION .....	339
LAAW_SHELLEXECUTEINFO .....	339
LAAW_SHELLEXECUTEVERB .....	340
LAAW_STARTUPINFO .....	340
MAINTENANCE .....	343
MAINT_OPTION .....	343
MEDIA .....	343
MODE .....	344
MSI_TARGETDIR .....	344
MULTI_INSTANCE_COUNT .....	344
PACKAGE_LOCATION .....	345
PRODUCT_GUID .....	345
PRODUCT_INSTALLED .....	345
PROGRAMFILES .....	345
PROGRAMFILES64 .....	346
REGDB_OPTIONS .....	346
REINSTALLMODE .....	348
REMOVEALLMODE .....	348
REMOVEONLY .....	348
SELECTED_LANGUAGE .....	349
SHAREDSUPPORTDIR .....	349
SHELL_OBJECT_FOLDER .....	349
SHOW_PASSWORD_DIALOG .....	350
SRCDIR .....	350
SRCDISK .....	350
SUPPORTDIR .....	350
SYSINFO .....	351

SYSPROCESSORINFO .....	357
TARGETDIR .....	359
TARGETDISK .....	359
UNINST .....	359
UNINSTALLKEY .....	360
UNINSTALL_DISPLAYNAME .....	360
UNINSTALL_STRING .....	361
UPDATEMODE .....	361
WINDIR .....	361
WINDISK .....	361
WINSYSDIR .....	362
WINSYSDIR64 .....	362
WINSYSDISK .....	363
<b>Preprocessor Directives .....</b>	<b>365</b>
#define .....	366
#elif .....	367
#error .....	367
#if .. #else .. #endif .....	367
#ifdef and #ifndef .....	368
#include .....	369
#undef .....	370
#warning .....	370
<b>Event Handlers .....</b>	<b>371</b>
Event Handler Index .....	373
Component Event Handlers .....	378
Global Event Handlers .....	378
Initialization Handlers .....	378
OnCheckMediaPassword .....	379
OnFilterComponents .....	379
OnSetTARGETDIR .....	380
OnSetUpdateMode .....	381
Before Move Data Handlers .....	381
OnAppSearch .....	384
OnBegin .....	385
OnCCPSearch .....	386
OnFirstUIBefore .....	386
OnIISInitialize .....	386
OnMaintUIBefore .....	387
OnSQLLogin .....	387
OnSQLServerInitialize .....	387
OnSQLServerInitializeMaint .....	388
OnSuiteInstallBefore .....	388
OnSuiteMaintBefore .....	388
OnSuiteUpdateBefore .....	388
OnUpdateUIBefore .....	389
OnXMLInitialize .....	389

Move Data Handlers .....	390
OnCustomizeUninstInfo .....	392
OnGeneratedMSIScript .....	392
OnGeneratingMSIScript .....	392
OnISComponentInstalled .....	392
OnISVRootUninstalling .....	393
OnInstalledFile .....	393
OnInstalledFontFile .....	393
OnInstallFilesActionAfter .....	394
OnInstallFilesActionBefore .....	394
OnInstallingFile .....	394
OnMoved .....	395
OnMoveData .....	395
OnMoving .....	395
OnNetApiCreateUserAccount .....	396
OnSQLBatchScripts .....	396
OnSQLComponentInstalled .....	396
OnSQLComponentUninstalled .....	397
OnUninstalledFile .....	397
OnUninstallingFile .....	397
OnUninstallingDIFxDriverFile .....	398
OnUninstallingFontFile .....	398
OnXMLComponentInstalled .....	399
OnXMLComponentUninstalling .....	399
After Data Move Handlers .....	400
OnEnd .....	402
OnFirstUIAfter .....	402
OnISUninitialize .....	402
OnMaintUIAfter .....	403
OnSuiteInstallAfter .....	403
OnSuiteMaintAfter .....	403
OnSuiteUpdateAfter .....	403
OnUpdateUIAfter .....	404
OnXMLUninitialize .....	404
Feature Event Handlers .....	404
OnInstalled .....	405
OnInstalling .....	406
OnUnInstalled .....	406
OnUnInstalling .....	407
Miscellaneous Event Handlers .....	408
OnAbort .....	411
OnAdminInstallUIAfter .....	411
OnAdminInstallUIBefore .....	411
OnAdminPatchUIAfter .....	412
OnAdminPatchUIBefore .....	412
OnAdvertisementAfter .....	412

OnAdvertisementBefore	412
OnCanceling	412
OnComponentError	413
OnDIFxLogCallback	414
OnError	415
OnException	415
OnFileError	416
OnFileLocked	416
OnFileReadOnly	417
OnFilesInUse	418
OnHelp	419
OnInternetError	419
OnLaunchAppAndWaitCallback	420
OnLogonUserSetMsiProperties	421
OnMD5Error	421
OnMsiSilentInstall	423
OnNextDisk	423
OnOutOfDiskSpace	423
OnPatchUIAfter	423
OnPatchUIBefore	424
OnRebooted	424
OnRemovingSharedFile	424
OnResumeUIAfter	425
OnResumeUIBefore	426
OnRMFilesInUse	426
OnSelfRegistrationError	427
OnWarning	428
Advanced Event Handlers	429
OnShowUI	429
OnSuiteShowUI	431
OnUninstall	431
<b>Functions</b>	<b>433</b>
Using Built-In Functions	433
Built-In Functions by Category	435
Batch File Functions	437
Ez Batch File Functions	437
Advanced Batch File Functions	437
Component Functions	438
Configuration File Functions	438
Ez Config.sys File Functions	439
Advanced Configuration File Functions	439
Device Driver Functions	440
Dialog Functions	441
Dialog Customization Functions	450
Extensibility Functions	453
Feature Functions	454

<a href="#">Script-Created Feature Set vs. File Media Library</a>	457
<a href="#">File Media Library</a>	459
File and Folder Functions	459
FlexNet Connect Functions	461
Information Functions	463
Initialization File Functions	464
List Processing Functions	465
Log File Functions	466
Long File Name Functions	467
Miscellaneous Functions	467
Object Functions	468
Path Buffer Functions	469
Registry Functions	470
Service Functions	472
Shared and Locked File Functions	472
Shell Functions	473
Special Registry-Related Functions	475
SQL Functions	477
String Functions	482
Suite/Advanced UI and Advanced UI Interaction Functions	483
Text Substitutions	484
Uninstallation Functions	485
User Interface Functions	486
Version-Checking Functions	487
Windows Installer Functions	488
<a href="#">Windows Installer API Functions</a>	488
<a href="#">Windows Installer API Functions Example</a>	495
<b>Operators</b>	<b>497</b>
Address Operator (&)	497
Append to Path Operator (^)	498
Arithmetic Operators (+, -, *, /)	498
<a href="#">Arithmetic Operator Precedence</a>	498
<a href="#">Binary Arithmetic Operators</a>	499
<a href="#">Unary Arithmetic Operators</a>	500
Assignment Operator	500
Bit Operators (&,  , ^, ~, <<, >>)	501
BYREF Operator	502
BYVAL Operator	503
Concatenate Operator (+)	503
Indirection Operator (*)	504
Logical Operators (&&,   , !)	504
Member Operator (.)	505
Relational Operators (<, >, =, <=, >=, !=)	506
<a href="#">Relational Operator Precedence</a>	507
String Operators (^, +, %)	508
String Constant Operator (@)	508

Structure Pointer Operator (->)	509
Find String Operator (%)	509
<b>Objects and Object Handlers</b>	<b>511</b>
Objects	511
Err Object	511
Objects Object	513
Reboot Object	513
TextSub Object	514
Object Handlers	514
InitProperties	515
ReadProperties	515
WriteProperties	515
Exception Handling	516
<b>Built-In Functions (A-D)</b>	<b>519</b>
AddFolderIcon	519
AddFolderIcon Examples	522
AddFolderIcon Example 1	522
AddFolderIcon Example 2	523
AddFolderIcon Example 3	525
AddProfString	527
AddProfString Example	529
AdminAskPath	530
AdminAskPath Example	531
AskDestPath	531
AskDestPath Example	533
AskOptions	534
AskOptions Example	537
AskPath	539
AskPath Example	541
AskText	542
AskText Example	543
AskYesNo	544
AskYesNo Example	546
BatchAdd	547
BatchAdd Example	550
BatchDeleteEx	553
BatchDeleteEx Example	554
BatchFileLoad	555
BatchFileLoad Example	557
BatchFileSave	559
BatchFileSave Example	561
BatchFind	562
BatchFind Example	564
BatchGetFileName	565
BatchGetFileName Example	566
BatchMoveEx	567

BatchMoveEx Example .....	569
BatchSetFileName .....	570
BatchSetFileName Example .....	571
CalculateAndAddFileCost .....	572
CallDLLFx .....	573
CallDLLFx Example .....	574
ChangeDirectory .....	575
ChangeDirectory Example .....	576
CharReplace .....	577
CharReplace Example .....	578
CloseFile .....	579
CloseFile Example .....	580
CmdGetHwndDlg .....	581
CmdGetHwndDlg Example .....	582
CoCreateObject .....	584
CoCreateObjectDotNet .....	585
CoGetObject .....	585
CoGetObject Example .....	586
ConfigAdd .....	587
ConfigAdd Example .....	589
ConfigDelete .....	590
ConfigDelete Example .....	591
ConfigFileLoad .....	592
ConfigFileLoad Example .....	593
ConfigFileSave .....	595
ConfigFileSave Example .....	596
ConfigFind .....	598
ConfigFind Example .....	600
ConfigGetFileName .....	601
ConfigGetFileName Example .....	602
ConfigGetInt .....	603
ConfigGetInt Example .....	604
ConfigMove .....	606
ConfigMove Example .....	607
ConfigSetFileName .....	609
ConfigSetFileName Example .....	610
ConfigSetInt .....	611
ConfigSetInt Example .....	612
ConvertSizeToUnits .....	614
ConvertWinHighLowSizeToISHighLowSize .....	616
CopyBytes .....	616
CopyBytes Example .....	618
CopyCharArrayToIStringArray .....	619
CopyFile .....	620
CopyFile Example .....	623
CreateDir .....	624



<a href="#">CreateDir Example</a> .....	625
CreateFile .....	626
<a href="#">CreateFile Example</a> .....	628
CreateInstallationInfo .....	629
CreateObject .....	630
CreateProgramFolder .....	631
<a href="#">CreateProgramFolder Example</a> .....	632
CreateRegistrySet .....	633
<a href="#">CreateRegistrySet Example</a> .....	634
CreateShellObjects .....	635
<a href="#">CreateShellObjects Example</a> .....	636
CreateShortcut .....	637
<a href="#">CreateShortcut Examples</a> .....	642
<i>CreateShortcut Example 1</i> .....	643
<i>CreateShortcut Example 2</i> .....	644
<i>CreateShortcut Example 3</i> .....	645
CreateShortcutFolder .....	647
<a href="#">CreateShortcutFolder Example</a> .....	647
CtrlClear .....	648
<a href="#">CtrlClear Example</a> .....	649
CtrlDir .....	652
<a href="#">CtrlDir Example</a> .....	653
CtrlGetCurSel .....	656
<a href="#">CtrlGetCurSel Example</a> .....	656
CtrlGetDlgItem .....	659
CtrlGetMLEText .....	660
<a href="#">CtrlGetMLEText Example</a> .....	661
CtrlGetMultCurSel .....	664
<a href="#">CtrlGetMultCurSel Example</a> .....	665
CtrlGetState .....	668
<a href="#">CtrlGetState Example</a> .....	669
CtrlGetSubCommand .....	672
<a href="#">CtrlGetSubCommand Example</a> .....	672
CtrlGetText .....	675
<a href="#">CtrlGetText Example</a> .....	676
CtrlGetUrlForLinkClicked .....	678
<a href="#">CtrlGetUrlForLinkClicked Example</a> .....	679
CtrlPGroups .....	681
<a href="#">CtrlPGroups Example</a> .....	681
CtrlSelectText .....	684
<a href="#">CtrlSelectText Example</a> .....	684
CtrlSetCurSel .....	686
<a href="#">CtrlSetCurSel Example</a> .....	687
CtrlSetFont .....	690
<a href="#">CtrlSetFont Example</a> .....	691
CtrlSetList .....	694

CtrlSetList Example .....	695
CtrlSetMLEText .....	699
CtrlSetMLEText Example .....	700
CtrlSetMultCurSel .....	703
CtrlSetMultCurSel Example .....	704
CtrlSetState .....	707
CtrlSetState Example .....	708
CtrlSetText .....	711
CtrlSetText Example .....	712
DefineDialog .....	714
DefineDialog Example .....	718
DeinstallSetReference .....	720
DeinstallStart .....	720
Delay .....	721
Delay Example .....	721
DeleteCHARArray .....	722
DeleteDir .....	722
DeleteDir Example .....	723
DeleteFile .....	724
DeleteFile Example .....	725
DeleteFolderIcon .....	726
DeleteFolderIcon Example .....	727
DeleteProgramFolder .....	728
DeleteProgramFolder Example .....	729
DeleteShortcut .....	730
DeleteShortcut Example .....	731
DeleteShortcutFolder .....	732
DeleteShortcutFolder Example .....	733
DeleteWCHARArray .....	734
DialogSetFont .....	734
DialogSetInfo .....	735
DialogSetInfo Example .....	740
Dialog Styles .....	741
CHECKBOX Dialog Style .....	741
CHECKBOX95 Dialog Style .....	742
CHECKMARK Dialog Style .....	743
CHECKLINE Dialog Style .....	744
DIFxDriverPackageGetPath .....	744
DIFxDriverPackageInstall .....	745
DIFxDriverPackagePreinstall .....	749
DIFxDriverPackageUninstall .....	751
Disable .....	754
Disable Example .....	758
Do .....	759
Do Example .....	760
DoInstall .....	761

DoInstall Example .....	764
DotNetCoCreateObject .....	765
DotNetUnloadAppDomain .....	767
<b>Built-In Functions (E-G) .....</b>	<b>769</b>
Enable .....	769
Enable Example .....	772
EndCurrentDialog .....	773
EndDialog .....	774
EndDialog Example .....	775
EnterDisk .....	777
EnterDisk Example .....	778
EnterDiskError .....	779
EnterLoginInfo .....	780
EnterPassword .....	782
ExistsDir .....	783
ExistsDir Example .....	783
ExistsDisk .....	784
ExistsDisk Example .....	785
EzBatchAddPath .....	786
EzBatchAddPath Example .....	787
EzBatchAddString .....	789
EzBatchAddString Example .....	791
EzBatchReplace .....	793
EzBatchReplace Example .....	794
EzConfigAddDriver .....	795
EzConfigAddDriver Example .....	796
EzConfigAddString .....	798
EzConfigAddString Example .....	799
EzConfigGetValue .....	801
EzConfigGetValue Example .....	802
EzConfigSetValue .....	803
EzConfigSetValue Example .....	804
EzDefineDialog .....	805
EzDefineDialog Example .....	807
FeatureAddCost .....	809
FeatureAddItem .....	810
FeatureAddItem Example .....	812
FeatureAddUninstallCost .....	813
FeatureCompareSizeRequired .....	814
FeatureCompareSizeRequired Example .....	816
FeatureConfigureFeaturesFromSuite .....	818
FeatureDialog .....	819
FeatureDialog Example .....	822
FeatureError .....	823
FeatureError Example .....	826
FeatureErrorInfo .....	827

FeatureErrorInfo Example .....	828
FeatureFileEnum .....	830
FeatureFileEnum Example .....	832
FeatureFileInfo .....	834
FeatureFileInfo Example .....	840
FeatureFilterLanguage .....	843
FeatureFilterLanguage Example .....	845
FeatureFilterOS .....	846
FeatureFilterOS Example .....	850
FeatureGetCost .....	852
FeatureGetCost Example .....	853
FeatureGetCostEx .....	854
FeatureGetData .....	855
FeatureGetData Example .....	859
FeatureGetItemSize .....	860
FeatureGetItemSize Example .....	861
FeatureGetTotalCost .....	862
FeatureInitialize .....	863
FeatureInitialize Example .....	865
FeatureIsItemSelected .....	866
FeatureIsItemSelected Example .....	867
FeatureListItems .....	868
FeatureListItems Example .....	869
FeatureLoadTarget .....	870
FeatureMoveData .....	871
FeatureMoveData Example .....	872
FeaturePatch .....	876
FeatureReinstall .....	877
FeatureRemoveAll .....	878
FeatureRemoveAllInLogOnly .....	879
FeatureRemoveAllInMedia .....	880
FeatureRemoveAllInMediaAndLog .....	881
FeatureSaveTarget .....	882
FeatureSelectItem .....	882
FeatureSelectItem Example .....	884
FeatureSelectNew .....	885
FeatureSetData .....	886
FeatureSetData Example .....	889
FeatureSetTarget .....	890
FeatureSetTarget Example .....	891
FeatureSetupTypeEnum .....	892
FeatureSetupTypeEnum Example .....	892
FeatureSetupTypeGetData .....	893
FeatureSetupTypeGetData Example .....	895
FeatureSetupTypeSet .....	898
FeatureSetupTypeSet Example .....	899

FeatureSpendCost .....	900
FeatureSpendUninstallCost .....	901
FeatureStandardSetupTypeSet .....	902
FeatureTotalSize .....	904
<a href="#">FeatureTotalSize Example</a> .....	905
FeatureTransferData .....	907
FeatureUpdate .....	908
FeatureValidate .....	909
<a href="#">FeatureValidate Example</a> .....	910
FileCompare .....	912
<a href="#">FileCompare Example</a> .....	914
FileDeleteLine .....	916
<a href="#">FileDeleteLine Example</a> .....	917
FileGrep .....	919
<a href="#">FileGrep Example</a> .....	920
FileInsertLine .....	922
<a href="#">FileInsertLine Example</a> .....	923
FindAllDirs .....	925
<a href="#">FindAllDirs Example</a> .....	926
FindAllFiles .....	927
<a href="#">FindAllFiles Example</a> .....	929
FindFile .....	931
<a href="#">FindFile Example</a> .....	932
FindWindow .....	933
<a href="#">FindWindow Example</a> .....	934
FormatMessage .....	935
<a href="#">FormatMessage Example</a> .....	936
GetAndAddAllFilesCost .....	937
GetAndAddFileCost .....	938
GetCArrayFromISArray .....	939
GetCHARArrayFromISStringArray .....	940
GetCurrentDialogName .....	941
GetCurrentDir .....	942
GetDir .....	943
<a href="#">GetDir Example</a> .....	944
GetDisk .....	945
<a href="#">GetDisk Example</a> .....	946
GetDiskInfo .....	947
<a href="#">GetDiskInfo Example</a> .....	949
GetDiskSpace .....	950
<a href="#">GetDiskSpace Example</a> .....	950
GetDiskSpaceEx .....	952
<a href="#">GetDiskSpaceEx Example</a> .....	953
GetEnvVar .....	954
<a href="#">GetEnvVar Example</a> .....	955
GetExtendedErrInfo .....	956

GetExtents .....	957
<a href="#">GetExtents Example</a> .....	958
GetFileInfo .....	959
<a href="#">GetFileInfo Example</a> .....	962
GetFolderNameList .....	964
<a href="#">GetFolderNameList Example</a> .....	966
GetFont .....	967
<a href="#">GetFont Example</a> .....	968
GetLine .....	971
<a href="#">GetLine Example</a> .....	972
GetMemFree .....	973
GetObject .....	973
GetObjectByIndex .....	974
GetObjectCount .....	975
GetProfInt .....	976
<a href="#">GetProfInt Example</a> .....	977
GetProfSectionKeyCount .....	977
GetProfString .....	978
<a href="#">GetProfString Example</a> .....	980
GetProfStringList .....	981
<a href="#">GetProfStringList Example</a> .....	982
GetShortcutInfo .....	984
<a href="#">GetShortcutInfo Example</a> .....	986
GetStatus .....	988
GetSystemInfo .....	988
<a href="#">GetSystemInfo Example</a> .....	994
GetTempFileNameIS .....	997
GetTrueTypeFontFileInfo .....	999
GetUpdateStatus .....	1000
GetUpdateStatusReboot .....	1001
GetValidDrivesList .....	1001
<a href="#">GetValidDrivesList Example</a> .....	1002
GetWCHARArrayFromISStringArray .....	1004
GetWindowHandle .....	1004
<a href="#">GetWindowHandle Example</a> .....	1005
<b>Built-In Functions (H-P) .....</b>	<b>1007</b>
Handler .....	1007
HandlerEx .....	1007
<a href="#">HandlerEx Example</a> .....	1009
HIBYTE .....	1010
HIWORD .....	1011
<a href="#">HIWORD Example</a> .....	1012
InstallationInfo .....	1012
Is .....	1012
<a href="#">Is Example</a> .....	1020
ISCompareServicePack .....	1021

ISCompareServicePack Example .....	1022
ISDeterminePlatform .....	1023
IsEmpty .....	1023
IsEmpty Example .....	1024
IsObject .....	1025
LaunchApp .....	1025
LaunchApp Example .....	1025
LaunchAppAndWait .....	1026
LaunchAppAndWait Example .....	1026
LaunchAppAndWaitInitStartupInfo .....	1028
LaunchApplication .....	1030
LaunchApplicationInit .....	1036
ListAddItem .....	1039
ListAddItem Example .....	1040
ListAddList .....	1042
ListAddString .....	1043
ListAddString Example .....	1044
ListAppendFromArray .....	1046
ListAppendToArray .....	1046
ListConvertNumToStr .....	1047
ListConvertStrToNum .....	1048
ListCount .....	1049
ListCount Example .....	1050
ListCreate .....	1051
ListCreate Example .....	1052
ListCurrentItem .....	1053
ListCurrentItem Example .....	1054
ListCurrentString .....	1055
ListCurrentString Example .....	1056
ListDeleteAll .....	1057
ListDeleteItem .....	1058
ListDeleteItem Example .....	1059
ListDeleteString .....	1061
ListDeleteString Example .....	1062
ListDestroy .....	1064
ListDestroy Example .....	1064
ListFindItem .....	1065
ListFindItem Example .....	1066
ListFindKeyValueString .....	1068
ListFindString .....	1070
ListFindString Example .....	1070
ListGetFirstItem .....	1072
ListGetFirstItem Example .....	1073
ListGetFirstString .....	1074
ListGetFirstString Example .....	1075
ListGetIndex .....	1076

ListGetNextItem .....	1077
<a href="#">ListGetNextItem Example</a> .....	<a href="#">1078</a>
ListGetNextString .....	1079
<a href="#">ListGetNextString Example</a> .....	<a href="#">1080</a>
ListGetType .....	1082
<a href="#">ListGetType Example</a> .....	<a href="#">1082</a>
ListReadFromFile .....	1083
<a href="#">ListReadFromFile Example</a> .....	<a href="#">1084</a>
ListSetCurrentItem .....	1085
<a href="#">ListSetCurrentItem Example</a> .....	<a href="#">1086</a>
ListSetCurrentString .....	1088
<a href="#">ListSetCurrentString Example</a> .....	<a href="#">1089</a>
ListSetIndex .....	1090
<a href="#">ListSetIndex Example</a> .....	<a href="#">1092</a>
ListValid .....	1094
<a href="#">ListValid Example</a> .....	<a href="#">1095</a>
ListValidType .....	1096
<a href="#">ListValidType Example</a> .....	<a href="#">1097</a>
ListWriteToFile .....	1098
<a href="#">ListWriteToFile Example</a> .....	<a href="#">1099</a>
ListWriteToFileEx .....	1100
LoadStringFromStringTable .....	1101
LOBYTE .....	1102
LogReadCustomNumber .....	1103
<a href="#">LogReadCustomNumber Example</a> .....	<a href="#">1104</a>
LogReadCustomString .....	1105
<a href="#">LogReadCustomString Example</a> .....	<a href="#">1107</a>
LogWriteCustomNumber .....	1108
<a href="#">LogWriteCustomNumber Example</a> .....	<a href="#">1109</a>
LogWriteCustomString .....	1110
<a href="#">LogWriteCustomString Example</a> .....	<a href="#">1111</a>
LongPathFromShortPath .....	1112
<a href="#">LongPathFromShortPath Example</a> .....	<a href="#">1113</a>
LongPathToQuote .....	1114
<a href="#">LongPathToQuote Example</a> .....	<a href="#">1115</a>
LongPathToShortPath .....	1116
<a href="#">LongPathToShortPath Example</a> .....	<a href="#">1117</a>
LOWORD .....	1118
<a href="#">LOWORD Example</a> .....	<a href="#">1119</a>
MaintenanceStart .....	1120
MediaGetData .....	1123
MediaGetDataEx .....	1123
MessageBeep .....	1126
<a href="#">MessageBeep Example</a> .....	<a href="#">1126</a>
MessageBox .....	1127
<a href="#">MessageBox Example</a> .....	<a href="#">1128</a>



MessageBoxEx .....	1129
NumToStr .....	1131
NumToStr Example .....	1132
OpenFile .....	1133
OpenFile Example .....	1134
OpenFileMode .....	1135
OpenFileMode Example .....	1138
ParsePath .....	1139
ParsePath Example .....	1141
ParseUrl .....	1142
ParseUrl Example .....	1143
PathAdd .....	1144
PathAdd Example .....	1145
PathDelete .....	1147
PathDelete Example .....	1148
PathFind .....	1150
PathFind Example .....	1151
PathGet .....	1153
PathGet Example .....	1153
PathMove .....	1155
PathMove Example .....	1156
PathSet .....	1158
PathSet Example .....	1159
PlaceBitmap .....	1160
PlaceBitmap Example .....	1165
PlaceWindow .....	1166
PlaceWindow Example .....	1169
PlayMMedia .....	1170
PlayMMedia Example .....	1172
PostShowComponentDlg .....	1173
PreShowComponentDlg .....	1174
ProgDefGroupType .....	1175
<b>Built-In Functions (Q-R) .....</b>	<b>1177</b>
QueryProgItem .....	1177
QueryProgItem Example .....	1179
QueryShellMgr .....	1181
QueryShellMgr Example .....	1182
ReadArrayProperty .....	1183
ReadBoolProperty .....	1184
ReadBytes .....	1185
ReadBytes Example .....	1186
ReadNumberProperty .....	1187
ReadStringProperty .....	1188
RebootDialog .....	1189
RebootDialog Example .....	1190
RegDBConnectRegistry .....	1191

<a href="#">RegDBConnectRegistry Example</a>	1194
RegDBCopYKeys	1195
RegDBCopYValues	1198
RegDBCreateKeyEx	1200
<a href="#">RegDBCreateKeyEx Example</a>	1202
RegDBDeleteItem	1203
RegDBDeleteKey	1207
<a href="#">RegDBDeleteKey Example</a>	1208
RegDBDeleteValue	1209
<a href="#">RegDBDeleteValue Example</a>	1210
RegDBDisConnectRegistry	1211
<a href="#">RegDBDisConnectRegistry Example</a>	1212
RegDBGetAppInfo	1214
<a href="#">RegDBGetAppInfo Example</a>	1215
RegDBGetDefaultRoot	1217
RegDBGetItem	1218
<a href="#">RegDBGetItem Example</a>	1222
RegDBGetKeyValueEx	1223
<a href="#">RegDBGetKeyValueEx Example</a>	1225
RegDBGetUninstCmdLine	1227
RegDBKeyExist	1228
<a href="#">RegDBKeyExist Example</a>	1229
RegDBQueryKey	1231
<a href="#">RegDBQueryKey Example</a>	1232
RegDBQueryKeyCount	1234
RegDBQueryStringMultiStringCount	1235
RegDBSetAppInfo	1237
<a href="#">RegDBSetAppInfo Example</a>	1239
RegDBSetDefaultRoot	1240
<a href="#">RegDBSetDefaultRoot Example</a>	1241
RegDBSetItem	1243
<a href="#">RegDBSetItem Example</a>	1246
RegDBSetKeyValueEx	1248
<a href="#">RegDBSetKeyValueEx Example</a>	1251
RegDBSetVersion	1252
RegisterFontResource	1253
<a href="#">RegisterFontResource Example</a>	1256
ReleaseDialog	1256
<a href="#">ReleaseDialog Example</a>	1257
RenameFile	1259
<a href="#">RenameFile Example</a>	1261
ReplaceFolderIcon	1263
<a href="#">ReplaceFolderIcon Example</a>	1265
ReplaceProfString	1266
<a href="#">ReplaceProfString Example</a>	1268
ReplaceShortcut	1269

<a href="#">ReplaceShortcut Example</a> .....	1272
Resize .....	1273
RGB .....	1274
<a href="#">RGB Example</a> .....	1274
<b>Built-In Functions (S-T)</b> .....	<b>1277</b>
SdAskDestPath .....	1277
<a href="#">SdAskDestPath Example</a> .....	1278
SdAskDestPath2 .....	1279
<a href="#">SdAskDestPath2 Example</a> .....	1280
SdAskOptions .....	1281
<a href="#">SdAskOptions Example</a> .....	1284
SdAskOptionsList .....	1285
<a href="#">SdAskOptionsList Example</a> .....	1287
SdBitmap .....	1288
<a href="#">SdBitmap Example</a> .....	1290
SdConfirmNewDir .....	1291
<a href="#">SdConfirmNewDir Example</a> .....	1292
SdConfirmRegistration .....	1294
<a href="#">SdConfirmRegistration Example</a> .....	1295
SdCustomerInformation .....	1296
<a href="#">SdCustomerInformation Example</a> .....	1299
SdCustomerInformationEx .....	1300
<a href="#">SdCustomerInformationEx Example</a> .....	1303
SdDiskSpace2 .....	1304
<a href="#">SdDiskSpace2 Example</a> .....	1305
SdDiskSpaceRequirements .....	1306
SdDisplayTopics .....	1307
<a href="#">SdDisplayTopics Example</a> .....	1308
SdExceptions .....	1310
<a href="#">SdExceptions Example</a> .....	1311
SdFeatureDialog .....	1312
<a href="#">SdFeatureDialog Example</a> .....	1315
SdFeatureDialog2 .....	1316
<a href="#">SdFeatureDialog2 Example</a> .....	1318
SdFeatureDialogAdv .....	1319
<a href="#">SdFeatureDialogAdv Example</a> .....	1321
SdFeatureMult .....	1322
<a href="#">SdFeatureMult Example</a> .....	1324
SdFeatureTree .....	1325
<a href="#">SdFeatureTree Example</a> .....	1327
SdFilesInUse .....	1328
<a href="#">SdFilesInUse Example</a> .....	1330
SdFinish .....	1331
<a href="#">SdFinish Example</a> .....	1332
SdFinishEx .....	1334
<a href="#">SdFinishEx Example</a> .....	1335

SdFinishReboot.....	1335
<a href="#">SdFinishReboot Example .....</a>	<a href="#">1337</a>
SdFinishUpdate.....	1338
SdFinishUpdateEx.....	1338
SdFinishUpdateReboot.....	1340
<a href="#">SdFinishUpdateReboot Example.....</a>	<a href="#">1342</a>
SdGeneralInit .....	1342
<a href="#">SdGeneralInit Example .....</a>	<a href="#">1343</a>
SdInit .....	1345
<a href="#">SdInit Example .....</a>	<a href="#">1345</a>
SdLicense.....	1346
<a href="#">SdLicense Example .....</a>	<a href="#">1347</a>
SdLicense2.....	1348
<a href="#">SdLicense2 Example .....</a>	<a href="#">1351</a>
SdLicense2Ex.....	1351
SdLicense2Rtf.....	1354
<a href="#">SdLicense2Rtf Example.....</a>	<a href="#">1356</a>
SdLicenseEx.....	1356
SdLicenseRtf .....	1359
<a href="#">SdLicenseRtf Example.....</a>	<a href="#">1360</a>
SdLoadString.....	1361
<a href="#">SdLoadString Example .....</a>	<a href="#">1362</a>
SdLogonUserBrowse.....	1363
SdLogonUserCreateUser.....	1363
SdLogonUserInformation.....	1363
SdLogonUserListGroup.....	1364
SdLogonUserListServers.....	1365
SdLogonUserListUsers.....	1365
SdMakeName.....	1365
<a href="#">SdMakeName Example.....</a>	<a href="#">1366</a>
SdOptionsButtons .....	1370
<a href="#">SdOptionsButtons Example .....</a>	<a href="#">1373</a>
SdOutOfDiskSpace.....	1375
SdPatchWelcome.....	1376
<a href="#">SdPatchWelcome Example .....</a>	<a href="#">1377</a>
SdProductName.....	1378
<a href="#">SdProductName Example.....</a>	<a href="#">1379</a>
SdRegisterUser.....	1379
<a href="#">SdRegisterUser Example.....</a>	<a href="#">1382</a>
SdRegisterUserEx.....	1383
<a href="#">SdRegisterUserEx Example.....</a>	<a href="#">1385</a>
SdRMFilesInUse .....	1386
SdSelectFolder .....	1388
<a href="#">SdSelectFolder Example .....</a>	<a href="#">1389</a>
SdSetupCompleteError.....	1390
<a href="#">SdSetupCompleteError Example.....</a>	<a href="#">1391</a>

SdSetupType	1392
<a href="#">SdSetupType Example</a>	1393
SdSetupType2	1394
<a href="#">SdSetupType2 Example</a>	1396
SdSetupTypeEx	1397
<a href="#">SdSetupTypeEx Example</a>	1398
SdShowAnyDialog	1399
<a href="#">SdShowAnyDialog Example</a>	1400
SdShowDlgEdit1	1401
<a href="#">SdShowDlgEdit1 Example</a>	1402
SdShowDlgEdit2	1403
<a href="#">SdShowDlgEdit2 Example</a>	1404
SdShowDlgEdit3	1405
<a href="#">SdShowDlgEdit3 Example</a>	1408
SdShowFileMods	1409
<a href="#">SdShowFileMods Example</a>	1410
SdShowInfoList	1412
<a href="#">SdShowInfoList Example</a>	1413
SdShowMsg	1414
<a href="#">SdShowMsg Example</a>	1416
SdStartCopy	1416
<a href="#">SdStartCopy Example</a>	1418
SdStartCopy2	1419
<a href="#">SdStartCopy2 Example</a>	1420
SdSubstituteProductInfo	1422
SdWelcome	1422
<a href="#">SdWelcome Example</a>	1423
SdWelcomeMaint	1424
<a href="#">SdWelcomeMaint Example</a>	1425
SeekBytes	1426
<a href="#">SeekBytes Example</a>	1427
SelectDir	1429
<a href="#">SelectDir Example</a>	1431
SelectDirEx	1432
<a href="#">SelectDirEx Example</a>	1435
SelectFolder	1436
<a href="#">SelectFolder Example</a>	1437
SendMessage	1438
<a href="#">SendMessage Example</a>	1439
ServiceAddService	1441
ServiceExistsService	1443
ServiceGetServiceState	1443
ServiceInitParams	1444
ServiceRemoveService	1445
ServiceStartService	1446
ServiceStopService	1447

SetColor .....	1448
<a href="#">SetColor Example</a> .....	1451
SetDialogTitle .....	1452
<a href="#">SetDialogTitle Example</a> .....	1453
SetDisplayEffect .....	1454
<a href="#">SetDisplayEffect Example</a> .....	1455
SetErrorMsg .....	1457
<a href="#">SetErrorMsg Example</a> .....	1458
SetErrorTitle .....	1459
<a href="#">SetErrorTitle Example</a> .....	1460
SetExtendedErrInfo .....	1461
SetFileInfo .....	1462
<a href="#">SetFileInfo Example</a> .....	1464
SetFont .....	1465
<a href="#">SetFont Example</a> .....	1466
SetInstallationInfo .....	1467
SetObjectPermissions .....	1468
<a href="#">SetObjectPermissions Example</a> .....	1474
SetShortcutProperty .....	1475
<a href="#">SetShortcutProperty Example</a> .....	1478
SetStatus .....	1479
SetStatusEx .....	1480
SetStatusExStaticText .....	1481
SetStatusWindow .....	1482
<a href="#">SetStatusWindow Example</a> .....	1483
SetTitle .....	1484
<a href="#">SetTitle Example</a> .....	1488
SetUpdateStatus .....	1488
SetUpdateStatusReboot .....	1489
SetupType .....	1489
<a href="#">SetupType Example</a> .....	1491
SetupType2 .....	1493
<a href="#">SetupType2 Example</a> .....	1495
ShowObjWizardPages .....	1497
ShowProgramFolder .....	1497
<a href="#">ShowProgramFolder Example</a> .....	1498
ShowWindow .....	1499
SilentReadData .....	1502
<a href="#">SilentReadData Example</a> .....	1504
SilentWriteData .....	1508
<a href="#">SilentWriteData Example</a> .....	1510
SizeOf .....	1514
SizeWindow .....	1514
<a href="#">SizeWindow Example</a> .....	1516
Sprintf .....	1516
<a href="#">Sprintf Example</a> .....	1517

SprintfBox .....	1518
<a href="#">SprintfBox Example</a> .....	<a href="#">1521</a>
SprintfMsiLog .....	1522
SQLBrowse .....	1523
SQLBrowse2 .....	1524
SQLDatabaseBrowse .....	1525
SQLRTComponentInstall .....	1526
SQLRTComponentUninstall .....	1527
SQLRTConnect .....	1528
SQLRTConnect2 .....	1529
SQLRTConnectDB .....	1531
SQLRTDoRollbackAll .....	1532
SQLRTGetBatchList .....	1533
SQLRTGetBatchMode .....	1534
SQLRTGetBrowseOption .....	1535
SQLRTGetComponentScriptError .....	1537
SQLRTGetComponentScriptError2 .....	1538
SQLRTGetConnectionAuthentication .....	1540
SQLRTGetConnectionInfo .....	1541
SQLRTGetConnections .....	1542
SQLRTGetDatabases .....	1543
SQLRTGetErrorMessage .....	1544
SQLRTGetLastError .....	1545
SQLRTGetLastError2 .....	1546
SQLRTGetScriptErrorMessage .....	1546
SQLRTGetServers .....	1547
SQLRTGetServers2 .....	1548
SQLRTInitialize .....	1549
SQLRTInitialize2 .....	1550
SQLRTPutConnectionAuthentication .....	1551
SQLRTPutConnectionInfo .....	1551
SQLRTPutConnectionInfo2 .....	1552
SQLRTServerValidate .....	1553
SQLRTSetBrowseOption .....	1555
SQLRTTestConnection .....	1556
SQLRTTestConnection2 .....	1558
SQLServerLogin .....	1560
SQLServerSelect .....	1561
SQLServerSelectLogin .....	1562
SQLServerSelectLogin2 .....	1563
SQLServerSelectLoginEx .....	1566
StatusUpdate .....	1567
<a href="#">StatusUpdate Example</a> .....	<a href="#">1569</a>
StrAddLastSlash .....	1570
StrCompare .....	1571
<a href="#">StrCompare Example</a> .....	<a href="#">1572</a>

StrConvertSizeUnit .....	1573
StreamFileFromBinary .....	1574
StrFind .....	1575
StrFind Example .....	1575
StrFindEx .....	1576
StrGetTokens .....	1577
StrGetTokens Example .....	1578
StrLength .....	1580
StrLength Example .....	1580
StrLengthChars .....	1581
StrLengthChars Example .....	1582
StrPutTokens .....	1583
StrRemoveLastSlash .....	1584
StrRemoveLastSlash Example .....	1585
StrReplace .....	1586
StrSub .....	1587
StrSub Example .....	1588
STRTOCHAR .....	1589
StrToLower .....	1590
StrToLower Example .....	1591
StrToNum .....	1592
StrToNum Example .....	1593
StrToNumHex .....	1594
StrToUpper .....	1595
StrToUpper Example .....	1596
StrTrim .....	1597
SuiteFormatString .....	1598
SuiteFormatString Example .....	1599
SuiteGetProperty .....	1600
SuiteGetProperty Example .....	1601
SuiteLogInfo .....	1602
SuiteLogInfo Example .....	1603
SuiteReportError .....	1604
SuiteResolveString .....	1605
SuiteResolveString Example .....	1606
SuiteSetProperty .....	1607
SuiteSetProperty Example .....	1608
System .....	1609
System Example .....	1610
TextSubGetValue .....	1610
TextSubGetValue Example .....	1611
TextSubParseTextSub .....	1612
TextSubParseTextSub Example .....	1613
TextSubSetValue .....	1614
TextSubSetValue Example .....	1615
TextSubSubstitute .....	1616



<a href="#">TextSubSubstitute Example</a> .....	1617
<b>Built-In Functions (U-Z)</b> .....	<b>1619</b>
UninstallApplication .....	1619
UnUseDLL .....	1621
<a href="#">UnUseDLL Example</a> .....	1621
UpdateServiceCheckForUpdates .....	1623
UpdateServiceCreateShortcut .....	1623
UpdateServiceEnableUpdateManagerInstall .....	1624
UpdateServiceGetAgentTarget .....	1624
UpdateServiceOnEnabledStateChange .....	1624
UpdateServiceRegisterProduct .....	1624
UpdateServiceRegisterProductEx .....	1625
UpdateServiceSetHost .....	1625
UpdateServiceSetLanguage .....	1625
UseDLL .....	1625
<a href="#">UseDLL Example</a> .....	1628
VarInit .....	1629
VarRestore .....	1630
<a href="#">VarRestore Example</a> .....	1633
VarSave .....	1634
<a href="#">VarSave Example</a> .....	1636
<a href="#">VarSave Stack Example</a> .....	1637
VerCompare .....	1638
<a href="#">VerCompare Example</a> .....	1639
VerFindFileVersion .....	1641
<a href="#">VerFindFileVersion Example</a> .....	1642
VerGetFileLanguages .....	1644
<a href="#">VerGetFileLanguages Example</a> .....	1645
VerGetFileVersion .....	1646
<a href="#">VerGetFileVersion Example</a> .....	1647
VerProductCompareVersions .....	1648
VerProductGetInstalledVersion .....	1649
VerProductIsVersionSupported .....	1650
VerProductNumToStr .....	1651
VerProductStrToNum .....	1652
VerProductVerFromVerParts .....	1653
VerProductVerPartsFromVer .....	1654
VerSearchAndUpdateFile .....	1655
<a href="#">VerSearchAndUpdateFile Example</a> .....	1658
VerUpdateFile .....	1659
<a href="#">VerUpdateFile Example</a> .....	1663
WaitForApplication .....	1664
WaitOnDialog .....	1667
<a href="#">WaitOnDialog Example</a> .....	1668
Welcome .....	1670
<a href="#">Welcome Example</a> .....	1671

WizardDirection .....	1672
WriteArrayProperty .....	1673
WriteBoolProperty .....	1674
WriteBytes .....	1675
WriteBytes Example. ....	1676
WriteLine .....	1677
WriteLine Example. ....	1678
WriteNumberProperty .....	1680
WriteProfInt .....	1680
WriteProfInt Example .....	1682
WriteProfString .....	1683
WriteProfString Example .....	1684
WriteStringProperty .....	1685
XCopyFile .....	1686
XCopyFile Example .....	1691
<b>Index .....</b>	<b>1693</b>

# InstallScript Language Reference

InstallShield makes designing your installation easy with InstallScript, a simple but powerful programming language. InstallScript is similar to the C language. It has a defined format and regulated syntax. It uses certain data types, each with specific properties. It also allows you to create custom functions.

InstallScript, however, does not provide the full range of programming functionality that C does. InstallScript was designed to do one thing—create installations. And it does so better than any programming language in the world. Regardless of your programming background, you can quickly learn to build an installation with InstallScript.



**Project** • Some InstallScript functions, events, and variables are limited to specific project types.

**Table 1** • InstallScript Language Reference

Section	Description
<b>Integrated Compiler</b>	Provides general information about the InstallScript integrated compiler.
<b>Command-Line Compiler</b>	Contains details about the InstallScript command-line compiler that can be invoked from the DOS prompt or from a DOS batch file.
<b>Setup Scripts</b>	Introduces you to the InstallScript language and the structure of a script.
<b>Language Keywords</b>	Presents background about language keywords, which are words InstallScript uses as commands in the script. Language keywords are interpreted by the InstallScript compiler to perform some action, or are considered part of a statement.
<b>Predefined Constants</b>	Identifies and describes each of the predefined constants reserved by InstallScript. These constants represent specific literal values that are passed to and returned by built-in functions.

**Table 1 •** InstallScript Language Reference (cont.)

Section	Description
<b>Predefined Script Variables</b>	Contains information about script variables that you can use with InstallScript.
<b>Data Types and Predefined Structures</b>	Contains content about the data types and predefined structures supported in InstallScript.
<b>Preprocessor Directives</b>	Discusses preprocessor directives, which are instructions to the InstallScript compiler that are executed as the script is compiled. Preprocessor directives can instruct the compiler to include other source files in the compilation, to define constants, to include or exclude statements based on compile-time conditions, and to display a user-defined error message.
<b>Flow Control</b>	Contains information on how to control the flow of execution within scripts.
<b>Event Handlers</b>	InstallScript project installation programs are driven by the InstallScript engine, which generates a series of events in a specific order.
<b>Functions</b>	Describes the different types of functions that you can use in your installation scripts; also includes details about—and examples of—each of the built-in functions available in InstallScript.
<b>Operators</b>	Contains information about supported operators in InstallScript.
<b>Objects and Object Handlers</b>	Contains information about the objects that InstallScript supports and how to separate error handling from the rest of your script code.



**Note •** Some of the functions that were available in InstallShield Professional are deprecated in later versions of InstallShield. To view a list of the functions, see *Unsupported Functions*.

Because the InstallShield Help Library is designed to interact with InstallShield, it is recommended that you open the help from within InstallShield. Copying the help files to another folder or system causes many of its features to work incorrectly.

## Integrated Compiler

If you want to compile your script without building a release, you can use the integrated InstallScript compiler in InstallShield.

**Task****To compile your script:**

On the **Build** menu, click **Compile**.

InstallShield displays compiler messages in the Output window.

## Command-Line Compiler

In addition to the integrated compiler, which you can launch from within InstallShield, InstallShield includes a command-line compiler that you can invoke from the DOS prompt or from a DOS batch file. This program, called **Compile.exe**, is in the following folder:

*InstallShield Program Files Folder/System*

After you have completed the design of an installation project, you can use **Compile.exe** to compile the installation script using different options than those that are used when you compile the script from within InstallShield.

For the syntax and available command-line parameters and switches for **Compile.exe**, see [Compile.exe](#).



**Note** • When you build a release from the command line using **ISCmdBld.exe**, the build engine automatically compiles your script; therefore, you do not need to use **Compile.exe** directly unless you want to use compiler options that are different from those specified for the project in InstallShield. For more information, see [ISCmdBld.exe](#).



# Setup Scripts

A setup script is a collection of event handlers, functions called by those event handlers, and data used by the event handlers and functions. These elements are expressed in the InstallScript Language, a simple but powerful programming language. InstallScript is similar to the C language. It has a defined format and regulated syntax. It uses certain data types, each with specific properties. It also allows you to create custom functions.

InstallScript, however, does not provide the full range of programming functionality that C does. InstallScript was designed to do one thing—create setups. And it does this effectively and efficiently.

Regardless of your programming background, you can learn quickly to build your setup with InstallScript.

## InstallScript Limits

Following are limits for the compiled script file (**setup.inx**):

- Maximum number of statements: about 4,294,967,295 (If this limit is exceeded, error -5009 may occur during setup initialization.)
- Maximum number of global variables: about 196,605 (65,535 numbers, 65,535 variants, 65,535 strings)
- Maximum number of typedefs: about 65,535
- Maximum number of prototypes: about 65,535
- Maximum number of functions: about 65,535
- Maximum number of statements per function: about 65,535
- Maximum local variables per function: about 196,605 (65,535 numbers, 65,535 variants, 65,535 strings)

Following are limits for script files (.rul):

- Maximum line width: 1,024 characters
- Maximum number of nested include files: 80
- Maximum number of include files: 2,048
- Maximum identifier length: 63 characters
- Maximum number of macro expansions: 100
- Maximum macro expansion text length: 256 characters
- Maximum file name length: 256 characters
- Maximum number of nested #if statements: 10
- Maximum number of parameters per function: 16

Compile errors occur if one or more of these .rul limits are exceeded.



**Tip** • If you encounter any of the aforementioned limits, consider reducing the size of your installation script by removing code, or by splitting up your installation script into multiple projects, creating separate installations, and calling the child installations from your main installation (parent installation).

## Structure of a Script

Every script includes declarations and function blocks. Declarations can precede function declarations or appear between a function statement and the begin statement for that function.

The general outline of a script is shown below:

```
// Constant definitions, global data declarations, and function declarations

// Function blocks
```

## Declarations

Every script begins with global data declarations. Here, you define constants and declare each of the global variables and user-defined functions that you will be using. Declarations instruct the InstallScript compiler that the script will be using the listed items at a later time. Declarations also build an association between a function and its attributes or values. You do not need to declare any of the built-in functions, since the InstallScript compiler already recognizes the function names.

Below are some examples of constant definitions, data declarations, and function declarations:

```
// Constant definitions
#define PRODUCT "InstallShield"
#define LIMIT    100

// Variable declarations
CHAR  cVal;
NUMBER nVal;
STRING szName;

// Function declarations
prototype DisplayMsg (NUMBER, STRING);
prototype GetName (BYREF STRING);
```

## Program Block

Program blocks are used in scripts written using InstallShield Professional 5.5 or earlier. A program block cannot be used for an InstallScript custom action or in an event-driven script. The only code that is executed is found in event handlers and entry-point functions.

```
program
```

```
// In an event-driven script, the program block is optional and remains empty.
```

```
endprogram
```

## Function Block




---

**Project** • This information applies to InstallScript projects.



All functions that have been declared with a prototype statement must be defined in the function block, which follows the keyword `endprogram` in a setup script. Additional global data declarations may be made in the function block, either between the `endprogram` statement and the first function declaration or between function declarations. However, data declared in the function block is visible only to functions that are defined after the data declaration.

## Identifiers

Identifiers are the names that you create to denote constants, variables, and functions in your script. Observe the following syntax rules when creating identifiers:

- An identifier may be of any length, but only the first 63 characters are significant.
- The first character of an identifier must be alphabetic (a-z, A-Z) or an underscore.
- The remaining characters may be alphabetic (a-z, A-Z), numeric (0-9), or an underscore.
- Each identifier must be unique. Be careful not to create an identifier that is a reserved word in InstallScript.

## Syntax Punctuation Rules

Like any programming language, InstallScript has syntax rules that regulate its usage. The basic syntax of InstallScript is similar to that of the C programming language.

The following punctuation reminders apply to all sections of the script:

- Most statements end with a semicolon (;). This includes many one-word statements, such as `end;`, `exit;`, and `return;`.
- Preprocessor statements—such as `#define` and `#include`—never end with a semicolon.
- The keywords `program`, `endprogram`, and `begin` are placed on separate lines by themselves and receive no punctuation. The function line that begins each function block receives no punctuation.
- End a label, such as `start:` or `startthere;`, with a colon (:).
- Enclose parameter lists within parentheses. Separate multiple parameters with commas.

## Writing Comments

InstallScript gives you two ways to create comments in a script. You can use either method to add explanatory text to your script or to exclude or “comment out” certain parts of your script for testing and debugging purposes.




---

**Caution** • You can begin comments anywhere in a script—with one exception: Comments cannot be placed on the same line as an `#ifdef` or `#ifndef` statement. You must write comments before or after these statements, if necessary. Otherwise, the compiler returns an error.

### Block of Text

One way to create a comment is to enclose a block of text between the character pairs `/*` and `*/`. This method makes it easy to write a comment over multiple lines:

```
/* This is a line of sample code that shows you
 * how to use the InstallScript function PlaceBitmap. */
```

## Line by Line

The second way is to insert the characters `//` into a line. The compiler ignores everything to the right of the double slashes on that line only.

```
// This is a line of sample code showing the
// InstallScript function PlaceBitmap.
```

# Using White Space

Like C and other programming languages, InstallScript does not recognize white space (spaces and tabs, carriage returns) except in a string literal. It is recommended that you use white space to make your script easier to follow.

## Code Without White Space

For example, the following section of code is dense and difficult to decipher:

```
#define DISK_DRIVE "C:\\\"
    STRING szDrive, svString;
    NUMBER nSpace, nResult;
szDrive = DISK_DRIVE;
nSpace = GetDiskSpace(szDrive);
nResult = NumToStr(svString, nSpace);
if (nResult < 0) then
    MessageBox("NumToStr failed.", SEVERE);
    abort;
endif;
sprintfBox(INFORMATION, "Info", "Disk Space: %s", svString);
```

## Code With White Space

Adding white space with indentation makes the same code much easier to read:

```
#define DISK_DRIVE "C:\\\"

    STRING szDrive, svString;
    NUMBER nSpace, nResult;

    szDrive = DISK_DRIVE;
    nSpace = GetDiskSpace(szDrive);

    nResult = NumToStr(svString, nSpace);
    if (nResult < 0) then
        MessageBox("NumToStr failed.", SEVERE);
        abort;
    endif;

    sprintfBox(INFORMATION, "Info",
        "Disk Space: %s", svString);
```

# Hungarian Notation

InstallShield help topics employ an extended form of Hungarian notation—a naming convention that uses short, lowercase prefixes to indicate the data type. For example, `iPointSize` denotes an integer variable, while `szFileName` indicates a string variable.

Hungarian notation is used in example scripts to indicate the data type of all variables. In function syntax descriptions, Hungarian notation is used for parameter names to indicate the type of data that may be passed in a parameter. For example, the syntax description of `BatchDeleteEx` shows that it takes two parameters:

```
BatchDeleteEx ( szKey, nOptions );
```

The first parameter, identified as `szKey`, could be a string variable or constant. The second, identified as `nOptions`, could be a number variable or constant.

## Variable Parameters

In those cases where a variable parameter is required, the Language reference employs a special set of two-letter prefixes:

- The first letter indicates the data type.
- The second character is the letter *v*, for variable.

In the syntax description for `GetDir`, the first parameter can be a string variable or constant, but the second parameter must be a variable.

```
GetDir ( szPath, svDir );
```

Functions that require variable parameters generally return data to the caller in those parameters.

## Prefix Table

Because Hungarian notation makes it easy to recognize a variable's type, it is strongly recommended that you use Hungarian notation when you create variable names in your own scripts. The table below describes each of the prefixes used in InstallShield.

**Table 1 • Prefix Table**

Prefix	Data Type	When Used in Function Syntax
<b>b</b>	Boolean (BOOL)	Boolean constant, literal, or variable.
<b>bv</b>	Boolean (BOOL)	Boolean variable only. Constants and literals not allowed.
<b>c</b>	Character (CHAR)	Character constant, literal, or variable.
<b>const</b>	Constant	Constant or literal. Variables not allowed.
<b>h</b>	Handle (HWND)	Handle variable.
<b>i</b>	Integer (INT)	Integer constant, literal, or variable.
<b>l</b>	Long integer (LONG)	Long integer constant, literal, or variable.

**Table 1 •** Prefix Table (cont.)

Prefix	Data Type	When Used in Function Syntax
<b>lv</b>	Long integer (LONG)	Long integer variable only. Constants and literals not allowed.
<b>list</b>	List (LIST)	List variable.
<b>n</b>	Number (NUMBER)	Number constant, literal, or variable.
<b>nv</b>	Number (NUMBER)	Number variable only. Constants and literals not allowed.
<b>p</b>	Pointer (POINTER)	Pointer variable.
<b>pstruct</b>	Pointer to a defined structure type	Not used.
<b>s</b>	Short integer (SHORT)	Short integer constant, literal, or variable.
<b>sz</b>	String (STRING)	String constant, literal, or variable.
<b>sv</b>	String (STRING)	String variable only. Constants and literals not allowed.
<b>struct</b>	Defined structure type	Not used.

## Escape Sequences

An escape sequence is a set of characters used to insert into a string certain special characters—such as tabs, carriage returns and quotation marks. Escape sequences in InstallScript are very much like those in C. They begin with a backslash, called an escape character, and the backslash is followed by one or more characters that have special meaning. If the backslash is followed by characters other than those used in an escape sequence, the backslash is ignored.

### Inserting a Newline Character Into a String

A commonly used escape sequence is `\n`, which inserts a newline character into a string. The string “This is line one, This is line two.” is displayed or printed on a single line. However, the string “This is line one,\nThis is line two.” is displayed or printed as shown below:

```
This is line one,
This is line two.
```



**Note •** The `\n` escape sequence works only in multiline static text fields. For example, you can insert `\n` in the `szQuestion` argument of [AskText](#) to manually format the string. You can also use `\n` with [MessageBox](#) and [SprintfBox](#).

The newline escape sequence is case sensitive; that is, `\N` does not insert a newline character.

The percent sign (%) also has a special function in InstallScript; it is used as the first character of a format specifier, which is a sequence of characters that is used with functions such as *Sprintf* and *SprintBox* to indicate how the value stored in a variable should be displayed on screen.

## Supported Escape Sequences

The following table lists the escape sequences that are supported by InstallScript:

**Table 2 • Supported Escape Sequences**

Escape Sequence	Performs the Following Action
<code>\n</code>	Inserts a line feed.
<code>\'</code>	Inserts a single quotation mark in the string.
<code>\"</code>	Inserts a double quotation mark in the string.
<code>\r</code>	Inserts a carriage return only. Does not insert a line feed.
<code>\t</code>	Inserts a tab character.
<code>\ooo</code>	Indicates an ASCII character—not an integer—in octal notation.
<code>\\</code>	Inserts a backslash.

## Specifying a Universal Naming Convention Path

To specify a Universal Naming Convention (UNC) path in an InstallScript string, you must use *two* backslash escape sequences (that is, four backslashes—`\\\\`), to create the double backslash at the start of the path. For example, the path `\\MyServer\\Public\\Readme.txt` must be specified as follows:

```
"\\\\MyServer\\Public\\Readme.txt"
```

## Embedding Quotation Marks

You can insert double quotation marks as part of a string literal using one of two methods. If you begin the string literal with double quotation marks, you must use the `\` escape character to embed double quotation marks. You can, however, begin the literal with a single quotation mark and then type the double quotation mark:

```
//These two statements will both yield embedded double quotation marks
szQuote1 = "Who said, \"Quitters never win\"?";
szQuote2 = 'The same guy who said, "I quit. "';
```

To embed a single quotation mark, either use the `\` escape sequence or open the string literal with double quotation marks:

```
//These two statements will both yield embedded single quotation marks
szQuote1 = 'Who said, \'Nice guys finish last\'?';
szQuote2 = "The same guy who said, 'I win.'";
```



**Note** • Your setup scripts must use the standard quotation marks ( " and ' ) found to the right of the semicolon (;) key on the standard U.S. keyboard. Do not use open or closed typographer's quotation marks (""), such as this help files uses outside of example scripts.

## Format Specifiers

Format specifiers are used with the functions [Sprintf](#) and [SprintfBox](#) to control the display of values that are stored in variables. A format specifier begins with a percent sign (%) and is followed by at least one or two characters. Format specifications follow the format shown below:

% [-] [#] [0] [width] [.precision] type

Each field of a format specification is a single character or number that represents a particular format option. The type field, for example, determines whether [Sprintf](#) or [SprintfBox](#) interprets the associated argument as a character, a string, or a number. The initial character % and the type field are both required. Items enclosed within brackets are optional. The simplest format specification contains only the percent sign and a type character, for example %s.

In the following example, the value of svString is displayed in a message box. The format specifier %s, which is assigned to svFormat, indicates to [SprintfBox](#) that the value of svString should be displayed as a string of characters.

```
STRING szTitle, szFormat, szString;

szTitle = "Demonstrate format specifiers";
szFormat = "%s";
szString = "This is a string.";

SprintfBox(INFORMATION, szTitle, szFormat, szString);
```

The value assigned to svFormat may contain literal characters (including escape sequences) that are to be displayed along with the value of a variable. In the following example, an identifying label is displayed to the left of a number variable: nNumber = 100;.

```
STRING szTitle, szFormat;
NUMBER nNumber;

szTitle = "Demonstrate format specifiers";
szFormat = "nNumber = %d.";
nNumber = 100;

SprintfBox(INFORMATION, szTitle, szFormat, nNumber);
```



**Note** • To print a percent sign, you must insert two percent signs in the string assigned to svFormat. Assuming that the number to be printed is 100, the following format specification string displays "nNumber = 100%":

```
svFormat = "nNumber = %d%%."
```

**Table 3 • Format Specifier Fields**

Field	Meaning
-	If you include a hyphen after the percent character, the output value is aligned left and padded on the right to the width of the field with blanks or zeros. If you omit this field, the output value is right aligned and padded on the left.
#	Use this symbol to prefix hexadecimal values with 0x (lowercase) or 0X (uppercase).
0	Pads the output value with zeros to fill the field width. If you omit this field, the output value is padded with blanks.
<b>width</b>	Enter the minimum number of characters you want to place in this field. Type the width field as a non-negative integer. When you enter a width specification, the value is never truncated. If the number of characters in the output value is greater than the width specified, or if you omit the width field, every character of the value is displayed, subject to the value of the precision field.
<b>precision</b>	Enter the minimum number of digits you want in this field. If the number of digits in the argument is less than the precision value you enter, the output value on the left is padded with zeros. When the number of digits exceeds the precision value, the value is not truncated. If you enter a precision value of zero or omit it entirely, or if the period (.) appears without a number following it, the precision is set to 1. For strings, convert the maximum number of characters.
<b>type</b>	<p>Format the corresponding argument as a character, a string, or a number. When two format specifier letter combinations are shown, you can use one or the other, but not both at the same time. This is a required field. In this field you must enter one of the following characters:</p> <ul style="list-style-type: none"> <li>● <b>c</b>—Formats a single character of type CHAR. The Sprintf function ignores a character with a numeric value of zero.</li> <li>● <b>d, i</b>—Formats a single integer of type INT or of type NUMBER.</li> <li>● <b>ld, li</b>—Formats a single signed decimal integer of type LONG.</li> <li>● <b>lx, lX</b>—Formats a single unsigned hexadecimal integer of type LONG.</li> <li>● <b>s</b>—Formats a string (type STRING).</li> </ul>

Each format specifier has a matching variable. The variables are listed from left to right after the string, with the first variable matching the first format specifier in the string, the second variable matching the second format specifier in the string, and so on. At run time, InstallShield inserts each variable's contents into the string at the location of its matching format specifier.

# Reserved Words

Reserved words and characters have special meaning in InstallScript and cannot be used except for their intended purposes. InstallScript has the following classes of reserved words:

- [Functions](#)
- [Language Keywords](#)
- [Predefined Constants](#)
- [System Variables](#)
- [Event Handlers](#)
- [Predefined Script Variables](#)



# Language Keywords

Language keywords are words InstallScript uses as commands in the script. Language keywords are interpreted by the InstallScript compiler to perform some action, or are considered part of a statement. You cannot use these following keywords for any reason other than their predefined purpose (for example, these keywords cannot be used as variable names).

## abort

When the script encounters an abort statement, the setup terminates. The abort statement is also encountered in the InstallShield default exit handler (OnCanceling) when the end user exits the installation before it has completed by pressing the Esc key or the Cancel button of an InstallScript dialog.



**Note** • The abort statement exits the installation and runs the uninstaller in silent mode to clean up the aborted installation. The exit statement aborts the installation, but does not remove anything from the target system.

The abort statement does not call a rollback if initiated after the OnFirstUIAfter event.

## BOOL

Boolean data: either TRUE (1) or FALSE (0). Variables of this type should not be used to store any other values. Like C++, InstallScript evaluates non-zero values as TRUE; only the value of zero is evaluated as FALSE. Normally, the value of 1 is used to indicate TRUE.

## cdecl



**Project** • This information applies to InstallScript projects.

The cdecl keyword is used when declaring an external DLL function that uses the cdecl calling convention. For example:

```
prototype cdecl POINTER Msvcrt.memcpy( byref string, pointer, long );
```

In previous versions of InstallShield Professional, the setup engine always used the [stdcall](#) convention but would sometimes overlook an inconsistent DLL convention.

Most Windows API functions use the stdcall (WINAPI) calling convention. Consult Microsoft documentation for more information about calling conventions.

## exit

When the setup program encounters an exit statement in the script it is executing, the setup process terminates. Each setup script contains—at most—one exit statement. If your script includes conditional expressions that might cause it to exit before the installation has completed, you should use [abort](#) instead of exit.

## export

The prototype of any function that is called directly by the setup engine must be marked as export. An example is shown below:

```
export prototype NewFeature1_Installing();
```

## external

The keyword external is reserved and may not be used.

## for...endfor

The for statement is designed to execute one or more statements a fixed number of times. It begins with the keyword for and an expression that specifies the number of times statements within the for structure are to be executed. The for structure ends with the keyword endfor.



**Note** • The for statement itself is not terminated with a semicolon; however, a semicolon is required after the endfor statement.

### Using for...endfor

In the following example, the function MessageBox is called 10 times. On the first pass, iCount is set to 1. Because 1 is in the specified range (1 to 10), the message box is displayed. Then iCount is incremented by 1 and the for statement is resolved again. This time, iCount = 2 (still in the specified range) and the message box is displayed a second time.

When iCount is incremented after the tenth pass, its value becomes 11. Because this value is outside the specified range, the for statement ends.

```
for iCount = 1 to 10
    MessageBox ("This appears ten times.", INFORMATION);
endfor;
```

### Adjusting the Increment

The default increment in a for statement is one (1), but you can use the keyword step to adjust the increment. In the example below, step increases the value of iCount by 10 each time loop is executed. On the first pass, iCount = 10; on the second pass, iCount = 20; on the third pass, iCount = 30, and so on.

```
for iCount = 10 to 100 step 10
    MessageBox ("This appears ten times.", INFORMATION);
endfor;
```

#### Counting Down from a Higher Number to a Lower Number

You can count down from a higher number to a lower number by using the keyword downto in place of the keyword to. In the following example, a message box is displayed three times. The first time the loop is entered, j is set to 20.

Because downto specifies that the controlling variable be decremented and step 5 sets a decrement of 5 per loop, j is equal to 15 the second time the loop is entered. The third time, j is equal to 10.

```

for j = 20 downto 10 step 5
    MsgBox ("This appears three times.", INFORMATION);
endfor;

```



**Note** • You cannot define a label within a for statement.

## goto

The goto keyword is used to branch directly to the statement immediately following a specified label. In the following code fragment, the goto statement causes execution to continue with the AskText statement.

```

Name:
AskText("Company name:", "", szSrc);

if (szSrc = "") then
    MsgBox("Please enter the company name.", SEVERE);
    goto Name;
endif;

```

A goto statement in the main program must specify a label that has been declared in the main program. A goto statement in a function must specify a label that has been declared in that function.



**Note** • You cannot use a goto statement within a try...catch...endcatch statement.

## if

Use an if statement when you want your script to choose between two or more options. An if statement consists of the keyword if, a condition to be evaluated, the keyword then, and the keyword endif followed by a semicolon, as shown below:

```

if (condition) then
    // statements to be executed if condition is true
endif;

```

The condition can be one of the following:

- A Boolean or integer constant, variable or literal.
- An expression that produces a Boolean or integer result.
- A function that returns an integer result.

The parentheses around the condition are optional, but highly recommended for readability.



**Tip** • Many InstallScript functions return a negative value when they fail. When using the result of InstallScript functions as the condition in an if statement, test for failure by using a statement like the one below:

```

if (FunctionA (ParameterOne) < 0) then
    // Statements to handle the failure
else
    // Statements when the function succeeds

```

```
endif;
```

InstallScript provides the following if statement structures:

- [if Structure with goto](#)
- [if-then Structure](#)
- [if-then-else Structure](#)
- [Nested if-then-else Structure](#)
- [elseif Structure](#)

## if Structure with goto

InstallScript supports a special form of the if statement that can be used only with goto statements:

```
if condition goto labelname;
```

This special structure has the following features:

- The condition must be followed by a goto statement.
- The keyword then is not used.
- The keyword endif is not used.

In the following example, the user will be prompted to enter a company name as long as szSrc is a null string ("").

```
Name:
AskText("Company name:", "", szSrc);
if (szSrc = "") goto Name;
```

## if-then Structure

The simplest if statement evaluates an expression and performs a specified action if the expression is true. If the expression is not true, InstallShield ignores the entire statement. For example:

```
if (szStringA = "exit") then
    AskYesNo ( "Are you sure you want to exit?", NO );
endif;
```

If szStringA equals "exit", the test evaluates to TRUE (1) and the AskYesNo function is called. If szStringA contains anything else, the result is FALSE (0) and the entire statement is ignored.

The sample code below compares the values of the variable nDialog and the constant DLG\_ERR. If they are equivalent, InstallShield executes the MessageBox function:

```
if (nDialog = DLG_ERR) then
    MessageBox ("Error has occurred", WARNING);
endif;
```



**Tip** • You may find that your if statement is easier to read when you place the expression to be evaluated in parentheses, but the parentheses are optional in InstallScript.

## if-then-else Structure

An if statement can also specify one or more statements to be executed if the condition is false. This option is indicated with the keyword `else`, as shown below:

```
if (condition) then
    // statements to be executed if condition is true
else
    // statements to be executed if condition is false
endif;
```

In the example below, if `szStringA` equals "exit," the test evaluates to TRUE (1), and the `AskYesNo` function is called. If `szStringA` is not equal to "exit," the result is FALSE (0), and the `MessageBox` function is called following the `else` statement.

```
if szStringA = "exit" then
    AskYesNo ("Are you sure you want to exit?", NO );
else
    MessageBox ("Please wait... ", INFORMATION );
endif;
```

## Nested if-then-else Structure

You can create nested if statements, in which one if statement is embedded in another:

```
if (first condition) then
    if (second condition) then
        // statements to be executed if first and
        // second conditions are true
    else
        // statements to be executed if first is true but
        // second condition is false
    endif;
else
    if (third condition) then
        // statements to be executed if first condition is
        // false and third condition is true
    else
        // statements to be executed if first condition is
        // false and third condition is false
    endif;
endif;
```

In the following example, if the value of `szStringA` is "exit", `AskYesNo` is called. If the value of `szStringA` is "exit", the program displays a message box. If `szStringA` is not equal to either of those values, the execution proceeds to the label `UserErrorHandler`.

```
if szStringA = "exit" then
    AskYesNo ("Are you sure you want to exit?", NO);
else
    if szStringA = "continue" then
        MessageBox ("Please wait...", INFORMATION);
    else
        UserErrorHandler;
    endif;
endif;
```

## elseif Structure

InstallScript provides the elseif statement to create if structures in which the else branch of one if statement leads to another if statement:

```
if (first condition) then
    // statements to be executed if first condition
    // is true
elseif (second condition) then
    // statements to be executed if first condition
    // is false and second condition is true
elseif (third condition) then
    // statements to be executed if first and second
    // conditions are false and third condition is
    // true
endif;
```

In the following example, if szStringA equals "exit," AskYesNo is called. If szStringA is not equal to "exit," the program continues to the elseif statement to test if szStringA is equal to "continue." If szStringA is equal to "continue," the result is TRUE and MessageBox is called. If szStringA is not equal to "continue," the program moves to the next elseif, and so on.

```
if szStringA = "exit" then
    AskYesNo ("Are you sure you want to exit?", NO );
elseif szStringA = "continue" then
    MessageBox ("Please wait...", INFORMATION );
elseif szStringA = "reboot" then
    goto StartHere;
endif;
```




---

**Note** • You cannot define a label within an if statement.

## method




---

**Project** • This information applies to InstallScript projects.

The method keyword is used to declare a method in an object script with the following syntax:

```
method <return variable type> <method name> ( <argument variable type(s)> );
```

For example:

```
method STRING MyMethod ( STRING, NUMBER );
```

If you add a method to the object project by using the Add New Method dialog box, a method declaration is automatically placed in the object script.

## property()




---

**Project** • This information applies to InstallScript projects.

The `property()` keyword is used to declare a property and its get and/or put procedures in an object script with the following syntax:

**Table 1 • Property() Keyword Declarations**

Access	Declaration
<b>Read only</b>	<code>property(get) &lt;return variable type&gt; &lt;property name&gt; ( &lt;argument variable type(s)&gt; );</code>
<b>Write only</b>	<code>property(put) &lt;return variable type&gt; &lt;property name&gt; ( &lt;argument variable type(s)&gt; );</code>
<b>Read/write</b>	<code>property(get,put) &lt;return variable type&gt; &lt;property name&gt; ( &lt;argument variable type(s)&gt; );</code>

For example:

```
property(get,put) STRING MyProperty ( NUMBER );
```

If you add a property to the object project by using the Add New Property dialog box, a property declaration is automatically placed in the object script.

## prototype

The `prototype` keyword tells the InstallScript compiler that the line of code contains a function definition. To learn how to use this keyword, see [Declaring Functions](#).

## repeat...until

The `repeat` statement is analogous to the `do...while` loop in the C language. It also is very similar to the InstallScript `while` statement.

There are two main differences between `repeat` and `while` in InstallScript:

- The `repeat` statement must loop at least once. A `while` statement might not loop at all.
- A `while` statement terminates when the expression evaluates as false. A `repeat` statement terminates when the expression evaluates as true.



### Task

#### **To create a repeat loop:**

1. Set the variable you will be using in the conditional test as you would for a `while` loop.
2. Type `repeat` on its own line with no punctuation.
3. Build the operation(s) that you want repeated.
4. Add the operation that changes the test variable (for example, `nCount = nCount + 1`, or `nCount = SomeVariable`).
5. End the loop with an `until` statement containing the conditional test in parentheses.

The following example demonstrates repeat loop syntax:

```
nCount = 1;
repeat
    MessageBox("Count is less than 5", INFORMATION);
    nCount = nCount + 1;
until (nCount = 5);
```




---

**Note** • You cannot define a label within the repeat statement.

## return

You can use the return statement to return a value from a user-defined function (if the function prototype does not specify a return type of void). When a return statement is encountered, program flow returns to the point at which the function was called. When used to return from a call to a user-defined function, the return statement can return a specified value to the caller.

The return value of most built-in functions will be either 0 (zero), indicating the success of the function, or a value less than zero (< 0), indicating failure. You can assign a number to the return value by using a return statement above the end statement in the function block, as shown below:

```
    return -1;
end;
```

This attribute allows you to return the value of a local variable to the caller, even though the local variable itself is destroyed:

```
function MyFunction(ParamOne, ParamTwo)
    NUMBER nNumber;
begin
    nNumber = (ParamOne + ParamTwo);
    // . . .
    return nNumber;
end;
```

## set

The set keyword must precede the assignment of an OBJECT variable to a reference returned by the [CreateObject](#) function. For example:

```
function OnBegin( )
    OBJECT oMSI;
begin
    // create the object
    set oMSI = CreateObject("WindowsInstaller.Installer");

    // use the object (display MSI version on user's system)
    MessageBox("Your MSI version is: " + oMSI.Version, INFORMATION);

    // free the object
    set oMSI = NOTHING;
end;
```





**Note** • You can use the keywords `try-catch-endcatch` for more control over exception handling for COM objects.

## stdcall



**Project** • This information applies to InstallScript projects. The `stdcall` keyword is used when declaring an external DLL function that uses the `stdcall` calling convention. For example:

```
prototype stdcall POINTER kernel32.lstrcpy( byref string, byref string);
```

If no calling convention is specified, `stdcall` is assumed.

Most Windows API functions use the `stdcall` (WINAPI) calling convention. Consult Microsoft documentation for more information about calling conventions.

## switch...endswitch

The switch statement is similar to the [elseif Structure](#) statement. Use the switch statement to execute one of several different sections of code, depending on the value of an expression. The switch statement evaluates the expression and then branches to the case statement whose constant value matches the result of the expression. If no match is found among the case statements, control passes to a default statement, if one has been specified.

### Creating Switch Statements



#### Task

#### To create a switch statement:

1. Type the keyword `switch`, followed by the expression to be evaluated. The expression—which can be a constant, variable, arithmetic expression, logical expression, or function result—must be enclosed within parentheses. Do not punctuate this line.
2. For each option, type the keyword `case` and one or more constants followed by a colon. If more than one constant is specified, delimit them with commas. Note that only constants can be specified here. Specifying a variable name, string identifier, function result, or other type of expression after the keyword `case` results in an error.
3. For each case, follow the colon with the statement or statements to be executed for that option. Terminate each statement with a semicolon.
4. After all case statements have been specified, use the keyword `default`, followed by a colon (:), to control the program when the expression does not match any of the stated cases.
5. Close the block with the keyword `endswitch`, followed by a semicolon (;).

### Example Script

The following script segment displays the current video resolution of the computer on which it is executed:

```
STRING szMsg, svResult;
NUMBER nvResult;
```

```

GetSystemInfo (VIDEO, nvResult, svResult);

switch (nvResult)
    case IS_UNKNOWN:
        szMsg = "The user's video is unknown.";
    case IS_EGA:
        szMsg = "EGA resolution.";
    case IS_VGA:
        szMsg = "VGA resolution.";
    case IS_SVGA:
        szMsg = "Super VGA (800 x 600) resolution.";
    case IS_XVGA:
        szMsg = "XVGA (1024 x 768) resolution.";
    case IS_UVGA:
        szMsg = "Greater than 1024 x 768 resolution.";
    default:
        szMsg = "Error";
endswitch;

MessageBox (szMsg, INFORMATION);

```



**Note** • Only one case block is executed each time a switch statement is executed. After InstallShield executes a case block, it executes the next statement after the endswitch. A switch block can be quite useful inside of a while loop. By using the case statements as flags, you can create a loop with optional exit points.

## try, catch, and endcatch

The keywords try, catch, and endcatch are used for exception handling. For more information on exception handling, see [Exception Handling](#).



**Note** • You cannot use a goto statement within a try...catch...endcatch statement. In addition, you cannot define a label within a try...catch...endcatch statement.

## void

Void is not a true data type, in the sense that a variable cannot be declared as type void. Void is only used in function prototypes to indicate that the function does not return a value, as in the following:

```

prototype void Subroutine(int);

function void Subroutine(int);
begin
    // perform operations, but
    // do not return a value
end;

```

# while...endwhile

Use the while statement when you want to execute one or more statements repeatedly, as long as a particular condition is true. If the condition is not true when the statement is first executed, the loop is not performed.



## Task

### To create a while loop:

1. Set the variable you are using as the condition to an initial state.
2. Type the keyword while, followed by the conditional test in parentheses. Do not punctuate this line.
3. Build the operation(s) that you want repeated.
4. Add the operation that changes the test variable (for example, nCount = nCount + 1, or nCount = SomeVariable).
5. End the loop by typing endwhile, followed by a semicolon.

In the following example, the message box is displayed four times.

```
nCount = 1;

while (nCount < 5)
    MessageBox ("This is still true.", INFORMATION);
    nCount = nCount + 1;
endwhile;
```

Because nCount is assigned an initial value of 1, the while statement evaluates TRUE the first time it is executed; the message box is displayed and nCount is incremented by 1. After the fourth pass through the loop, nCount is equal to 5; the while statement evaluates FALSE and the program continues executing with the statement after endwhile.



**Note** • You cannot define a label within a while block. You can, however, nest while statements in InstallScript. You must end each while block with endwhile.

## Nested while Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/* This script illustrates a nested while loop.
 * It searches for the specified type of files and
 * shows the number of lines in each file.      */

#define SOURCEDIR "c:\\example";

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_Nested while(HWND);

function ExFn_Nested while(hMSI)
```

```

LIST    listID;
STRING  svTarget, svResult, filename, svLine, szPath, szFileName;
NUMBER  nResult, nOp, nFileHandle, count;
begin

    count = 0;
    nOp = RESET;
    svTarget = SOURCEDIR;
    listID = ListCreate (STRINGLIST);

    while FindAllFiles (svTarget, "*.txt", svResult, nOp) = 0;

        // To get the name of the file in the fully specified path
        StrGetTokens(listID,svResult,"\\");
        ListCurrentString(listID,filename);

        // Set the file mode to normal.
        OpenFileMode(FILE_MODE_NORMAL);
        szFileName = filename;
        szPath      = svTarget;

        // The following opens the file for editing.
        OpenFile(nFileHandle, szPath, szFileName);

/*-----*\
*
* The following retrieves each line of text from the open file and increments
* a count to find the number of lines.
*
*\-----*/

        while (GetLine (nFileHandle, svLine) = 0)
            count = count + 1;
        endwhile;

        sprintfBox(INFORMATION,"The Total lines in the file",
            "The No. of lines in the file %s is %d",filename,count);

        count = 0;

        // The following closes the file
        CloseFile(nFileHandle);

        // Continue searching files where last file was left off
        nOp = CONTINUE;

        if (FindAllFiles (svTarget, "*.txt", svResult, nOp) < 0) then
            abort;
        endif;

    endwhile;

end;

```

# Flow Control

Like most programming languages, InstallScript processes statements within a function block sequentially, starting with the first statement and ending with the last. The linear flow of execution within a function block can be controlled with conditional statements that perform branching and iteration.

Branching is most commonly performed with an if statement that directs execution down one path or another. Iteration is performed with loop statements that execute one or more statements repeatedly, either for a set number of times or as long as a specified condition is met.

To control the flow of execution within scripts, InstallShield provides the following keywords:

- abort
- exit
- for...endfor
- goto
- if...then...else...endif
- repeat...until
- return
- switch...endswitch
- while...endwhile



# Predefined Constants

A predefined constant is an identifier reserved by InstallScript to represent a specific literal value. InstallScript uses predefined constants to represent certain data values that are passed to and returned by built-in functions. By using these predefined constants rather than literal values, you can improve the readability of your setup scripts.

You cannot change the value InstallShield assigns to a predefined constant. However, you can determine the value of a predefined constant by calling [SprintfBox](#), as shown in the example below, which displays the value of the predefined constant `FEATURE_FIELD_SELECTED`:

```
SprintfBox (INFORMATION, "", "%d", FEATURE_FIELD_SELECTED);
```

Although you can use a literal value in place of a predefined constant, it is strongly recommended that you use predefined constants wherever indicated for a function.

Following is a list of the predefined constants used by InstallScript.

- 
- 
- 
- 

## AFTER

AFTER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [EzBatchAddString](#)
- [EzBatchAddPath](#)
- [ConfigAdd](#)
- [ConfigMove](#)
- [ListAddItem](#)
- [ListAddString](#)
- [PathAdd](#)
- [PathMove](#)
- [BatchAdd](#)
- [PathMove](#)
- [BatchMoveEx](#)
- [FileInsertLine](#)
- [EzConfigAddDriver](#)

- [EzConfigAddString](#)

## ALLCONTENTS

ALLCONTENTS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DeleteDir](#)

## ALLCONTROLS

ALLCONTROLS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [CtrlSetFont](#)

## APPEND

APPEND is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FileInsertLine](#)

## ASKDESTPATH

ASKDESTPATH is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [AskDestPath](#)

## ASKOPTIONS

ASKOPTIONS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [AskOptions](#)
- [PlaceWindow](#)



# ASKPATH

ASKPATH is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [AskPath](#)
- [PlaceWindow](#)

# ASKTEXT

ASKTEXT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [AskText](#)
- [PlaceWindow](#)

# BACK

BACK is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [AskDestPath](#)
- [AskOptions](#)
- [AskPath](#)
- [AskText](#)
- [FeatureDialog](#)
- [SdAskDestPath](#)
- [SdAskOptions](#)
- [SdAskOptionsList](#)
- [SdBitmap](#)
- [SdDisplayTopics](#)
- [SdFeatureDialog](#)
- [SdFeatureDialog2](#)
- [SdFeatureDialogAdv](#)
- [SdFeatureMult](#)

- [SdLicense](#)
- [SdOptionsButtons](#)
- [SdRegisterUser](#)
- [SdRegisterUserEx](#)
- [SdSelectFolder](#)
- [SdShowAnyDialog](#)
- [SdShowDlgEdit1](#)
- [SdShowDlgEdit2](#)
- [SdShowDlgEdit3](#)
- [SdShowFileMods](#)
- [SdShowInfoList](#)
- [SdStartCopy](#)
- [SdWelcome](#)
- [SelectFolder](#)
- [Welcome](#)

## BACKBUTTON

BACKBUTTON is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Disable](#)
- [Enable](#)
- [Is](#)

## BACKGROUND

BACKGROUND is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [PlaceWindow](#)
- [SetColor](#)
- [Enable](#)
- [SizeWindow](#)

- [Disable](#)

## BACKGROUNDCAPTION

BACKGROUNDCAPTION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetTitle](#)

## BASEMEMORY



**Project** • This information applies to InstallScript projects.

BASEMEMORY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## BEFORE

BEFORE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [PathMove](#)
- [FileInsertLine](#)
- [EzBatchAddPath](#)
- [EzBatchAddString](#)
- [BatchAdd](#)
- [BatchMoveEx](#)
- [EzConfigAddDriver](#)
- [EzConfigAddString](#)
- [ConfigAdd](#)
- [ConfigMove](#)
- [ListAddItem](#)
- [ListAddString](#)

- [PathAdd](#)

## BIF\_BROWSEFORCOMPUTER



**Project** • This information applies to InstallScript projects.

BIF\_BROWSEFORCOMPUTER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SelectDirEx](#)

## BIF\_BROWSEFORPRINTER



**Project** • This information applies to InstallScript projects.

BIF\_BROWSEFORPRINTER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SelectDirEx](#)

## BIF\_DONTGOBELOWDOMAIN



**Project** • This information applies to InstallScript projects.

BIF\_DONTGOBELOWDOMAIN is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SelectDirEx](#)

## BIF\_EDITBOX



**Project** • This information applies to InstallScript projects.

BIF\_EDITBOX is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SelectDirEx](#)

## BIF\_RETURNFSANCESTORS



**Project** • This information applies to InstallScript projects.

BIF\_RETURNFSANCESTORS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SelectDirEx](#)

## BIF\_RETURNONLYFSDIRS



**Project** • This information applies to InstallScript projects.

BIF\_RETURNONLYFSDIRS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SelectDirEx](#)

## BIF\_STATUSTEXT



**Project** • This information applies to InstallScript projects.

BIF\_STATUSTEXT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SelectDirEx](#)

## BILLBOARD

BILLBOARD is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Disable](#)
- [PlaceWindow](#)

## BITMAPICON

BITMAPICON is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [PlaceBitmap](#)

## BK\_BLUE

BK\_BLUE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetColor](#)

## BK\_GREEN

BK\_GREEN is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetColor](#)

## BK\_MAGENTA

BK\_MAGENTA is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetColor](#)

## BK\_ORANGE

BK\_ORANGE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetColor](#)

## BK\_PINK

BK\_PINK is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetColor](#)

## BK\_RED

BK\_RED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetColor](#)

## BK\_SMOOTH

BK\_SMOOTH is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetColor](#)

## BK\_SOLIDBLACK

BK\_SOLIDBLACK is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetColor](#)

## BK\_SOLIDBLUE

BK\_SOLIDBLUE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetColor](#)

## BK\_SOLIDGREEN

BK\_SOLIDGREEN is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetColor](#)

## BK\_SOLIDMAGENTA

BK\_SOLIDMAGENTA is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetColor](#)

## BK\_SOLIDORANGE

BK\_SOLIDORANGE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetColor](#)

## BK\_SOLIDPINK

BK\_SOLIDPINK is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetColor](#)

## BK\_SOLIDRED

BK\_SOLIDRED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



### Used With

- [SetColor](#)

## BK\_SOLIDWHITE

BK\_SOLIDWHITE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetColor](#)

## BK\_SOLIDYELLOW

BK\_SOLIDYELLOW is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetColor](#)

## BK\_YELLOW

BK\_YELLOW is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetColor](#)

## BLACK

BLACK is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetTitle](#)

## BLUE

BLUE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetColor](#)
- [SetTitle](#)

## BOOTUPDRIVE

BOOTUPDRIVE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## BUTTON\_CHECKED

BUTTON\_CHECKED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [CtrlSetState](#)
- [CtrlGetState](#)

## BUTTON\_UNCHECKED

BUTTON\_UNCHECKED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [CtrlSetState](#)
- [CtrlGetState](#)

## BYTES

BYTES is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ConvertSizeToUnits](#)
- [StrConvertSizeUnit](#)

# CANCEL

CANCEL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [SelectDir](#)

# CANCELBUTTON

CANCELBUTTON is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [Disable](#)
- [Enable](#)
- [Is](#)

# CDROM

CDROM is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetSystemInfo](#)

# CDROM\_DRIVE

CDROM\_DRIVE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetValidDrivesList](#)

# CENTERED

CENTERED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [PlaceWindow](#)

- [PlaceBitmap](#)

## CHECKBOX

CHECKBOX is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DialogSetInfo](#)

## CHECKBOX95

CHECKBOX95 is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DialogSetInfo](#)

## CHECKLINE

CHECKLINE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DialogSetInfo](#)

## CHECKMARK

CHECKMARK is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DialogSetInfo](#)

## CLEAR\_FILE\_ATTR

CLEAR\_FILE\_ATTR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [XCopyFile](#)

## COLORS

COLORS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## COMMAND

COMMAND is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [BatchMoveEx](#)
- [ConfigFind](#)
- [EzBatchAddString](#)
- [BatchAdd](#)
- [BatchDeleteEx](#)

## COMMON

COMMON is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ProgDefGroupType](#)

## COMPACT

COMPACT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetupType](#)
- [SdSetupType](#)

## COMPARE\_DATE

COMPARE\_DATE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [FileCompare](#)

## COMPARE\_MD5\_SIGNATURE

COMPARE\_MD5\_SIGNATURE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [FileCompare](#)

## COMPARE\_SIZE

COMPARE\_SIZE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [FileCompare](#)

## COMPARE\_VERSION

COMPARE\_VERSION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [FileCompare](#)

## COMP\_NORMAL

COMP\_NORMAL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [XCopyFile](#)

## COMP\_UPDATE\_DATE

COMP\_UPDATE\_DATE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [XCopyFile](#)

## COMP\_UPDATE\_SAME

COMP\_UPDATE\_SAME is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [XCopyFile](#)

## COMP\_UPDATE\_VERSION

COMP\_UPDATE\_VERSION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [XCopyFile](#)

## CONTINUE

CONTINUE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [FileGrep](#)
- [BatchFind](#)
- [FindFile](#)
- [ConfigFind](#)
- [PathFind](#)

## COPY\_ERR\_CREATEDIR

COPY\_ERR\_CREATEDIR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [CopyFile](#)
- [XCopyFile](#)

## COPY\_ERR\_MEMORY

COPY\_ERR\_MEMORY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [CopyFile](#)
- [XCopyFile](#)

## COPY\_ERR\_NODISKSPACE

COPY\_ERR\_NODISKSPACE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [CopyFile](#)
- [XCopyFile](#)

## COPY\_ERR\_OPENINPUT

COPY\_ERR\_OPENINPUT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [CopyFile](#)
- [XCopyFile](#)

## COPY\_ERR\_OPENOUTPUT

COPY\_ERR\_OPENOUTPUT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [CopyFile](#)
- [XCopyFile](#)

## COPY\_ERR\_TARGETREADONLY

COPY\_ERR\_TARGETREADONLY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



**Used With**

- [CopyFile](#)
- [XCopyFile](#)

## CPU

CPU is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [GetSystemInfo](#)

## CS\_OPTION\_FLAG\_NO\_NEW\_INSTALL\_HIGHLIGHT

CS\_OPTION\_FLAG\_NO\_NEW\_INSTALL\_HIGHLIGHT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [CreateShortcut](#)
- [ReplaceShortcut](#)

## CS\_OPTION\_FLAG\_NO\_STARTSCREEN\_PIN

CS\_OPTION\_FLAG\_NO\_STARTSCREEN\_PIN is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [CreateShortcut](#)
- [ReplaceShortcut](#)

## CS\_OPTION\_FLAG\_PREVENT\_PINNING

CS\_OPTION\_FLAG\_PREVENT\_PINNING is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [CreateShortcut](#)
- [ReplaceShortcut](#)

## CS\_OPTION\_FLAG\_REPLACE\_EXISTING

CS\_OPTION\_FLAG\_REPLACE\_EXISTING is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [CreateShortcut](#)
- [ReplaceShortcut](#)

## CS\_OPTION\_FLAG\_RUN\_MAXIMIZED

CS\_OPTION\_FLAG\_RUN\_MAXIMIZED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [CreateShortcut](#)
- [ReplaceShortcut](#)

## CS\_OPTION\_FLAG\_RUN\_MINIMIZED

CS\_OPTION\_FLAG\_RUN\_MINIMIZED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [CreateShortcut](#)
- [ReplaceShortcut](#)

## CURRENTROOTKEY

CURRENTROOTKEY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [VarRestore](#)
- [VarSave](#)

## CUSTOM

CUSTOM is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [SetupType](#)
- [SdSetupType](#)

## DATA\_COMPONENT

DATA\_COMPONENT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [SilentReadData](#)
- [SilentWriteData](#)

## DATA\_LIST

DATA\_LIST is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [SilentWriteData](#)

## DATA\_NUMBER

DATA\_NUMBER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [SilentReadData](#)
- [SilentWriteData](#)

## DATA\_STRING

DATA\_STRING is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [SilentReadData](#)
- [SilentWriteData](#)

## DATE

DATE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## DEFAULT

DEFAULT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [switch...endswitch](#)

## DEFWINDOWMODE

DEFWINDOWMODE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Enable](#)

## DELETE



---

**Project** • This information applies to InstallScript projects.

DELETE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)
- [SERVICE\\_IS\\_PARAMS](#)

## DELETE\_EOF

DELETE\_EOF is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FileDeleteLine](#)

## DIALOGCACHE

DIALOGCACHE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Disable](#)
- [Enable](#)

## DIFXAPI\_ERROR

DIFXAPI\_ERROR is a predefined constant that is used to represent a value that is available for use with one or more event handlers. You cannot change the value of a predefined constant.

### Used With

- [OnDIFxLogCallback](#)

## DIFXAPI\_INFO

DIFXAPI\_INFO is a predefined constant that is used to represent a value that is available for use with one or more event handlers. You cannot change the value of a predefined constant.

### Used With

- [OnDIFxLogCallback](#)

## DIFXAPI\_SUCCESS

DIFXAPI\_SUCCESS is a predefined constant that is used to represent a value that is available for use with one or more event handlers. You cannot change the value of a predefined constant.

### Used With

- [OnDIFxLogCallback](#)

## DIFXAPI\_WARNING

DIFXAPI\_WARNING is a predefined constant that is used to represent a value that is available for use with one or more event handlers. You cannot change the value of a predefined constant.

### Used With

- [OnDIFxLogCallback](#)

## DIRECTORY

DIRECTORY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ParsePath](#)

## DIR\_WRITEABLE

DIR\_WRITEABLE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Is](#)

## DISABLE\_ALLUSERBTN



---

**Project** • This information applies to *InstallScript* projects.

DISABLE\_ALLUSERBTN is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

The DISABLE\_ALLUSERBTN constant indicates that the all-users option should be disabled (or hidden) in cases where it would normally be enabled. The default value of this variable is FALSE. Note that the all-users option is always hidden if the installation is being run without administrator or power-user privileges, regardless of the value of this variable.

### Used With

- [SdCustomerInformation](#)
- [SdCustomerInformationEx](#)

## DISABLE\_PERUSERBTN

DISABLE\_PERUSERBTN is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

The `DISABLE_PERUSERBTN` constant indicates that the per-user option should be disabled (or hidden if `HIDE_DISABLED_BTNS` is `TRUE`) in cases where it would normally be enabled. The default value of this variable is `FALSE`. Note that the per-user option is always hidden on Windows 9x platforms, regardless of the value of this variable.

### Used With

- [SdCustomerInformation](#)
- [SdCustomerInformationEx](#)

## DISK

`DISK` is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ParsePath](#)

## DISK1FEATURE



**Project** • This information applies to *InstallScript* projects.

`DISK1FEATURE` is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

`DISK1FEATURE` specifies the feature with the files needed for maintenance setups and uninstallation. (Note that this feature is automatically placed in your `.cab` files by the media builder and is not displayed in the IDE.)

### Used With

- [FeatureSelectItem](#)
- [FeatureIsItemSelected](#)

## DISK\_INFO\_QUERY\_ALL

`DISK_INFO_QUERY_ALL` is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetDiskInfo](#)

## DISK\_INFO\_QUERY\_BYTES\_PER\_CLUSTER

DISK\_INFO\_QUERY\_BYTES\_PER\_CLUSTER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetDiskInfo](#)

## DISK\_INFO\_QUERY\_DISK\_FREE\_SPACE

DISK\_INFO\_QUERY\_DISK\_FREE\_SPACE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetDiskInfo](#)

## DISK\_INFO\_QUERY\_DISK\_TOTAL\_SPACE

DISK\_INFO\_QUERY\_DISK\_TOTAL\_SPACE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetDiskInfo](#)

## DISK\_INFO\_QUERY\_DRIVE\_TYPE

DISK\_INFO\_QUERY\_DRIVE\_TYPE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetDiskInfo](#)

## DISK\_TOTALSPACE

DISK\_TOTALSPACE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)



## DISK\_TOTALSPACE\_EX

DISK\_TOTALSPACE\_EX is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## DLG\_ASK\_OPTIONS

DLG\_ASK\_OPTIONS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetDialogTitle](#)

## DLG\_ASK\_PATH

DLG\_ASK\_PATH is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetDialogTitle](#)

## DLG\_ASK\_TEXT

DLG\_ASK\_TEXT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetDialogTitle](#)

## DLG\_ASK\_YESNO

DLG\_ASK\_YESNO is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetDialogTitle](#)

## DLG\_CENTERED

DLG\_CENTERED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DefineDialog](#)

## DLG\_CLOSE

DLG\_CLOSE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [WaitOnDialog](#)

## DLG\_DIR\_DIRECTORY

DLG\_DIR\_DIRECTORY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [CtrlDir](#)

## DLG\_DIR\_DRIVE

DLG\_DIR\_DRIVE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [CtrlDir](#)

## DLG\_DIR\_FILE

DLG\_DIR\_FILE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [CtrlDir](#)

## DLG\_ENTER\_DISK

DLG\_ENTER\_DISK is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetDialogTitle](#)

## DLG\_ERR

DLG\_ERR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [WaitOnDialog](#)
- [EzDefineDialog](#)
- [ReleaseDialog](#)
- [CtrlGetState](#)
- [DefineDialog](#)

## DLG\_ERR\_ALREADY\_EXISTS

DLG\_ERR\_ALREADY\_EXISTS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DefineDialog](#)
- [EzDefineDialog](#)

## DLG\_ERR\_ENDDLG

DLG\_ERR\_ENDDLG is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ReleaseDialog](#)

## DLG\_INFO\_ALTIMAGE

DLG\_INFO\_ALTIMAGE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DialogSetInfo](#)

## DLG\_INFO\_ALTIMAGE\_HIDPI

DLG\_INFO\_ALTIMAGE\_HIDPI is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DialogSetInfo](#)

## DLG\_INFO\_ALTIMAGE\_REVERT\_IMAGE

DLG\_INFO\_ALTIMAGE\_REVERT\_IMAGE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DialogSetInfo](#)

## DLG\_INFO\_ALTIMAGE\_VERIFY\_BMP

DLG\_INFO\_ALTIMAGE\_VERIFY\_BMP is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DialogSetInfo](#)

## DLG\_INFO\_CHECKSELECTION

This constant is now obsolete. The checkbox controls in InstallScript are now drawn automatically using your current Windows theme to provide a modern and uniform look that is compatible with high DPI displays.

## DLG\_INFO\_KUNITS

DLG\_INFO\_KUNITS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DialogSetInfo](#)

## DLG\_INFO\_USEDECIMAL

DLG\_INFO\_USEDECIMAL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DialogSetInfo](#)

## DLG\_INIT

DLG\_INIT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [WaitOnDialog](#)

## DLG\_MSG\_ALL

DLG\_MSG\_ALL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DefineDialog](#)

## DLG\_MSG\_INFORMATION

DLG\_MSG\_INFORMATION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetDialogTitle](#)

## DLG\_MSG\_SEVERE

DLG\_MSG\_SEVERE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetDialogTitle](#)

## DLG\_MSG\_STANDARD

DLG\_MSG\_STANDARD is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DefineDialog](#)

## DLG\_MSG\_WARNING

DLG\_MSG\_WARNING is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetDialogTitle](#)

## DLG\_STATUS

DLG\_STATUS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetDialogTitle](#)

## DLG\_USER\_CAPTION

DLG\_USER\_CAPTION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetDialogTitle](#)

## DOINSTALL\_OPTION\_NOHIDEPROGRESS

DOINSTALL\_OPTION\_NOHIDEPROGRESS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DoInstall](#)

## DOINSTALL\_OPTION\_NOHIDESPLASH

DOINSTALL\_OPTION\_NOHIDESPLASH is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DoInstall](#)

## DOINSTALL\_OPTION\_NOLANGSWITCH

DOINSTALL\_OPTION\_NOLANGSWITCH is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DoInstall](#)

## DOINSTALL\_OPTION\_NOSETBATCHINSTALL

DOINSTALL\_OPTION\_NOSETBATCHINSTALL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DoInstall](#)

## DOTNETFRAMEWORKINSTALLED

DOTNETFRAMEWORKINSTALLED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Is](#)

## DOTNETSERVICEPACKINSTALLED

DOTNETSERVICEPACKINSTALLED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Is](#)

# DRIVE

DRIVE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetSystemInfo](#)

# DRIVE\_CDROM

DRIVE\_CDROM is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetDiskInfo](#)

# DRIVE\_FIXED

DRIVE\_FIXED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetDiskInfo](#)

# DRIVE\_NO\_ROOT\_DIR

DRIVE\_NO\_ROOT\_DIR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetDiskInfo](#)

# DRIVE\_RAMDISK

DRIVE\_RAMDISK is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetDiskInfo](#)



## DRIVE\_REMOTE

DRIVE\_REMOTE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetDiskInfo](#)

## DRIVE\_REMOVABLE

DRIVE\_REMOVABLE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetDiskInfo](#)

## DRIVE\_UNKNOWN

DRIVE\_UNKNOWN is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetDiskInfo](#)

## DRIVER\_PACKAGE\_DELETE\_FILES

DRIVER\_PACKAGE\_DELETE\_FILES is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DIFxDriverPackageUninstall](#)

## DRIVER\_PACKAGE\_FORCE

DRIVER\_PACKAGE\_FORCE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DIFxDriverPackageInstall](#)
- [DIFxDriverPackagePreinstall](#)
- [DIFxDriverPackageUninstall](#)

## DRIVER\_PACKAGE\_LEGACY\_MODE

DRIVER\_PACKAGE\_LEGACY\_MODE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DIFxDriverPackageInstall](#)
- [DIFxDriverPackagePreinstall](#)

## DRIVER\_PACKAGE\_ONLY\_IF\_DEVICE\_PRESENT

DRIVER\_PACKAGE\_ONLY\_IF\_DEVICE\_PRESENT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DIFxDriverPackageInstall](#)
- [DIFxDriverPackagePreinstall](#)

## DRIVER\_PACKAGE\_REPAIR

DRIVER\_PACKAGE\_REPAIR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DIFxDriverPackageInstall](#)
- [DIFxDriverPackagePreinstall](#)
- [DIFxDriverPackageUninstall](#)

## DRIVER\_PACKAGE\_SILENT

DRIVER\_PACKAGE\_SILENT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DIFxDriverPackageInstall](#)
- [DIFxDriverPackagePreinstall](#)

## EDITBOX\_CHANGE

EDITBOX\_CHANGE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [CtrlGetSubCommand](#)

## EFF\_BOXSTRIPE

EFF\_BOXSTRIPE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetDisplayEffect](#)

## EFF\_FADE

EFF\_FADE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetDisplayEffect](#)

## EFF\_HORIZREVEAL

EFF\_HORIZREVEAL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetDisplayEffect](#)

## EFF\_HORIZSTRIPE

EFF\_HORIZSTRIPE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetDisplayEffect](#)

## EFF\_NONE

EFF\_NONE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetDisplayEffect](#)

## EFF\_REVEAL

EFF\_REVEAL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetDisplayEffect](#)

## EFF\_VERTSTRIPE

EFF\_VERTSTRIPE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetDisplayEffect](#)

## END\_OF\_FILE

END\_OF\_FILE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FileGrep](#)

## END\_OF\_LIST

END\_OF\_LIST is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ListCurrentItem](#)
- [ListCurrentString](#)
- [ListGetFirstItem](#)
- [ListSetIndex](#)
- [ListDeleteItem](#)
- [ListDeleteString](#)
- [ListFindItem](#)

- [ListFindString](#)
- [ListCurrentString](#)
- [ListGetNextItem](#)
- [ListGetNextString](#)
- [ListSetCurrentItem](#)
- [ListSetCurrentString](#)

## ENTERDISK

ENTERDISK is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [EnterDisk](#)
- [PlaceWindow](#)

## EQUALS

EQUALS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [VerCompare](#)
- [FileCompare](#)

## ERROR\_ACCESS\_DENIED



---

**Project** • This information applies to *InstallScript* projects.

ERROR\_ACCESS\_DENIED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetExtendedErrInfo](#)

## ERROR\_CIRCULAR\_DEPENDENCY



**Project** • This information applies to InstallScript projects.

ERROR\_CIRCULAR\_DEPENDENCY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetExtendedErrInfo](#)

## ERROR\_DATABASE\_DOES\_NOT\_EXIST



**Project** • This information applies to InstallScript projects.

ERROR\_DATABASE\_DOES\_NOT\_EXIST is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetExtendedErrInfo](#)

## ERROR\_DEPENDENT\_SERVICES\_RUNNING



**Project** • This information applies to InstallScript projects.

ERROR\_DEPENDENT\_SERVICES\_RUNNING is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetExtendedErrInfo](#)

## ERROR\_DUP\_NAME



**Project** • This information applies to InstallScript projects.

ERROR\_DUP\_NAME is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetExtendedErrInfo](#)

# ERROR\_FILE\_NOT\_FOUND



**Project** • This information applies to InstallScript projects.

ERROR\_FILE\_NOT\_FOUND is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetExtendedErrInfo](#)

# ERROR\_INVALID\_HANDLE



**Project** • This information applies to InstallScript projects.

ERROR\_INVALID\_HANDLE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetExtendedErrInfo](#)

# ERROR\_INVALID\_PARAMETER



**Project** • This information applies to InstallScript projects.

ERROR\_INVALID\_PARAMETER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetExtendedErrInfo](#)

# ERROR\_INVALID\_SERVICE\_ACCOUNT



**Project** • This information applies to InstallScript projects.

ERROR\_INVALID\_SERVICE\_ACCOUNT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetExtendedErrInfo](#)

# ERROR\_INVALID\_SERVICE\_CONTROL



**Project** • This information applies to InstallScript projects.

ERROR\_INVALID\_SERVICE\_CONTROL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetExtendedErrInfo](#)

# ERROR\_PATH\_NOT\_FOUND



**Project** • This information applies to InstallScript projects.

ERROR\_PATH\_NOT\_FOUND is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetExtendedErrInfo](#)

# ERROR\_SERVICE\_ALREADY\_RUNNING



**Project** • This information applies to InstallScript projects.

ERROR\_SERVICE\_ALREADY\_RUNNING is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetExtendedErrInfo](#)

# ERROR\_SERVICE\_CANNOT\_ACCEPT\_CTRL



**Project** • This information applies to InstallScript projects.

ERROR\_SERVICE\_CANNOT\_ACCEPT\_CTRL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetExtendedErrInfo](#)



## ERROR\_SERVICE\_DATABASE\_LOCKED



**Project** • This information applies to InstallScript projects.

ERROR\_SERVICE\_DATABASE\_LOCKED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetExtendedErrInfo](#)

## ERROR\_SERVICE\_DEPENDENCY\_DELETED



**Project** • This information applies to InstallScript projects.

ERROR\_SERVICE\_DEPENDENCY\_DELETED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetExtendedErrInfo](#)

## ERROR\_SERVICE\_DEPENDENCY\_FAIL



**Project** • This information applies to InstallScript projects.

ERROR\_SERVICE\_DEPENDENCY\_FAIL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetExtendedErrInfo](#)

## ERROR\_SERVICE\_DISABLED



**Project** • This information applies to InstallScript projects.

ERROR\_SERVICE\_DISABLED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetExtendedErrInfo](#)

# ERROR\_SERVICE\_DOES\_NOT\_EXIST



**Project** • This information applies to InstallScript projects.

ERROR\_SERVICE\_DOES\_NOT\_EXIST is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetExtendedErrInfo](#)

# ERROR\_SERVICE\_EXISTS



**Project** • This information applies to InstallScript projects.

ERROR\_SERVICE\_EXISTS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetExtendedErrInfo](#)

# ERROR\_SERVICE\_LOGON\_FAILED



**Project** • This information applies to InstallScript projects.

ERROR\_SERVICE\_LOGON\_FAILED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetExtendedErrInfo](#)

# ERROR\_SERVICE\_NOT\_ACTIVE



**Project** • This information applies to InstallScript projects.

ERROR\_SERVICE\_NOT\_ACTIVE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetExtendedErrInfo](#)

# ERROR\_SERVICE\_NO\_THREAD



**Project** • This information applies to InstallScript projects.

ERROR\_SERVICE\_NO\_THREAD is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetExtendedErrInfo](#)

# ERROR\_SERVICE\_REQUEST\_TIMEOUT



**Project** • This information applies to InstallScript projects.

ERROR\_SERVICE\_REQUEST\_TIMEOUT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetExtendedErrInfo](#)

# ERROR\_TIMEOUT



**Project** • This information applies to InstallScript projects.

ERROR\_TIMEOUT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetExtendedErrInfo](#)

# ERR\_ABORT

ERR\_ABORT is a predefined constant that is used to represent a value that is passed to or returned by one or more event handlers. You cannot change the value of a predefined constant.

## Used With

- [OnNextDisk](#)

## ERR\_BOX\_BADPATH

ERR\_BOX\_BADPATH is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetErrorMsg](#)
- [SetErrorTitle](#)

## ERR\_BOX\_BADTAGFILE

ERR\_BOX\_BADTAGFILE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetErrorMsg](#)
- [SetErrorTitle](#)

## ERR\_BOX\_DISKID

ERR\_BOX\_DISKID is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetErrorTitle](#)
- [SetErrorMsg](#)

## ERR\_BOX\_DRIVEOPEN

ERR\_BOX\_DRIVEOPEN is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetErrorTitle](#)
- [SetErrorMsg](#)

## ERR\_IGNORE

ERR\_IGNORE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or event handlers. You cannot change the value of a predefined constant.

### Used With

- [SdExceptions](#)

## ERR\_NO

ERR\_NO is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or event handlers. You cannot change the value of a predefined constant.

### Used With

- [SdExceptions](#)

## ERR\_PERFORM\_AFTER\_REBOOT

ERR\_PERFORM\_AFTER\_REBOOT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or event handlers. You cannot change the value of a predefined constant.

### Used With

- [SdExceptions](#)

## ERR\_RETRY

ERR\_RETRY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or event handlers. You cannot change the value of a predefined constant.

### Used With

- [OnNextDisk](#)
- [SdExceptions](#)

## ERR\_YES

ERR\_YES is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or event handlers. You cannot change the value of a predefined constant.

### Used With

- [SdExceptions](#)

## EXCLUDE\_SUBDIR

EXCLUDE\_SUBDIR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [XCopyFile](#)
- [FindAllDirs](#)

## EXCLUSIVE

EXCLUSIVE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SdAskOptionsList](#)
- [AskOptions](#)
- [SdAskOptions](#)

## EXISTS

EXISTS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ExistsDir](#)
- [ExistsDisk](#)

## EXIT

EXIT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Do](#)
- [HandlerEx](#)

## EXTENDEDMEMORY

EXTENDEDMEMORY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

# EXTENSION\_ONLY

EXTENSION\_ONLY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [ParsePath](#)

# FALSE

FALSE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [AskOptions](#)
- [CtrlSetMultCurSel](#)
- [DialogSetInfo](#)
- [FeatureAddItem](#)
- [FeatureGetData](#)
- [FeatureIsItemSelected](#)
- [FeatureSelectItem](#)
- [FeatureTotalSize](#)
- [LongPathToQuote](#)
- [SdDiskSpace2](#)
- [SelectDir](#)
- [SdShowMsg](#)
- [SQLDatabaseBrowse](#)
- [SQLRTConnect](#)
- [SQLRTConnect2](#)
- [SQLRTConnectDB](#)
- [SQLRTGetDatabases](#)
- [SQLRTGetServers](#)
- [SQLRTGetServers2](#)
- [SQLRTPutConnectionAuthentication](#)
- [SQLRTTestConnection](#)
- [SQLRTTestConnection2](#)

- [SQLServerSelectLogin](#)
- [SQLServerSelectLogin2](#)

## FEATURE\_FIELD\_CDROM\_FOLDER



---

**Project** • This information applies to *InstallScript* projects.

FEATURE\_FIELD\_CDROM\_FOLDER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureGetData](#)

## FEATURE\_FIELD\_DESCRIPTION



---

**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_FIELD\_DESCRIPTION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureGetData](#)
- [FeatureSetData](#)

## FEATURE\_FIELD\_DISPLAYNAME



---

**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_FIELD\_DISPLAYNAME is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureGetData](#)
- [FeatureSetData](#)



# FEATURE\_FIELD\_ENCRYPT



**Project** • This information applies to InstallScript projects.

FEATURE\_FIELD\_ENCRYPT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FeatureGetData](#)
- [FeatureSetData](#)

# FEATURE\_FIELD\_FILENEED



**Project** • This information applies to InstallScript projects.

FEATURE\_FIELD\_FILENEED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FeatureGetData](#)

# FEATURE\_FIELD\_FLAGS



**Project** • This information applies to InstallScript projects.

FEATURE\_FIELD\_FLAGS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FeatureGetData](#)

# FEATURE\_FIELD\_FTPLOCATION



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_FIELD\_FTPLOCATION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureGetData](#)

## FEATURE\_FIELD\_GUID



---

**Project** • This information applies to InstallScript projects.

FEATURE\_FIELD\_GUID is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureGetData](#)

## FEATURE\_FIELD\_HANDLER\_ONINSTALLED



---

**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_FIELD\_HANDLER\_ONINSTALLED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureGetData](#)

## FEATURE\_FIELD\_HANDLER\_ONINSTALLING



---

**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_FIELD\_HANDLER\_ONINSTALLING is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureGetData](#)

# FEATURE\_FIELD\_HANDLER\_ONUNINSTALLED



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_FIELD\_HANDLER\_ONUNINSTALLED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FeatureGetData](#)

# FEATURE\_FIELD\_HANDLER\_ONUNINSTALLING



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_FIELD\_HANDLER\_ONUNINSTALLING is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FeatureGetData](#)

# FEATURE\_FIELD\_HTTPLOCATION



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_FIELD\_HTTPLOCATION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FeatureGetData](#)

# FEATURE\_FIELD\_IMAGE



---

**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_FIELD\_IMAGE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FeatureSetData](#)

# FEATURE\_FIELD\_MISC



---

**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_FIELD\_MISC is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FeatureGetData](#)
- [FeatureSetData](#)

# FEATURE\_FIELD\_PASSWORD



---

**Project** • This information applies to *InstallScript* projects.

FEATURE\_FIELD\_PASSWORD is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FeatureGetData](#)

# FEATURE\_FIELD\_SELECTED



---

**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_FIELD\_SELECTED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FeatureGetData](#)
- [FeatureSetData](#)

# FEATURE\_FIELD\_SIZE



---

**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_FIELD\_SIZE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FeatureGetData](#)
- [FeatureSetData](#)

# FEATURE\_FIELD\_STATUS



---

**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_FIELD\_STATUS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FeatureGetData](#)
- [FeatureSetData](#)

# FEATURE\_FIELD\_VISIBLE



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_FIELD\_VISIBLE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FeatureGetData](#)
- [FeatureSetData](#)

# FEATURE\_INFO\_ATTRIBUTE



**Project** • This information applies to *InstallScript MSI* projects.

FEATURE\_INFO\_ATTRIBUTE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

## Used With

- [FeatureFileInfo](#)

# FEATURE\_INFO\_COMPONENT\_FLAGS



**Project** • This information applies to *InstallScript* projects.

FEATURE\_INFO\_COMPONENT\_FLAGS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

## Used With

- [FeatureFileInfo](#)

# FEATURE\_INFO\_COMPSIZE\_HIGH



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_INFO\_COMPSIZE\_HIGH is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [FeatureFileInfo](#)

## FEATURE\_INFO\_COMPSIZE\_LOW



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_INFO\_COMPSIZE\_LOW is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [FeatureFileInfo](#)

## FEATURE\_INFO\_DATE



**Project** • This information applies to InstallScript projects.

FEATURE\_INFO\_DATE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFileInfo](#)

## FEATURE\_INFO\_DATE\_EX



**Project** • This information applies to InstallScript projects.

FEATURE\_INFO\_DATE\_EX is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFileInfo](#)

## FEATURE\_INFO\_DESTINATION



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_INFO\_DESTINATION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [FeatureFileInfo](#)

## FEATURE\_INFO\_FTPLOCATION



**Project** • This information applies to *InstallScript* projects.

FEATURE\_INFO\_FTPLOCATION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [FeatureFileInfo](#)

## FEATURE\_INFO\_HTTPLOCATION



**Project** • This information applies to *InstallScript* projects.

FEATURE\_INFO\_HTTPLOCATION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [FeatureFileInfo](#)



# FEATURE\_INFO\_LANGUAGE



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_INFO\_LANGUAGE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

## Used With

- [FeatureFileInfo](#)

# FEATURE\_INFO\_MD5\_SIGNATURE



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_INFO\_MD5\_SIGNATURE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FeatureFileInfo](#)

# FEATURE\_INFO\_MISC



**Project** • This information applies to *InstallScript* projects.

FEATURE\_INFO\_MISC is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

## Used With

- [FeatureFileInfo](#)

## FEATURE\_INFO\_ORIGSIZE\_HIGH



---

**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_INFO\_ORIGSIZE\_HIGH is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFileInfo](#)

## FEATURE\_INFO\_ORIGSIZE\_LOW



---

**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_INFO\_ORIGSIZE\_LOW is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFileInfo](#)

## FEATURE\_INFO\_OS



---

**Project** • This information applies to *InstallScript* projects.

FEATURE\_INFO\_OS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [FeatureFileInfo](#)

## FEATURE\_INFO\_OVERWRITE



---

**Project** • This information applies to *InstallScript* projects.

FEATURE\_INFO\_OVERWRITE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [FeatureFileInfo](#)

## FEATURE\_INFO\_PLATFORM\_SUITE



**Project** • This information applies to InstallScript projects.

FEATURE\_INFO\_PLATFORM\_SUITE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [FeatureFileInfo](#)

## FEATURE\_INFO\_TIME



**Project** • This information applies to InstallScript projects.

FEATURE\_INFO\_TIME is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [FeatureFileInfo](#)

## FEATURE\_INFO\_VERSIONLS



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_INFO\_VERSIONLS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [FeatureFileInfo](#)

## FEATURE\_INFO\_VERSIONMS



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_INFO\_VERSIONMS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [FeatureFileInfo](#)

## FEATURE\_INFO\_VERSIONSTR



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

FEATURE\_INFO\_VERSIONSTR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [FeatureFileInfo](#)

## FEATURE\_OPCOST\_UNINSTALL\_FILE

FEATURE\_OPCOST\_UNINSTALL\_FILE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureAddUninstallCost](#)
- [FeatureSpendUninstallCost](#)

## FEATURE\_OPCOST\_UNINSTALL\_REGORINI

FEATURE\_OPCOST\_UNINSTALL\_REGORINI is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureAddUninstallCost](#)
- [FeatureSpendUninstallCost](#)

## FEATURE\_OPCOST\_UNINSTALL\_UNREGFILE

FEATURE\_OPCOST\_UNINSTALL\_UNREGFILE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureAddUninstallCost](#)
- [FeatureSpendUninstallCost](#)

## FEATURE\_VALUE\_CRITICAL



---

**Project** • This information applies to *InstallScript* projects.

FEATURE\_VALUE\_CRITICAL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureGetData](#)

## FEATURE\_VALUE\_HIGHLYRECOMMENDED



---

**Project** • This information applies to *InstallScript* projects.

FEATURE\_VALUE\_HIGHLYRECOMMENDED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureGetData](#)

# FEATURE\_VALUE\_STANDARD



**Project** • This information applies to InstallScript projects.

FEATURE\_VALUE\_STANDARD is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FeatureGetData](#)

# File Attributes



**Project** • This information applies to InstallScript projects.

**Table 1** • File Attributes

Attribute	Description
FILE_ATTR_NORMAL	The file is a normal file.
FILE_ATTR_ARCHIVED	The file is archived.
FILE_ATTR_DIRECTORY	The file is a directory.
FILE_ATTR_HIDDEN	The file is hidden.
FILE_ATTR_READONLY	The file is read-only.
FILE_ATTR_SYSTEM	The file is a system file.

# FILE\_ADD\_FILE

FILE\_ADD\_FILE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [SetObjectPermissions](#)

# FILE\_ADD\_SUBDIRECTORY

FILE\_ADD\_SUBDIRECTORY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## FILE\_ALL\_ACCESS

FILE\_ALL\_ACCESS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## FILE\_APPEND\_DATA

FILE\_APPEND\_DATA is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## FILE\_ATTR\_ARCHIVED

FILE\_ATTR\_ARCHIVED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetFileInfo](#)

## FILE\_ATTR\_HIDDEN

FILE\_ATTR\_HIDDEN is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetFileInfo](#)

## FILE\_ATTR\_NORMAL

FILE\_ATTR\_NORMAL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetFileInfo](#)

## FILE\_ATTR\_READONLY

FILE\_ATTR\_READONLY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetFileInfo](#)

## FILE\_ATTR\_SYSTEM

FILE\_ATTR\_SYSTEM is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetFileInfo](#)

## FILE\_ATTRIBUTE

FILE\_ATTRIBUTE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetFileInfo](#)
- [SetFileInfo](#)

## FILE\_BIN\_CUR

FILE\_BIN\_CUR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SeekBytes](#)

## FILE\_BIN\_END

FILE\_BIN\_END is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



### Used With

- [SeekBytes](#)

## FILE\_BIN\_START

FILE\_BIN\_START is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SeekBytes](#)

## FILE\_DATE

FILE\_DATE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetFileInfo](#)
- [SetFileInfo](#)

## FILE\_DELETE\_CHILD

FILE\_DELETE\_CHILD is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## FILE\_EXECUTE

FILE\_EXECUTE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## FILE\_EXISTS

FILE\_EXISTS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Is](#)

## FILE\_INSTALLED

FILE\_INSTALLED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [VerSearchAndUpdateFile](#)

## FILE\_IS\_LOCKED

FILE\_IS\_LOCKED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [VerUpdateFile](#)
- [VerSearchAndUpdateFile](#)

## FILE\_LINE\_LENGTH

FILE\_LINE\_LENGTH is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FileInsertLine](#)
- [FileGrep](#)

## FILE\_LIST\_DIRECTORY

FILE\_LIST\_DIRECTORY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## FILE\_LOCKED

FILE\_LOCKED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Is](#)

## FILE\_MD5\_SIGNATURE

FILE\_MD5\_SIGNATURE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetFileInfo](#)

## FILE\_MODE\_APPEND

FILE\_MODE\_APPEND is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [OpenFileMode](#)
- [CreateFile](#)

## FILE\_MODE\_APPEND\_UNICODE

FILE\_MODE\_APPEND\_UNICODE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [OpenFileMode](#)

## FILE\_MODE\_BINARY

FILE\_MODE\_BINARY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [WriteBytes](#)
- [OpenFileMode](#)

## FILE\_MODE\_BINARYREADONLY

FILE\_MODE\_BINARYREADONLY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [OpenFileMode](#)

## FILE\_MODE\_NORMAL

FILE\_MODE\_NORMAL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [OpenFileMode](#)

## FILE\_NOT\_FOUND

FILE\_NOT\_FOUND is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [VerGetFileVersion](#)
- [FileGrep](#)
- [FileInsertLine](#)
- [FileCompare](#)
- [FileDeleteLine](#)
- [VerFindFileVersion](#)

## FILE\_NO\_VERSION

FILE\_NO\_VERSION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [VerGetFileVersion](#)
- [VerSearchAndUpdateFile](#)
- [VerFindFileVersion](#)
- [VerUpdateFile](#)

## FILE\_RD\_ONLY

FILE\_RD\_ONLY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [VerUpdateFile](#)
- [FileDeleteLine](#)
- [FileInsertLine](#)
- [VerSearchAndUpdateFile](#)

## FILE\_READ\_ATTRIBUTES

FILE\_READ\_ATTRIBUTES is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## FILE\_READ\_DATA

FILE\_READ\_DATA is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## FILE\_READ\_EA

FILE\_LIST\_DIRECTORY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## FILE\_SHARED\_COUNT



**Project** • This information applies to InstallScript projects.

FILE\_SHARED\_COUNT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetFileInfo](#)

## FILE\_SIZE

FILE\_SIZE (same as FILE\_SIZE\_LOW) is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetFileInfo](#)

## FILE\_SIZE\_HIGH

FILE\_SIZE\_HIGH is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetFileInfo](#)

## FILE\_SIZE\_LOW

FILE\_SIZE\_LOW is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetFileInfo](#)

## FILE\_SRC\_OLD

FILE\_SRC\_OLD is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [VerSearchAndUpdateFile](#)
- [VerUpdateFile](#)

## FILE\_TIME

FILE\_TIME is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetFileInfo](#)
- [SetFileInfo](#)

## FILE\_TRAVERSE

FILE\_TRAVERSE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## FILE\_WRITE\_ATTRIBUTES

FILE\_WRITE\_ATTRIBUTES is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## FILE\_WRITE\_DATA

FILE\_WRITE\_DATA is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## FILE\_WRITE\_EA

FILE\_WRITE\_EA is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## FILE\_WRITEABLE




---

**Project** • This information applies to the following project types:

- *Installscript*
- *InstallScript Object*

FILE\_WRITEABLE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Is](#)

## FILENAME

FILENAME is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ParsePath](#)

## FILENAME\_ONLY

FILENAME\_ONLY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ParsePath](#)

## FIXED\_DRIVE

FIXED\_DRIVE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetValidDrivesList](#)

## FONT\_AVAILABLE

FONT\_AVAILABLE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Is](#)

## FULL

FULL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



### Used With

- [PathAdd](#)
- [PathFind](#)
- [PathMove](#)
- [PathDelete](#)

## FULLSCREEN

FULLSCREEN is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [PlaceBitmap](#)

## FULLSCREENSIZE

FULLSCREENSIZE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [PlaceBitmap](#)

## FULLWINDOWMODE

FULLWINDOWMODE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Enable](#)

## FUNCTION\_EXPORTED



---

**Project** • This information applies to InstallScript projects.

FUNCTION\_EXPORTED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Is](#)

## GBYTES

GBYTES is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ConvertSizeToUnits](#)
- [StrConvertSizeUnit](#)

## GENERIC\_ALL

GENERIC\_ALL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## GENERIC\_EXECUTE

GENERIC\_EXECUTE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## GENERIC\_READ

GENERIC\_READ is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## GENERIC\_WRITE

GENERIC\_WRITE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

# GREATER\_THAN

GREATER\_THAN is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FileCompare](#)
- [VerCompare](#)

# GREEN

GREEN is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [SetColor](#)
- [SetTitle](#)

# GTFIS\_OPTION\_DELETE\_TEMP\_FILE

GTFIS\_OPTION\_DELETE\_TEMP\_FILE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetTempFileNameIS](#)

# GTFIS\_OPTION\_DONT\_CREATE\_DIR

GTFIS\_OPTION\_DONT\_CREATE\_DIR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetTempFileNameIS](#)

# GTFIS\_OPTION\_DONT\_RESOLVE\_TEXTSUBS

GTFIS\_OPTION\_DONT\_RESOLVE\_TEXTSUBS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetTempFileNameIS](#)

## GTFIS\_OPTION\_NONE

GTFIS\_OPTION\_NONE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetTempFileNameIS](#)

## HELP



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

HELP is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Do](#)
- [HandlerEx](#)

## HIDE\_DISABLED\_BTNS

HIDE\_DISABLED\_BTNS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

The HIDE\_DISABLED\_BTNS constant indicates that the per-user and all-users options should be hidden instead of being disabled. The default value of this variable is TRUE. Note that when this variable is set to TRUE, both options are hidden if either option is determined to be disabled.

### Used With

- [SdCustomerInformation](#)
- [SdCustomerInformationEx](#)

## HKEY\_CLASSES\_ROOT

HKEY\_CLASSES\_ROOT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBSetDefaultRoot](#)

- [RegDBSetKeyValueEx](#)
- [RegDBDeleteKey](#)
- [RegDBDeleteValue](#)
- [RegDBGetKeyValueEx](#)
- [RegDBKeyExist](#)
- [RegDBCreateKeyEx](#)

## HKEY\_CURRENT\_USER

HKEY\_CURRENT\_USER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBSetDefaultRoot](#)

## HKEY\_LOCAL\_MACHINE

HKEY\_LOCAL\_MACHINE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.




---

**Note** • Windows NT 4.0 does not allow the creation of a key directly under HKEY\_LOCAL\_MACHINE.

### Used With

- [RegDBConnectRegistry](#)
- [InstallationInfo](#)
- [RegDBSetDefaultRoot](#)

## HKEY\_USERS

HKEY\_USERS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.




---

**Note** • Windows NT 4.0 does not allow the creation of a key directly under HKEY\_USERS.

### Used With

- [RegDBSetDefaultRoot](#)
- [RegDBConnectRegistry](#)

# HKEY\_USER\_SELECTABLE



**Project** • This information applies to InstallScript projects.

HKEY\_USER\_SELECTABLE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [RegDBSetDefaultRoot](#)

# HOURLASS

HOURLASS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [Disable](#)
- [Enable](#)

# HWND\_DESKTOP

HWND\_DESKTOP is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetWindowHandle](#)

# HWND\_INSTALL

HWND\_INSTALL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetWindowHandle](#)

# IDCANCEL



**Project** • This information applies to InstallScript projects.

IDCANCEL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SelectDir](#)
- [SelectDirEx](#)

## IDOK



**Project** • This information applies to InstallScript projects.

IDOK is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SelectDir](#)
- [SelectDirEx](#)

## IDS\_IFX\_ERROR\_INVALID\_MEDIA\_PASSWORD



**Project** • This information applies to InstallScript projects.

IDS\_IFX\_ERROR\_INVALID\_MEDIA\_PASSWORD is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SdLoadString](#)

## IFX\_ONNEXTDISK\_PACKAGE\_CAPTION



**Project** • This information applies to InstallScript projects.

IFX\_ONNEXTDISK\_PACKAGE\_CAPTION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SdLoadString](#)

# IFX\_ONNEXTDISK\_PACKAGE\_MSG



**Project** • This information applies to InstallScript projects.

IFX\_ONNEXTDISK\_PACKAGE\_MSG is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

## Used With

- [SdLoadString](#)

# INCLUDE\_SUBDIR

INCLUDE\_SUBDIR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FindAllDirs](#)
- [XCopyFile](#)

# INDVFILESTATUS

INDVFILESTATUS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [Disable](#)
- [SetStatusWindow](#)
- [Enable](#)

# INFORMATION

INFORMATION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [MessageBox](#)
- [SprintfBox](#)



## IS\_PERMISSIONS\_OPTION\_64BIT\_OBJECT

IS\_PERMISSIONS\_OPTION\_64BIT\_OBJECT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## IS\_PERMISSIONS\_OPTION\_ALLOW\_ACCESS

IS\_PERMISSIONS\_OPTION\_ALLOW\_ACCESS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## IS\_PERMISSIONS\_OPTION\_DENY\_ACCESS

IS\_PERMISSIONS\_OPTION\_DENY\_ACCESS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## IS\_PERMISSIONS\_OPTION\_NO\_APPLYDOWN

IS\_PERMISSIONS\_OPTION\_NO\_APPLYDOWN is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## IS\_PERMISSIONS\_TYPE\_FILE

IS\_PERMISSIONS\_TYPE\_FILE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## IS\_PERMISSIONS\_TYPE\_FOLDER

IS\_PERMISSIONS\_TYPE\_FOLDER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## IS\_PERMISSIONS\_TYPE\_REGISTRY

IS\_PERMISSIONS\_TYPE\_REGISTRY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## ISDIFX\_OPTION\_DONT\_ASSOCIATE

ISDIFX\_OPTION\_DONT\_ASSOCIATE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DIFxDriverPackageInstall](#)
- [DIFxDriverPackageUninstall](#)

## ISDIFX\_OPTION\_DONT\_RESOLVE\_TEXTSUBS

ISDIFX\_OPTION\_DONT\_RESOLVE\_TEXTSUBS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DIFxDriverPackageGetPath](#)
- [DIFxDriverPackageInstall](#)
- [DIFxDriverPackagePreinstall](#)
- [DIFxDriverPackageUninstall](#)

## ISDIFX\_OPTION\_LOG\_IN\_DRIVER\_PACKAGE\_PATH

ISDIFX\_OPTION\_LOG\_IN\_DRIVER\_PACKAGE\_PATH is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DIFxDriverPackageInstall](#)
- [DIFxDriverPackagePreinstall](#)

## ISDIFX\_OPTION\_NO\_REPAIR

ISDIFX\_OPTION\_NO\_REPAIR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DIFxDriverPackageInstall](#)
- [DIFxDriverPackagePreinstall](#)

## ISERR\_GEN\_FAILURE



---

**Project** • This information applies to *InstallScript* projects.

ISERR\_GEN\_FAILURE is a predefined constant that is used to represent the value that is returned by built-in functions when they fail and more specific information on the cause of the failure is not available. You cannot change the value of a predefined constant.

## ISERR\_SUCCESS



---

**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

ISERR\_SUCCESS is a predefined constant that is used to represent the value that is returned by built-in functions when they are successful. You cannot change the value of a predefined constant.

## ISLANG\_AFRIKAANS

ISLANG\_AFRIKAANS is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_AFRIKAANS\_STANDARD

ISLANG\_AFRIKAANS\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ALBANIAN

ISLANG\_ALBANIAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ALBANIAN\_STANDARD

ISLANG\_ALBANIAN\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ALL

ISLANG\_ALL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFilterLanguage](#)

## ISLANG\_ARABIC

ISLANG\_ARABIC is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ARABIC\_ALGERIA

ISLANG\_ARABIC\_ALGERIA is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ARABIC\_BAHRAIN

ISLANG\_ARABIC\_BAHRAIN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ARABIC\_EGYPT

ISLANG\_ARABIC\_EGYPT is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ARABIC\_IRAQ

ISLANG\_ARABIC\_IRAQ is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ARABIC\_JORDAN

ISLANG\_ARABIC\_JORDAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ARABIC\_KUWAIT

ISLANG\_ARABIC\_KUWAIT is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ARABIC\_LEBANON

ISLANG\_ARABIC\_LEBANON is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ARABIC\_LIBYA

ISLANG\_ARABIC\_LIBYA is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ARABIC\_MOROCCO

ISLANG\_ARABIC\_MOROCCO is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ARABIC\_OMAN

ISLANG\_ARABIC\_OMAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ARABIC\_QATAR

ISLANG\_ARABIC\_QATAR is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ARABIC\_SAUDIARABIA

ISLANG\_ARABIC\_SAUDIARABIA is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ARABIC\_SYRIA

ISLANG\_ARABIC\_SYRIA is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ARABIC\_TUNISIA

ISLANG\_ARABIC\_TUNISIA is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ARABIC\_UAE

ISLANG\_ARABIC\_UAE is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ARABIC\_YEMEN

ISLANG\_ARABIC\_YEMEN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_BASQUE

ISLANG\_BASQUE is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_BASQUE\_STANDARD

ISLANG\_BASQUE\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_BELARUSIAN

ISLANG\_BELARUSIAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_BELARUSIAN\_STANDARD

ISLANG\_BELARUSIAN\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_BULGARIAN

ISLANG\_BULGARIAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_BULGARIAN\_STANDARD

ISLANG\_BULGARIAN\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_CATALAN

ISLANG\_CATALAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_CATALAN\_STANDARD

ISLANG\_CATALAN\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_CHINESE

ISLANG\_CHINESE is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_CHINESE\_HONGKONG

ISLANG\_CHINESE\_HONGKONG is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_CHINESE\_PRC

ISLANG\_CHINESE\_PRC is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_CHINESE\_SINGAPORE

ISLANG\_CHINESE\_SINGAPORE is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_CHINESE\_TAIWAN

ISLANG\_CHINESE\_TAIWAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_CROATIAN

ISLANG\_CROATIAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_CROATIAN\_STANDARD

ISLANG\_CROATIAN\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_CZECH

ISLANG\_CZECH is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_CZECH\_STANDARD

ISLANG\_CZECH\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_DANISH

ISLANG\_DANISH is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_DANISH\_STANDARD

ISLANG\_DANISH\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_DUTCH

ISLANG\_DUTCH is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).



## ISLANG\_DUTCH\_BELGIAN

ISLANG\_DUTCH\_BELGIAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_DUTCH\_STANDARD

ISLANG\_DUTCH\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ENGLISH

ISLANG\_ENGLISH is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ENGLISH\_AUSTRALIAN

ISLANG\_ENGLISH\_AUSTRALIAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ENGLISH\_BELIZE

ISLANG\_ENGLISH\_BELIZE is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ENGLISH\_CANADIAN

ISLANG\_ENGLISH\_CANADIAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ENGLISH\_CARIBBEAN

ISLANG\_ENGLISH\_CARIBBEAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ENGLISH\_IRELAND

ISLANG\_ENGLISH\_IRELAND is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ENGLISH\_JAMAICA

ISLANG\_ENGLISH\_JAMAICA is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ENGLISH\_NEWZEALAND

ISLANG\_ENGLISH\_NEWZEALAND is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ENGLISH\_SOUTHAFRICA

ISLANG\_ENGLISH\_SOUTHAFRICA is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ENGLISH\_TRINIDAD

ISLANG\_ENGLISH\_TRINIDAD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ENGLISH\_UNITEDKINGDOM

ISLANG\_ENGLISH\_UNITEDKINGDOM is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ENGLISH\_UNITEDSTATES

ISLANG\_ENGLISH\_UNITEDSTATES is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ESTONIAN

ISLANG\_ESTONIAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ESTONIAN\_STANDARD

ISLANG\_ESTONIAN\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_FAEROESE

ISLANG\_FAEROESE is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_FAEROESE\_STANDARD

ISLANG\_FAEROESE\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_FARSI

ISLANG\_FARSI is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_FARSI\_STANDARD

ISLANG\_FARSI\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_FINNISH

ISLANG\_FINNISH is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_FINNISH\_STANDARD

ISLANG\_FINNISH\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_FRENCH

ISLANG\_FRENCH is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_FRENCH\_BELGIAN

ISLANG\_FRENCH\_BELGIAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_FRENCH\_CANADIAN

ISLANG\_FRENCH\_CANADIAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_FRENCH\_LUXEMBOURG

ISLANG\_FRENCH\_LUXEMBOURG is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_FRENCH\_STANDARD

ISLANG\_FRENCH\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_FRENCH\_SWISS

ISLANG\_FRENCH\_SWISS is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_GERMAN

ISLANG\_GERMAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_GERMAN\_AUSTRIAN

ISLANG\_GERMAN\_AUSTRIAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_GERMAN\_LIECHTENSTEIN

ISLANG\_GERMAN\_LIECHTENSTEIN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_GERMAN\_LUXEMBOURG

ISLANG\_GERMAN\_LUXEMBOURG is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_GERMAN\_STANDARD

ISLANG\_GERMAN\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_GERMAN\_SWISS

ISLANG\_GERMAN\_SWISS is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_GREEK

ISLANG\_GREEK is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_GREEK\_STANDARD

ISLANG\_GREEK\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_HEBREW

ISLANG\_HEBREW is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_HEBREW\_STANDARD

ISLANG\_HEBREW\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_HUNGARIAN

ISLANG\_HUNGARIAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_HUNGARIAN\_STANDARD

ISLANG\_HUNGARIAN\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ICELANDIC

ISLANG\_ICELANDIC is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ICELANDIC\_STANDARD

ISLANG\_ICELANDIC\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_INDONESIAN

ISLANG\_INDONESIAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_INDONESIAN\_STANDARD

ISLANG\_INDONESIAN\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ITALIAN

ISLANG\_ITALIAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ITALIAN\_STANDARD

ISLANG\_ITALIAN\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ITALIAN\_SWISS

ISLANG\_ITALIAN\_SWISS is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_JAPANESE

ISLANG\_JAPANESE is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_JAPANESE\_STANDARD

ISLANG\_JAPANESE\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_KOREAN

ISLANG\_KOREAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_KOREAN\_JOHAB

ISLANG\_KOREAN\_JOHAB is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_KOREAN\_STANDARD

ISLANG\_KOREAN\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_LATVIAN

ISLANG\_LATVIAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_LATVIAN\_STANDARD

ISLANG\_LATVIAN\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_LITHUANIAN

ISLANG\_LITHUANIAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_LITHUANIAN\_STANDARD

ISLANG\_LITHUANIAN\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_NORWEGIAN

ISLANG\_NORWEGIAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_NORWEGIAN\_BOKMAL

ISLANG\_NORWEGIAN\_BOKMAL is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_NORWEGIAN\_NYNORSK

ISLANG\_NORWEGIAN\_NYNORSK is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_POLISH

ISLANG\_POLISH is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_POLISH\_STANDARD

ISLANG\_POLISH\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_PORTUGUESE

ISLANG\_PORTUGUESE is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_PORTUGUESE\_BRAZILIAN

ISLANG\_PORTUGUESE\_BRAZILIAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_PORTUGUESE\_STANDARD

ISLANG\_PORTUGUESE\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).



## ISLANG\_ROMANIAN

ISLANG\_ROMANIAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_ROMANIAN\_STANDARD

ISLANG\_ROMANIAN\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_RUSSIAN

ISLANG\_RUSSIAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_RUSSIAN\_STANDARD

ISLANG\_RUSSIAN\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SERBIAN\_CYRILLIC

ISLANG\_SERBIAN\_CYRILLIC is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SERBIAN\_LATIN

ISLANG\_SERBIAN\_LATIN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SLOVAK

ISLANG\_SLOVAK is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SLOVAK\_STANDARD

ISLANG\_SLOVAK\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SLOVENIAN

ISLANG\_SLOVENIAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SLOVENIAN\_STANDARD

ISLANG\_SLOVENIAN\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SPANISH

ISLANG\_SPANISH is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SPANISH\_ARGENTINA

ISLANG\_SPANISH\_ARGENTINA is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SPANISH\_BOLIVIA

ISLANG\_SPANISH\_BOLIVIA is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SPANISH\_CHILE

ISLANG\_SPANISH\_CHILE is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SPANISH\_COLOMBIA

ISLANG\_SPANISH\_COLOMBIA is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SPANISH\_COSTARICA

ISLANG\_SPANISH\_COSTARICA is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SPANISH\_DOMINICANREPUBLIC

ISLANG\_SPANISH\_DOMINICANREPUBLIC is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SPANISH\_ECUADOR

ISLANG\_SPANISH\_ECUADOR is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SPANISH\_ELSALVADOR

ISLANG\_SPANISH\_ELSALVADOR is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SPANISH\_GUATEMALA

ISLANG\_SPANISH\_GUATEMALA is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SPANISH\_HONDURAS

ISLANG\_SPANISH\_HONDURAS is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SPANISH\_MEXICAN

ISLANG\_SPANISH\_MEXICAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SPANISH\_MODERNSORT

ISLANG\_SPANISH\_MODERNSORT is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SPANISH\_NICARAGUA

ISLANG\_SPANISH\_NICARAGUA is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SPANISH\_PANAMA

ISLANG\_SPANISH\_PANAMA is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SPANISH\_PARAGUAY

ISLANG\_SPANISH\_PARAGUAY is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SPANISH\_PERU

ISLANG\_SPANISH\_PERU is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SPANISH\_PUERTORICO

ISLANG\_SPANISH\_PUERTORICO is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SPANISH\_TRADITIONALSORT

ISLANG\_SPANISH\_TRADITIONALSORT is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SPANISH\_URUGUAY

ISLANG\_SPANISH\_URUGUAY is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SPANISH\_VENEZUELA

ISLANG\_SPANISH\_VENEZUELA is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SWEDISH

ISLANG\_SWEDISH is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SWEDISH\_FINLAND

ISLANG\_SWEDISH\_FINLAND is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_SWEDISH\_STANDARD

ISLANG\_SWEDISH\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_THAI

ISLANG\_THAI is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_THAI\_STANDARD

ISLANG\_THAI\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_TURKISH

ISLANG\_TURKISH is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_TURKISH\_STANDARD

ISLANG\_TURKISH\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_UKRAINIAN

ISLANG\_UKRAINIAN is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_UKRAINIAN\_STANDARD

ISLANG\_UKRAINIAN\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_VIETNAMESE

ISLANG\_VIETNAMESE is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISLANG\_VIETNAMESE\_STANDARD

ISLANG\_VIETNAMESE\_STANDARD is a predefined constant that corresponds to a Windows language ID. For more information about how to use this constant, see [Language IDs](#).

## ISOSL\_ALL

ISOSL\_ALL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFilterOS](#)

## ISOSL\_SUPPORTED

ISOSL\_SUPPORTED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFilterOS](#)

## ISOSL\_WIN7\_SERVER2008R2

ISOSL\_WIN7\_SERVER2008R2 is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFilterOS](#)

## ISOSL\_WIN8

ISOSL\_WIN8 is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFilterOS](#)

## ISOSL\_WIN81

ISOSL\_WIN81 is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFilterOS](#)

## ISOSL\_WIN10

ISOSL\_WIN10 is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFilterOS](#)

## ISOSL\_WINSERVER2003

ISOSL\_WINSERVER2003 is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFilterOS](#)

## ISOSL\_WINVISTA



---

**Note** • [ISOSL\\_WINVISTA\\_SERVER2008](#) *supersedes* [ISOSL\\_WINVISTA](#).

ISOSL\_WINVISTA is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFilterOS](#)

## ISOSL\_WINVISTA\_SERVER2008

ISOSL\_WINVISTA\_SERVER2008 is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFilterOS](#)

## ISOSL\_WINXP

ISOSL\_WINXP is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFilterOS](#)

## ISOS\_ST\_ALL



---

**Project** • This information applies to InstallScript projects.

ISOS\_ST\_ALL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFilterOS](#)

## ISOS\_ST\_BACKOFFICE



---

**Project** • This information applies to InstallScript projects.

ISOS\_ST\_BACKOFFICE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFilterOS](#)

## ISOS\_ST\_DATACENTER



---

**Project** • This information applies to InstallScript projects.

ISOS\_ST\_DATACENTER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



### Used With

- [FeatureFilterOS](#)

## ISOS\_ST\_ENTERPRISE



**Project** • This information applies to InstallScript projects.

ISOS\_ST\_ENTERPRISE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFilterOS](#)

## ISOS\_ST\_PROC\_ARCH\_32



**Project** • This information applies to InstallScript projects.

ISOS\_ST\_PROC\_ARCH\_32 is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFilterOS](#)

## ISOS\_ST\_PROC\_ARCH\_AMD64



**Project** • This information applies to InstallScript projects.

ISOS\_ST\_PROC\_ARCH\_AMD64 is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFilterOS](#)

## ISOS\_ST\_PROC\_ARCH\_IA64



**Project** • This information applies to InstallScript projects.

ISOS\_ST\_PROC\_ARCH\_IA64 is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFilterOS](#)

## ISOS\_ST\_SERVER



---

**Project** • This information applies to InstallScript projects.

ISOS\_ST\_SERVER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFilterOS](#)

## ISOS\_ST\_SERVER2003\_R2



---

**Project** • This information applies to InstallScript projects.

ISOS\_ST\_SERVER2003\_R2 is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFilterOS](#)

## ISOS\_ST\_SMALLBUSINESS



---

**Project** • This information applies to InstallScript projects.

ISOS\_ST\_SMALLBUSINESS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFilterOS](#)

# ISOS\_ST\_SMALLBUSINESS\_RESTRICTED



**Project** • This information applies to InstallScript projects.

ISOS\_ST\_SMALLBUSINESS\_RESTRICTED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FeatureFilterOS](#)

# ISOS\_ST\_TERMINAL



**Project** • This information applies to InstallScript projects.

ISOS\_ST\_TERMINAL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FeatureFilterOS](#)

# ISOS\_ST\_WORKSTATION



**Project** • This information applies to InstallScript projects.

ISOS\_ST\_WORKSTATION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FeatureFilterOS](#)

# ISOS\_ST\_XP\_HOME



**Project** • This information applies to InstallScript projects.

ISOS\_ST\_XP\_HOME is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FeatureFilterOS](#)

# ISOS\_ST\_XP\_PRO



**Project** • This information applies to InstallScript projects.

ISOS\_ST\_XP\_PRO is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FeatureFilterOS](#)

# ISUS\_AGENT\_FEATURE



**Project** • This information applies to InstallScript projects.

This constant is obsolete. For information on adding FlexNet Connect support to an InstallScript project, consult the [Knowledge Base](#).

# ISUS\_MAIN\_FEATURE



**Project** • This information applies to InstallScript projects.

This constant is obsolete. For information on adding FlexNet Connect support to an InstallScript project, consult the [Knowledge Base](#).

# ISUS\_TEXTSUB\_HOST



**Project** • This information applies to InstallScript projects.

This constant is obsolete. For information on adding FlexNet Connect support to an InstallScript project, consult the [Knowledge Base](#).

# ISUS\_TEXTSUB\_INTERVAL



**Project** • This information applies to InstallScript projects.

This constant is obsolete. For information on adding FlexNet Connect support to an InstallScript project, consult the [Knowledge Base](#).

## ISUS\_TEXTSUB\_LANGUAGE



---

**Project** • This information applies to InstallScript projects.

This constant is obsolete. For information on adding FlexNet Connect support to an InstallScript project, consult the [Knowledge Base](#).

## ISUS\_TEXTSUB\_LOGO



---

**Project** • This information applies to InstallScript projects.

This constant is obsolete. For information on adding FlexNet Connect support to an InstallScript project, consult the [Knowledge Base](#).

## ISUS\_TEXTSUB\_MANAGER



---

**Project** • This information applies to InstallScript projects.

This constant is obsolete. For information on adding FlexNet Connect support to an InstallScript project, consult the [Knowledge Base](#).

## ISUS\_TEXTSUB\_VERSION



---

**Project** • This information applies to InstallScript projects.

This constant is obsolete. For information on adding FlexNet Connect support to an InstallScript project, consult the [Knowledge Base](#).

## ISUS\_UPDATEMANAGER\_FEATURE



---

**Project** • This information applies to InstallScript projects.

This constant is obsolete. For information on adding FlexNet Connect support to an InstallScript project, consult the [Knowledge Base](#).

## IS\_386

IS\_386 is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## IS\_486

IS\_486 is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## IS\_ALPHA

IS\_ALPHA is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## IS\_CDROM

IS\_CDROM is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## IS\_EGA

IS\_EGA is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## IS\_FIXED

IS\_FIXED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## IS\_FOLDER

IS\_FOLDER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [QueryProgItem](#)
- [ReplaceFolderIcon](#)

## IS\_ITEM

IS\_ITEM is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [QueryProgItem](#)
- [ReplaceFolderIcon](#)

## IS\_PENTIUM

IS\_PENTIUM is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## IS\_REMOTE

IS\_REMOTE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## IS\_REMOVABLE

IS\_REMOVABLE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## IS\_SVGA

IS\_SVGA is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## IS\_UNKNOWN

IS\_UNKNOWN is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## IS\_UVGA

IS\_UVGA is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## IS\_VGA

IS\_VGA is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)



## IS\_WINDOWS

IS\_WINDOWS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## IS\_WINDOWS9X

IS\_WINDOWS9X is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## IS\_WINDOWSNT

IS\_WINDOWSNT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## IS\_XVGA

IS\_XVGA is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## KBYTES

KBYTES is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ConvertSizeToUnits](#)
- [StrConvertSizeUnit](#)

## KEY\_CREATE\_LINK

KEY\_CREATE\_LINK is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## KEY\_CREATE\_SUB\_KEY

KEY\_CREATE\_SUB\_KEY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## KEY\_ENUMERATE\_SUB\_KEYS

KEY\_ENUMERATE\_SUB\_KEYS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## KEY\_NOTIFY

GENERIC\_WRITE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## KEY\_QUERY\_VALUE

KEY\_QUERY\_VALUE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## KEY\_SET\_VALUE

KEY\_SET\_VALUE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## LAAW\_OPTION\_CHANGEDIRECTORY

LAAW\_OPTION\_CHANGEDIRECTORY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [LaunchApplication](#)

## LAAW\_OPTION\_FIXUP\_PROGRAM

LAAW\_OPTION\_CHANGEDIRECTORY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [LaunchApplication](#)

## LAAW\_OPTION\_HIDDEN

LAAW\_OPTION\_HIDDEN is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DoInstall](#)
- [LaunchAppAndWait](#)
- [LaunchApplication](#)
- [WaitForApplication](#)

## LAAW\_OPTION\_MAXIMIZED

LAAW\_OPTION\_MAXIMIZED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DoInstall](#)
- [LaunchAppAndWait](#)
- [LaunchApplication](#)
- [WaitForApplication](#)

## LAAW\_OPTION\_MINIMIZED

LAAW\_OPTION\_MINIMIZED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DoInstall](#)
- [LaunchAppAndWait](#)
- [LaunchApplication](#)
- [WaitForApplication](#)

## LAAW\_OPTION\_NO\_CHANGEDIRECTORY

LAAW\_OPTION\_NO\_CHANGEDIRECTORY is an obsolete predefined constant.

## LAAW\_OPTION\_NOWAIT

LAAW\_OPTION\_NOWAIT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DoInstall](#)
- [LaunchAppAndWait](#)
- [LaunchApplication](#)
- [WaitForApplication](#)

## LAAW\_OPTION\_SET\_BATCH\_INSTALL



---

**Project** • This constant is available for InstallScript event-driven code in InstallScript and InstallScript MSI projects. It does not have any effect in InstallScript custom actions in Basic MSI, InstallScript MSI, or Suite/Advanced UI projects.

LAAW\_OPTION\_SET\_BATCH\_INSTALL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DoInstall](#)
- [LaunchAppAndWait](#)
- [LaunchApplication](#)
- [WaitForApplication](#)

## LAAW\_OPTION\_SHOW\_HOURLASS

LAAW\_OPTION\_SHOW\_HOURLASS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DoInstall](#)
- [LaunchAppAndWait](#)

## LAAW\_OPTION\_USE\_CALLBACK

LAAW\_OPTION\_USE\_CALLBACK is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DoInstall](#)
- [LaunchAppAndWait](#)
- [LaunchApplication](#)
- [WaitForApplication](#)

## LAAW\_OPTION\_USE\_SHELLEXECUTE

LAAW\_OPTION\_USE\_SHELLEXECUTE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [LaunchApplication](#)

## LAAW\_OPTION\_WAIT

LAAW\_OPTION\_WAIT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DoInstall](#)
- [LaunchAppAndWait](#)
- [LaunchApplication](#)
- [WaitForApplication](#)

## LAAW\_OPTION\_WAIT\_INCL\_CHILD

LAAW\_OPTION\_WAIT\_INCL\_CHILD is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DoInstall](#)
- [LaunchAppAndWait](#)
- [LaunchApplication](#)
- [WaitForApplication](#)

## LANGUAGE\_SUPPORTED

LANGUAGE\_SUPPORTED is a predefined constant that is used to specify the language that an installation supports. The language is a four-digit hexadecimal language code, including the 0x prefix. For example, for English, the value should be 0x0409. To build a string with STANDARD\_SELECTED\_LANGUAGE in this format, use a statement such as:

```
Sprintf (szLang, "0x%.04lx", STANDARD_SELECTED_LANGUAGE);
```

You cannot change the value of a predefined constant.

### Used With

- [Is](#)

## LANGUAGE

LANGUAGE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## LESS\_THAN

LESS\_THAN is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FileCompare](#)
- [VerCompare](#)

## LINE\_NUMBER

LINE\_NUMBER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FileDeleteLine](#)
- [FileInsertLine](#)

## LISTBOX\_ENTER

LISTBOX\_ENTER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [CtrlGetSubCommand](#)

## LISTBOX\_SELECT

LISTBOX\_SELECT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [CtrlGetSubCommand](#)

## LISTFIRST

LISTFIRST is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ListSetIndex](#)

## LISTLAST

LISTLAST is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ListSetIndex](#)

## LISTNEXT

LISTNEXT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ListSetIndex](#)

## LISTPREV

LISTPREV is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ListSetIndex](#)

## LIST\_NULL

LIST\_NULL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ListCreate](#)

## LOCKEDFILE

LOCKEDFILE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [XCopyFile](#)
- [InstallationInfo](#)
- [VerUpdateFile](#)



- [DeinstallStart](#)

## LOGGING

LOGGING is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DeinstallStart](#)
- [Disable](#)
- [Enable](#)
- [InstallationInfo](#)

## LOWER\_LEFT

LOWER\_LEFT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [PlaceBitmap](#)
- [PlaceWindow](#)

## LOWER\_RIGHT

LOWER\_RIGHT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [PlaceBitmap](#)
- [PlaceWindow](#)

## LWTF\_OPTION\_APPEND\_TO\_FILE

LWTF\_OPTION\_APPEND\_TO\_FILE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ListWriteToFileEx](#)

## LWTF\_OPTION\_WRITE\_AS\_ANSI

LWTF\_OPTION\_WRITE\_AS\_ANSI is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



**Note** • In earlier versions of InstallShield, this constant was called `LWFT_OPTION_WRITE_AS_ANSI` (where **LWFT** was used instead of **LWTF**). To maintain backwards compatibility, both constants are now available, and they are defined the same way.

### Used With

- [ListWriteToFileEx](#)

## LWTF\_OPTION\_WRITE\_AS\_UNICODE

LWTF\_OPTION\_WRITE\_AS\_UNICODE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



**Note** • In earlier versions of InstallShield, this constant was called `LWFT_OPTION_WRITE_AS_UNICODE` (where **LWFT** was used instead of **LWTF**). To maintain backwards compatibility, both constants are now available, and they are defined the same way.

### Used With

- [ListWriteToFileEx](#)

## MAGENTA

MAGENTA is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetColor](#)
- [SetTitle](#)

## MATH\_COPROCESSOR

MATH\_COPROCESSOR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Is](#)

# MBYTES

MBYTES is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [ConvertSizeToUnits](#)
- [StrConvertSizeUnit](#)

# MEDIA\_FIELD\_ADDREMOVE\_NOMODIFY



**Project** • This information applies to InstallScript projects.

MEDIA\_FIELD\_ADDREMOVE\_NOMODIFY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [MediaGetData](#)
- [MediaGetDataEx](#)

# MEDIA\_FIELD\_ADDREMOVE\_NOREMOVE



**Project** • This information applies to InstallScript projects.

MEDIA\_FIELD\_ADDREMOVE\_NOREMOVE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [MediaGetData](#)
- [MediaGetDataEx](#)

# MEDIA\_FIELD\_COMPANY\_NAME



**Project** • This information applies to InstallScript projects.

MEDIA\_FIELD\_COMPANY\_NAME is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [MediaGetData](#)

## MEDIA\_FIELD\_MEDIA\_FLAGS



---

**Project** • This information applies to InstallScript projects.

MEDIA\_FIELD\_MEDIA\_FLAGS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [MediaGetData](#)

## MEDIA\_FIELD\_PREVIOUS\_VERSIONS



---

**Project** • This information applies to InstallScript projects.

MEDIA\_FIELD\_PREVIOUS\_VERSIONS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [MediaGetData](#)

## MEDIA\_FIELD\_PRODUCT\_COMMENTS



---

**Project** • This information applies to InstallScript projects.

MEDIA\_FIELD\_PRODUCT\_COMMENTS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [MediaGetData](#)
- [MediaGetDataEx](#)

## MEDIA\_FIELD\_PRODUCT\_EXE



---

**Project** • This information applies to InstallScript projects.

MEDIA\_FIELD\_PRODUCT\_EXE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [MediaGetData](#)

## MEDIA\_FIELD\_PRODUCT\_ICON



---

**Project** • This information applies to InstallScript projects.

MEDIA\_FIELD\_PRODUCT\_ICON is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [MediaGetData](#)
- [MediaGetDataEx](#)

## MEDIA\_FIELD\_PRODUCT\_NAME



---

**Project** • This information applies to InstallScript projects.

MEDIA\_FIELD\_PRODUCT\_NAME is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [MediaGetData](#)

## MEDIA\_FIELD\_PRODUCT\_README



---

**Project** • This information applies to InstallScript projects.

MEDIA\_FIELD\_PRODUCT\_README is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [MediaGetData](#)
- [MediaGetDataEx](#)

# MEDIA\_FIELD\_PRODUCT\_SUPPORT\_CONTACT



---

**Project** • This information applies to InstallScript projects.

MEDIA\_FIELD\_PRODUCT\_SUPPORT\_CONTACT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [MediaGetData](#)
- [MediaGetDataEx](#)

# MEDIA\_FIELD\_PRODUCT\_SUPPORT\_PHONE



---

**Project** • This information applies to InstallScript projects.

MEDIA\_FIELD\_PRODUCT\_SUPPORT\_PHONE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [MediaGetData](#)
- [MediaGetDataEx](#)

# MEDIA\_FIELD\_PRODUCT\_SUPPORT\_URL



---

**Project** • This information applies to InstallScript projects.

MEDIA\_FIELD\_PRODUCT\_SUPPORT\_URL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [MediaGetData](#)
- [MediaGetDataEx](#)

# MEDIA\_FIELD\_PRODUCT\_UPDATE\_URL



---

**Project** • This information applies to InstallScript projects.

MEDIA\_FIELD\_PRODUCT\_UPDATE\_URL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [MediaGetData](#)
- [MediaGetDataEx](#)

## MEDIA\_FIELD\_PRODUCT\_URL



---

**Project** • This information applies to InstallScript projects.

MEDIA\_FIELD\_PRODUCT\_URL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [MediaGetData](#)
- [MediaGetDataEx](#)

## MEDIA\_FIELD\_PRODUCT\_VERSION



---

**Project** • This information applies to InstallScript projects.

MEDIA\_FIELD\_PRODUCT\_VERSION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [MediaGetData](#)

## MEDIA\_FIELD\_TARGETDIR



---

**Project** • This information applies to InstallScript projects.

MEDIA\_FIELD\_TARGETDIR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [MediaGetData](#)

# MEDIA\_FLAG\_FORMAT\_DIFFERENTIAL



**Project** • This information applies to InstallScript projects.

MEDIA\_FLAG\_FORMAT\_DIFFERENTIAL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [MediaGetData](#)

# MEDIA\_FLAG\_FORMAT\_PATCH



**Project** • This information applies to InstallScript projects.

MEDIA\_FLAG\_FORMAT\_PATCH is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [MediaGetData](#)

# MEDIA\_FLAG\_UPDATEMODE\_SUPPORTED



**Project** • This information applies to InstallScript projects.

MEDIA\_FLAG\_UPDATEMODE\_SUPPORTED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

The MEDIA\_FLAG\_UPDATEMODE\_SUPPORTED flag is always set.

## Used With

- [MediaGetData](#)
- [MediaGetDataEx](#)

# MEDIA\_PASSWORD\_KEY



**Project** • This information applies to InstallScript projects.



MEDIA\_PASSWORD\_KEY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

#### Used With

- [LogReadCustomString](#)
- [LogWriteCustomString](#)

## METAFILE

METAFILE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

#### Used With

- [SizeWindow](#)

## MMEDIA\_AVI

MMEDIA\_AVI is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

#### Used With

- [PlayMMedia](#)
- [PlaceWindow](#)
- [SizeWindow](#)

## MMEDIA\_MIDI

MMEDIA\_MIDI is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

#### Used With

- [PlayMMedia](#)

## MMEDIA\_PLAYASYNCH

MMEDIA\_PLAYASYNCH is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

#### Used With

- [PlayMMedia](#)

## MMEDIA\_PLAYCONTINUOUS

MMEDIA\_PLAYCONTINUOUS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [PlayMMedia](#)

## MMEDIA\_PLAYSYNCH

MMEDIA\_PLAYSYNCH is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [PlayMMedia](#)

## MMEDIA\_STOP

MMEDIA\_STOP is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [PlayMMedia](#)

## MMEDIA\_SWF

MMEDIA\_SWF is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [PlayMMedia](#)
- [PlaceWindow](#)
- [SizeWindow](#)

## MMEDIA\_WAVE

MMEDIA\_WAVE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [PlayMMedia](#)

# MODIFY



**Project** • This information applies to *InstallScript* projects.

MODIFY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [SdWelcomeMaint](#)

# NEXT

NEXT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [AskDestPath](#)
- [AskOptions](#)
- [AskPath](#)
- [AskText](#)
- [FeatureDialog](#)
- [SdAskDestPath](#)
- [SdAskOptions](#)
- [SdAskOptionsList](#)
- [SdBitmap](#)
- [SdDisplayTopics](#)
- [SdFeatureDialog](#)
- [SdFeatureDialog2](#)
- [SdFeatureDialogAdv](#)
- [SdFeatureMult](#)
- [SdLicense](#)
- [SdOptionsButtons](#)
- [SdRegisterUser](#)
- [SdRegisterUserEx](#)
- [SdSelectFolder](#)
- [SdShowAnyDialog](#)

- [SdShowDlgEdit1](#)
- [SdShowDlgEdit2](#)
- [SdShowDlgEdit3](#)
- [SdShowFileMods](#)
- [SdShowInfoList](#)
- [SdStartCopy](#)
- [SdWelcome](#)
- [SelectFolder](#)
- [Welcome](#)

## NEXTBUTTON

NEXTBUTTON is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Disable](#)
- [Enable](#)
- [Is](#)

## NO

NO is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SdConfirmNewDir](#)
- [SdConfirmRegistration](#)
- [AskYesNo](#)

## NONEXCLUSIVE

NONEXCLUSIVE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SdAskOptionsList](#)
- [AskOptions](#)

- [SdAskOptions](#)

## NORMALMODE

NORMALMODE is a predefined constant that can be used to test whether or not a setup is running in silent mode. For more information, refer to the InstallShield system variable [MODE](#).

## NORMAL\_PRIORITY\_CLASS



---

**Project** • This information applies to InstallScript projects.

NORMAL\_PRIORITY\_CLASS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_PARAMS](#)

## NOSET

NOSET is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [EzBatchAddString](#)

## NOTEXISTS

NOTEXISTS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ExistsDir](#)
- [ExistsDisk](#)

## NO\_SUBDIR



---

**Project** • This information applies to InstallScript projects.

NO\_SUBDIR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureFileEnum](#)

## NULL

NULL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [AddFolderIcon](#)
- [CreateShortcut](#)
- [GetShortcutInfo](#)
- [FindWindow](#)
- [QueryProgItem](#)
- [ReplaceFolderIcon](#)
- [ReplaceShortcut](#)

## NUMBERLIST

NUMBERLIST is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ListCreate](#)

## OFF

OFF is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [StatusUpdate](#)

## OK

OK is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [EnterDisk](#)

## ON

ON is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [StatusUpdate](#)

## ONLYDIR

ONLYDIR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [DeleteDir](#)

## OTHER\_FAILURE

OTHER\_FAILURE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FileDeleteLine](#)
- [VerUpdateFile](#)
- [FileCompare](#)
- [FileGrep](#)
- [FileInsertLine](#)

## OUT\_OF\_DISK\_SPACE

OUT\_OF\_DISK\_SPACE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [VerUpdateFile](#)
- [FileInsertLine](#)
- [FileDeleteLine](#)

- [VerSearchAndUpdateFile](#)

## PARALLEL

PARALLEL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## PARTIAL

PARTIAL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [PathAdd](#)
- [PathFind](#)
- [PathMove](#)
- [PathDelete](#)

## PATH

PATH is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ParsePath](#)

## PATH\_EXISTS

» InstallScript Language Reference

PATH\_EXISTS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Is](#)



# PCRESTORE



**Project** • This information applies to *InstallScript* projects.

PCRESTORE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [Disable](#)
- [Enable](#)

# PERSONAL

PERSONAL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [ProgDefGroupType](#)

# READ\_CONTROL

READ\_CONTROL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

## Used With

- [SERVICE\\_IS\\_PARAMS](#)
- [SetObjectPermissions](#)

# REBOOTED

REBOOTED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [Is](#)

## RECORDMODE

RECORDMODE is a predefined constant that can be used to test whether or not a setup is automatically generating a silent setup file (.iss file), which is a record of the setup input, in the Windows folder. For more information, refer to the InstallScript system variable [MODE](#).

## RED

RED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetColor](#)
- [SetTitle](#)

## REGDBREMOTEREGCONNECTED

REGDBREMOTEREGCONNECTED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Is](#)

## REGDB\_APPPATH

REGDB\_APPPATH is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

## REGDB\_APPPATH\_DEFAULT

REGDB\_APPPATH\_DEFAULT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)

- [RegDBSetItem](#)

## REGDB\_BINARY

REGDB\_BINARY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBSetAppInfo](#)
- [RegDBGetKeyValueEx](#)
- [RegDBSetKeyValueEx](#)
- [RegDBGetAppInfo](#)
- [RegDBSetKeyValueEx](#)

## REGDB\_ERR\_CONNECTIONEXISTS

REGDB\_ERR\_CONNECTIONEXISTS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBConnectRegistry](#)

## REGDB\_ERR\_CORRUPTEDREGISTRY

REGDB\_ERR\_CORRUPTEDREGISTRY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBConnectRegistry](#)

## REGDB\_ERR\_INITIALIZATION

REGDB\_ERR\_INITIALIZATION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBConnectRegistry](#)

## REGDB\_ERR\_INVALIDHANDLE

REGDB\_ERR\_INVALIDHANDLE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBConnectRegistry](#)

## REGDB\_ERR\_INVALIDNAME

REGDB\_ERR\_INVALIDNAME is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBConnectRegistry](#)

## REGDB\_KEYPATH\_APPPATHS

REGDB\_KEYPATH\_APPPATHS is a predefined constant whose value is the registry location (not including the root key) of the general application paths key, that is, **Software\Microsoft\Windows\CurrentVersion\App Paths\**. You can use this constant to specify a key when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_KEYPATH\_DOTNET\_10

REGDB\_KEYPATH\_DOTNET\_10 is a predefined constant whose value is the registry location (not including the root key) of the registry key for version 1.0 of the .NET Framework. It is defined as follows:

**Software\Microsoft\NET Framework Setup\Full\v1.0.3705\1033\Microsoft .NET Framework Full v1.0.3705 (1033)\**

You cannot change the value of a predefined constant. You can use this constant to specify a key when calling a general registry-related function. This predefined constant is also supported when using the **Is** function.

### Used With

- [Is](#)

## REGDB\_KEYPATH\_DOTNET\_11

REGDB\_KEYPATH\_DOTNET\_11 is a predefined constant whose value is the registry location (not including the root key) of the registry key for version 1.1 of the .NET Framework. It is defined as follows:

**Software\Microsoft\NET Framework Setup\NDP\v1.1.4322\**

You cannot change the value of a predefined constant. You can use this constant to specify a key when calling a general registry-related function. This predefined constant is also supported when using the **Is** function.

### Used With

- [Is](#)

## REGDB\_KEYPATH\_DOTNET\_20

REGDB\_KEYPATH\_DOTNET\_20 is a predefined constant whose value is the registry location (not including the root key) of the registry key for version 2.0 of the .NET Framework. It is defined as follows:

**Software\Microsoft\NET Framework Setup\NDP\v2.0.50215\**

You cannot change the value of a predefined constant. You can use this constant to specify a key when calling a general registry-related function. This predefined constant is also supported when using the **Is** function.

### Used With

- [Is](#)

## REGDB\_KEYPATH\_DOTNET\_30

REGDB\_KEYPATH\_DOTNET\_30 is a predefined constant whose value is the registry location (not including the root key) of the registry key for the RTM version of the .NET Framework 3.0. It is defined as follows:

**Software\Microsoft\NET Framework Setup\NDP\v3.0\Setup\**



**Tip** • You can use the `REGDB_KEYPATH_DOTNET_30` variable to detect whether the RTM version of the .NET Framework 3.0 is installed. To detect whether SP1—or a later service pack—of the .NET Framework 3.0 is installed, use [REGDB\\_KEYPATH\\_DOTNET\\_30\\_SP](#).

You cannot change the value of a predefined constant. You can use this constant to specify a key when calling a general registry-related function. This predefined constant is also supported when using the **Is** function.

### Used With

- [Is](#)

## REGDB\_KEYPATH\_DOTNET\_30\_SP

REGDB\_KEYPATH\_DOTNET\_30\_SP is a predefined constant whose value is the registry location (not including the root key) of the registry key for SP1—or a later service pack—of the .NET Framework 3.0. It is defined as follows:

**Software\Microsoft\NET Framework Setup\NDP\v3.0\**



**Tip** • You can use the `REGDB_KEYPATH_DOTNET_30_SP` variable when querying whether SP1—or a later service pack—of the .NET Framework 3.0 is installed. To detect whether the RTM version of the .NET Framework 3.0 is installed, use [REGDB\\_KEYPATH\\_DOTNET\\_30](#).

You cannot change the value of a predefined constant. You can use this constant to specify a key when calling a general registry-related function. This predefined constant is also supported when using the **Is** function.

### Used With

- **Is**

## REGDB\_KEYPATH\_DOTNET\_35

REGDB\_KEYPATH\_DOTNET\_35 is a predefined constant whose value is the registry location (not including the root key) of the registry key for version 3.5 of the .NET Framework. It is defined as follows:

**Software\Microsoft\NET Framework Setup\NDP\v3.5\**

You cannot change the value of a predefined constant. You can use this constant to specify a key when calling a general registry-related function. This predefined constant is also supported when using the **Is** function.

### Used With

- **Is**

## REGDB\_KEYPATH\_DOTNET\_40\_CLIENT

REGDB\_KEYPATH\_DOTNET\_40\_CLIENT is a predefined constant whose value is the registry location (not including the root key) of the registry key for version 4.0 of the .NET Framework Client Profile. It is defined as follows:

**Software\Microsoft\NET Framework Setup\NDP\v4\Client**

You cannot change the value of a predefined constant. You can use this constant to specify a key when calling a general registry-related function. This predefined constant is also supported when using the **Is** function.

### Used With

- **Is**

## REGDB\_KEYPATH\_DOTNET\_40\_FULL

REGDB\_KEYPATH\_DOTNET\_40\_FULL is a predefined constant whose value is the registry location (not including the root key) of the registry key for version 4.0 of the .NET Framework. It is defined as follows:

**Software\Microsoft\NET Framework Setup\NDP\v4\Full**

You cannot change the value of a predefined constant. You can use this constant to specify a key when calling a general registry-related function. This predefined constant is also supported when using the **Is** function.

### Used With

- **Is**

## REGDB\_KEYPATH\_ISUNINSTINFO

ISUNINSTINFO is a predefined constant whose value is the registry location (not including the root key) of the InstallShield Uninstall Information key. It is defined as follows:

**Software\Microsoft\Windows\CurrentVersion\Uninstall\InstallShield Uninstall Information**

You cannot change the value of a predefined constant. You can use this constant to specify a key when calling a general registry-related function. This predefined constant is also supported when using the **Is** function.

### Used With

- **Is**

## REGDB\_KEYPATH\_RUN

REGDB\_KEYPATH\_RUN is a predefined constant whose value is the registry location (not including the root key) of the general application paths key, that is, **Software\Microsoft\Windows\CurrentVersion\Run\**. You can use this constant to specify a key when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_KEYPATH\_RUNONCE

REGDB\_KEYPATH\_RUNONCE is a predefined constant whose value is the registry location (not including the root key) of the general application paths key, that is, **Software\Microsoft\Windows\CurrentVersion\RunOnce\**. You can use this constant to specify a key when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_KEYPATH\_RUNONCEEX

REGDB\_KEYPATH\_RUNONCEEX is a predefined constant whose value is the registry location (not including the root key) of the general application paths key, that is, **Software\Microsoft\Windows\CurrentVersion\RunOnceEx\**. You can use this constant to specify a key when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_KEYPATH\_SHARED DLLS

REGDB\_KEYPATH\_SHARED DLLS is a predefined constant whose value is the registry location (not including the root key) of the general application paths key, that is, **Software\Microsoft\Windows\CurrentVersion\Shared DLLs\**. You can use this constant to specify a key when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_KEYPATH\_UNINSTALL

REGDB\_KEYPATH\_UNINSTALL is a predefined constant whose value is the registry location (not including the root key) of the general uninstallation key for applications, that is, **Software\Microsoft\Windows\CurrentVersion\Uninstall\**. You can use this constant to specify a key when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_KEYPATH\_WINCURRVER

REGDB\_KEYPATH\_WINCURRVER is a predefined constant whose value is the registry location (not including the root key) of the Windows current version key, that is, **Software\Microsoft\Windows\CurrentVersion\**. You can use this constant to specify a key when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_KEYPATH\_WINCURRVER\_AUTO

The value of this system variable is [REGDB\\_KEYPATH\\_WINCURRVER](#) on Windows 95, Windows 95, and Windows Me, and [REGDB\\_KEYPATH\\_WINNTCURRVER](#) on Windows NT, Windows 2000, and Windows XP and later.

## REGDB\_KEYPATH\_WINNTCURRVER

REGDB\_KEYPATH\_WINNTCURRVER is a predefined constant whose value is the registry location (not including the root key) of the Windows NT current version key, that is, **Software\Microsoft\Windows NT\CurrentVersion\**. You can use this constant to specify a key when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_KEYS

REGDB\_KEYS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBQueryKey](#)

## REGDB\_NAMES

REGDB\_NAMES is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBQueryKey](#)



## REGDB\_NUMBER

REGDB\_NUMBER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBGetAppInfo](#)
- [RegDBGetKeyValueEx](#)
- [RegDBSetKeyValueEx](#)
- [RegDBSetAppInfo](#)

## REGDB\_STRING

REGDB\_STRING is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBGetAppInfo](#)
- [RegDBGetKeyValueEx](#)
- [RegDBSetKeyValueEx](#)
- [RegDBSetAppInfo](#)

## REGDB\_STRING\_EXPAND

REGDB\_STRING\_EXPAND is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBSetAppInfo](#)
- [RegDBGetKeyValueEx](#)
- [RegDBSetKeyValueEx](#)
- [RegDBGetAppInfo](#)

## REGDB\_STRING\_MULTI

REGDB\_STRING\_MULTI is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBGetAppInfo](#)
- [RegDBGetKeyValueEx](#)
- [RegDBSetKeyValueEx](#)
- [RegDBSetAppInfo](#)

## REGDB\_UNINSTALL\_COMMENTS

REGDB\_UNINSTALL\_COMMENTS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

## REGDB\_UNINSTALL\_CONTACT

REGDB\_UNINSTALL\_CONTACT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

## REGDB\_UNINSTALL\_DISPLAYICON

REGDB\_UNINSTALL\_DISPLAYICON is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

# REGDB\_UNINSTALL\_DISPLAY\_VERSION



**Project** • This information applies to InstallScript projects.

REGDB\_UNINSTALL\_DISPLAY\_VERSION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

# REGDB\_UNINSTALL\_HELPLINK

REGDB\_UNINSTALL\_HELPLINK is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

# REGDB\_UNINSTALL\_HELPTELEPHONE

REGDB\_UNINSTALL\_HELPTELEPHONE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

# REGDB\_UNINSTALL\_INSTALLDATE

REGDB\_UNINSTALL\_INSTALLDATE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [RegDBDeleteItem](#)

- [RegDBGetItem](#)
- [RegDBSetItem](#)

## REGDB\_UNINSTALL\_INSTALLLOC



**Project** • This information applies to InstallScript projects.

REGDB\_UNINSTALL\_INSTALLLOC is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

## REGDB\_UNINSTALL\_INSTALLSOURCE

REGDB\_UNINSTALL\_INSTALLSOURCE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

## REGDB\_UNINSTALL\_LANGUAGE

REGDB\_UNINSTALL\_LANGUAGE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

# REGDB\_UNINSTALL\_LOGFILE



**Project** • This information applies to InstallScript projects.

REGDB\_UNINSTALL\_LOGFILE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

## Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

# REGDB\_UNINSTALL\_MAINT\_OPTION



**Project** • This information applies to InstallScript projects.

REGDB\_UNINSTALL\_MAINT\_OPTION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

# REGDB\_UNINSTALL\_MAJOR\_VERSION



**Project** • This information applies to InstallScript projects.

REGDB\_UNINSTALL\_MAJOR\_VERSION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

# REGDB\_UNINSTALL\_MAJOR\_VERSION\_OLD



**Project** • This information applies to InstallScript projects.

REGDB\_UNINSTALL\_MAJOR\_VERSION\_OLD is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

# REGDB\_UNINSTALL\_MINOR\_VERSION



**Project** • This information applies to InstallScript projects.

REGDB\_UNINSTALL\_MINOR\_VERSION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

# REGDB\_UNINSTALL\_MINOR\_VERSION\_OLD



**Project** • This information applies to InstallScript projects.

REGDB\_UNINSTALL\_MINOR\_VERSION\_OLD is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

## REGDB\_UNINSTALL\_MODIFYPATH

REGDB\_UNINSTALL\_MODIFYPATH is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

## REGDB\_UNINSTALL\_NAME

REGDB\_UNINSTALL\_NAME is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

## REGDB\_UNINSTALL\_NOMODIFY

REGDB\_UNINSTALL\_NOMODIFY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

## REGDB\_UNINSTALL\_NOREMOVE

REGDB\_UNINSTALL\_NOREMOVE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

## REGDB\_UNINSTALL\_NOREPAIR

REGDB\_UNINSTALL\_NOREPAIR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

## REGDB\_UNINSTALL\_PRODUCTGUID



---

**Project** • This information applies to InstallScript projects.

REGDB\_UNINSTALL\_PRODUCTGUID is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

## REGDB\_UNINSTALL\_PRODUCTID

REGDB\_UNINSTALL\_PRODUCTID is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

## REGDB\_UNINSTALL\_PUBLISHER

REGDB\_UNINSTALL\_PUBLISHER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBDeleteItem](#)



- [RegDBGetItem](#)
- [RegDBSetItem](#)

## REGDB\_UNINSTALL\_README

REGDB\_UNINSTALL\_README is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

## REGDB\_UNINSTALL\_REGCOMPANY

REGDB\_UNINSTALL\_REGCOMPANY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

## REGDB\_UNINSTALL\_REGOWNER

REGDB\_UNINSTALL\_REGOWNER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

## REGDB\_UNINSTALL\_STRING



---

**Project** • This information applies to InstallScript projects.

REGDB\_UNINSTALL\_STRING is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

## REGDB\_UNINSTALL\_SYSTEMCOMPONENT

REGDB\_UNINSTALL\_SYSTEMCOMPONENT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

## REGDB\_UNINSTALL\_URLINFOABOUT

REGDB\_UNINSTALL\_URLINFOABOUT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

## REGDB\_UNINSTALL\_URLUPDATEINFO

REGDB\_UNINSTALL\_URLUPDATEINFO is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

# REGDB\_UNINSTALL\_VERSION



**Project** • This information applies to *InstallScript* projects.

REGDB\_UNINSTALL\_VERSION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [RegDBDeleteItem](#)
- [RegDBGetItem](#)
- [RegDBSetItem](#)

# REGDB\_VALUENAME\_APPPATH

REGDB\_VALUENAME\_APPPATH is a predefined constant whose value is the path value name under the application path key, that is, *Path*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

# REGDB\_VALUENAME\_APPPATHDEFAULT

REGDB\_VALUENAME\_APPPATHDEFAULT is a predefined constant whose value is the default value name under the application path key, that is, a null string (""),. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

# REGDB\_VALUENAME\_INSTALL

REGDB\_VALUENAME\_INSTALL is a predefined constant whose value is *Install*. You can use this constant to specify a value when calling a general registry-related function. You cannot change the value of a predefined constant.

# REGDB\_VALUENAME\_INSTALLSUCCESS

REGDB\_VALUENAME\_INSTALLSUCCESS is a predefined constant whose value is *InstallSuccess*. You can use this constant to specify a value when calling a general registry-related function. You cannot change the value of a predefined constant.

# REGDB\_VALUENAME\_SP

REGDB\_VALUENAME\_INSTALL is a predefined constant whose value is *SP*. You can use this constant to specify a value when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_COMMENTS

REGDB\_VALUENAME\_UNINSTALL\_COMMENTS is a predefined constant whose value is the comments value name under the application uninstallation key—that is, *Comments*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_CONTACT

REGDB\_VALUENAME\_UNINSTALL\_CONTACT is a predefined constant whose value is the contact value name under the application uninstallation key—that is, *Contact*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_DISPLAYICON

REGDB\_VALUENAME\_UNINSTALL\_DISPLAYICON is a predefined constant whose value is the display icon value name under the application uninstallation key—that is, *DisplayIcon*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_DISPLAYNAME

REGDB\_VALUENAME\_UNINSTALL\_DISPLAYNAME is a predefined constant whose value is the display name value name under the application uninstallation key—that is, *DisplayName*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_DISPLAYVERSION

REGDB\_VALUENAME\_UNINSTALL\_DISPLAYVERSION is a predefined constant whose value is the display version value name under the application uninstallation key—that is, *DisplayVersion*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_HELP LINK

REGDB\_VALUENAME\_UNINSTALL\_HELP LINK is a predefined constant whose value is the help link value name under the application uninstallation key—that is, *HelpLink*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_HELP TELEPHONE

REGDB\_VALUENAME\_UNINSTALL\_HELP TELEPHONE is a predefined constant whose value is the help telephone value name under the application uninstallation key—that is, *HelpTelephone*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_INSTALLDATE

REGDB\_VALUENAME\_UNINSTALL\_INSTALLDATE is a predefined constant whose value is the installation date value name under the application uninstallation key—that is, *InstallDate*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_INSTALLLOCATION

REGDB\_VALUENAME\_UNINSTALL\_INSTALLLOCATION is a predefined constant whose value is the installation location value name under the application uninstallation key—that is, *InstallLocation*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_INSTALLSOURCE

REGDB\_VALUENAME\_UNINSTALL\_INSTALLSOURCE is a predefined constant whose value is the installation source value name under the application uninstallation key—that is, *InstallSource*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_LANGUAGE

REGDB\_VALUENAME\_UNINSTALL\_LANGUAGE is a predefined constant whose value is the language value name under the application uninstallation key—that is, *Language*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_LOGFILE

REGDB\_VALUENAME\_UNINSTALL\_LOGFILE is a predefined constant whose value is the log file value name under the application uninstallation key—that is, *LogFile*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_LOGMODE

REGDB\_VALUENAME\_UNINSTALL\_LOGMODE is a predefined constant whose value is the log mode value name under the application uninstallation key—that is, *LogMode*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_MAJORVERSION

REGDB\_VALUENAME\_UNINSTALL\_MAJORVERSION is a predefined constant whose value is the major version value name under the application uninstallation key—that is, *VersionMajor*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_MAJORVERSION\_OLD

REGDB\_VALUENAME\_UNINSTALL\_MAJORVERSION\_OLD is a predefined constant whose value is the major version value name under the application uninstallation key—that is, *MajorVersion*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_MINORVERSION

REGDB\_VALUENAME\_UNINSTALL\_MINORVERSION is a predefined constant whose value is the minor version value name under the application uninstallation key—that is, *VersionMinor*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_MINORVERSION\_OLD

REGDB\_VALUENAME\_UNINSTALL\_MINORVERSION\_OLD is a predefined constant whose value is the minor version value name under the application uninstallation key—that is, *MinorVersion*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_MODIFYPATH

REGDB\_VALUENAME\_UNINSTALL\_MODIFYPATH is a predefined constant whose value is the modify path value name under the application uninstallation key—that is, *ModifyPath*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_NOMODIFY

REGDB\_VALUENAME\_UNINSTALL\_NOMODIFY is a predefined constant whose value is the no modify value name under the application uninstallation key—that is, *NoModify*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_NOREMOVE

REGDB\_VALUENAME\_UNINSTALL\_NOREMOVE is a predefined constant whose value is the no remove value name under the application uninstallation key—that is, *NoRemove*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_NOREPAIR

REGDB\_VALUENAME\_UNINSTALL\_NOREPAIR is a predefined constant whose value is the no repair value name under the application uninstallation key—that is, *NoRepair*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_PRODUCTGUID

REGDB\_VALUENAME\_UNINSTALL\_PRODUCTGUID is a predefined constant whose value is the product GUID value name under the application uninstallation key—that is, *ProductGuid*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_PRODUCTID

REGDB\_VALUENAME\_UNINSTALL\_PRODUCTID is a predefined constant whose value is the product ID value name under the application uninstallation key—that is, *ProductId*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_PUBLISHER

REGDB\_VALUENAME\_UNINSTALL\_PUBLISHER is a predefined constant whose value is the publisher value name under the application uninstallation key—that is, *Publisher*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_README

REGDB\_VALUENAME\_UNINSTALL\_README is a predefined constant whose value is the readme value name under the application uninstallation key—that is, *Readme*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_REGCOMPANY

REGDB\_VALUENAME\_UNINSTALL\_REGCOMPANY is a predefined constant whose value is the reg company value name under the application uninstallation key—that is, *RegCompany*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_REGOWNER

REGDB\_VALUENAME\_UNINSTALL\_REGOWNER is a predefined constant whose value is the reg owner value name under the application uninstallation key—that is, *RegOwner*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_SYSTEMCOMPONENT

REGDB\_VALUENAME\_UNINSTALL\_SYSTEMCOMPONENT is a predefined constant whose value is the system component value name under the application uninstallation key—that is, *SystemComponent*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_UNINSTALLSTRING

REGDB\_VALUENAME\_UNINSTALL\_UNINSTALLSTRING is a predefined constant whose value is the uninstallation string value name under the application uninstallation key—that is, *UninstallString*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_URLINFOABOUT

REGDB\_VALUENAME\_UNINSTALL\_URLINFOABOUT is a predefined constant whose value is the URL info about value name under the application uninstallation key—that is, *URLInfoAbout*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_URLUPDATEINFO

REGDB\_VALUENAME\_UNINSTALL\_URLUPDATEINFO is a predefined constant whose value is the URL date info value name under the application uninstallation key—that is, *URLDateInfo*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALL\_VERSION

REGDB\_VALUENAME\_UNINSTALL\_VERSION is a predefined constant whose value is the application version value name under the application uninstallation key—that is, *Version*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_UNINSTALLKEY

REGDB\_VALUENAME\_UNINSTALLKEY is a predefined constant whose value is defined as *UninstallKey*. You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_VALUENAME\_WINCURREVER\_REGORGANIZATION

REGDB\_VALUENAME\_WINCURREVER\_REGORGANIZATION is a predefined constant whose value is the registered organization value name under the Windows (on Windows 95, Windows 95, and Windows Me) or Windows NT (on Windows 95, Windows 95, and Windows Me) current version key, that is, "RegisteredOrganization". You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.



## REGDB\_VALUENAME\_WINCURRVER\_REGOWNER

REGDB\_VALUENAME\_WINCURRVER\_REGOWNER is a predefined constant whose value is the registered owner value name under the Windows (on Windows 95, Windows 95, and Windows Me) or Windows NT (on Windows 95, Windows 95, and Windows Me) current version key, that is, "RegisteredOwner". You can use this constant to specify a value name when calling a general registry-related function. You cannot change the value of a predefined constant.

## REGDB\_WINCURRVER\_REGORGANIZATION

REGDB\_WINCURRVER\_REGORGANIZATION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBGetItem](#)

## REGDB\_WINCURRVER\_REGOWNER

REGDB\_WINCURRVER\_REGOWNER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegDBGetItem](#)

## REGFONT\_OPTION\_DEFAULT

REGFONT\_OPTION\_DEFAULT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegisterFontResource](#)

## REGFONT\_OPTION\_DONTBROADCASTFONTCHANGE MSG

REGFONT\_OPTION\_DONTBROADCASTFONTCHANGEMSG is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegisterFontResource](#)

## REGFONT\_OPTION\_DONTUPDATEREGISTRY

REGFONT\_OPTION\_DONTUPDATEREGISTRY is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [RegisterFontResource](#)

## REGISTRYFUNCTIONS\_USETEXTSUBS

REGISTRYFUNCTIONS\_USETEXTSUBS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Disable](#)
- [Enable](#)

## REMOTE\_DRIVE

REMOTE\_DRIVE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetValidDrivesList](#)

## REMOVE

REMOVE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [PlaceBitmap](#)

## REMOVEABLE\_DRIVE

REMOVEABLE\_DRIVE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetValidDrivesList](#)

# REMOVEALL



**Project** • This information applies to InstallScript projects.

REMOVEALL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [SdWelcomeMaint](#)

# REPAIR



**Project** • This information applies to InstallScript projects.

REPAIR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [SdWelcomeMaint](#)

# REPLACE

REPLACE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [ConfigAdd](#)
- [AddFolderIcon](#)
- [FileInsertLine](#)
- [BatchAdd](#)
- [ReplaceFolderIcon](#)

# RESET

RESET is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FindFile](#)

# RESTART

RESTART is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [FileGrep](#)
- [PathFind](#)
- [ConfigFind](#)
- [BatchFind](#)

# ROOT

ROOT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [DeleteDir](#)

# RUN\_MAXIMIZED

RUN\_MAXIMIZED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [AddFolderIcon](#)
- [ReplaceFolderIcon](#)

# RUN\_MINIMIZED

RUN\_MINIMIZED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [QueryProgItem](#)
- [AddFolderIcon](#)
- [ReplaceFolderIcon](#)

## SELECTFOLDER

SELECTFOLDER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SelectFolder](#)

## SELFREGISTER

SELFREGISTER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [VerUpdateFile](#)
- [XCopyFile](#)

## SELFREGISTERBATCH

SELFREGISTERBATCH is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Enable](#)
- [Disable](#)

## SELFREGISTRATIONPROCESS

SELFREGISTRATIONPROCESS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Do](#)

## SERIAL

SERIAL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

# SERVICE\_ADAPTER



**Project** • This information applies to InstallScript projects.

SERVICE\_ADAPTER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

## Used With

- [SERVICE\\_IS\\_PARAMS](#)

# SERVICE\_ALL\_ACCESS



**Project** • This information applies to InstallScript projects.

SERVICE\_ALL\_ACCESS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

## Used With

- [SERVICE\\_IS\\_PARAMS](#)

# SERVICE\_AUTO\_START



**Project** • This information applies to InstallScript projects.

SERVICE\_AUTO\_START is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

## Used With

- [SERVICE\\_IS\\_PARAMS](#)

# SERVICE\_BOOT\_START



**Project** • This information applies to InstallScript projects.

SERVICE\_BOOT\_START is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_PARAMS](#)

## SERVICE\_CHANGE\_CONFIG



---

**Project** • This information applies to InstallScript projects.

SERVICE\_CHANGE\_CONFIG is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_PARAMS](#)

## SERVICE\_CONTINUE\_PENDING



---

**Project** • This information applies to InstallScript projects.

SERVICE\_CONTINUE\_PENDING is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_STATUS](#)

## SERVICE\_DEMAND\_START



---

**Project** • This information applies to InstallScript projects.

SERVICE\_DEMAND\_START is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_PARAMS](#)

## SERVICE\_DISABLED



**Project** • This information applies to InstallScript projects.

SERVICE\_DISABLED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_PARAMS](#)

## SERVICE\_ENUMERATE\_DEPENDENTS



**Project** • This information applies to InstallScript projects.

SERVICE\_ENUMERATE\_DEPENDENTS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_PARAMS](#)

## SERVICE\_ERROR\_CRITICAL



**Project** • This information applies to InstallScript projects.

SERVICE\_ERROR\_CRITICAL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_PARAMS](#)

## SERVICE\_ERROR\_IGNORE



**Project** • This information applies to InstallScript projects.



SERVICE\_ERROR\_IGNORE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_PARAMS](#)

## SERVICE\_ERROR\_NORMAL



---

**Project** • This information applies to InstallScript projects.

SERVICE\_ERROR\_NORMAL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_PARAMS](#)

## SERVICE\_ERROR\_SEVERE



---

**Project** • This information applies to InstallScript projects.

SERVICE\_ERROR\_SEVERE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_PARAMS](#)

## SERVICE\_FILE\_SYSTEM\_DRIVER



---

**Project** • This information applies to InstallScript projects.

SERVICE\_FILE\_SYSTEM\_DRIVER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_PARAMS](#)

## SERVICE\_FLAG\_DIFX\_32



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript Object*

SERVICE\_FLAG\_DIFX\_32 is a predefined constant that is used to represent a value that can be set as a bit flag in the system variable [ENABLED\\_ISERVICES](#). You cannot change the value of a predefined constant.

## SERVICE\_FLAG\_DIFX\_AMD64



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript Object*

SERVICE\_FLAG\_DIFX\_AMD64 is a predefined constant that is used to represent a value that can be set as a bit flag in the system variable [ENABLED\\_ISERVICES](#). You cannot change the value of a predefined constant.

## SERVICE\_FLAG\_DIFX\_IA64



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript Object*

SERVICE\_FLAG\_DIFX\_IA64 is a predefined constant that is used to represent a value that can be set as a bit flag in the system variable [ENABLED\\_ISERVICES](#). You cannot change the value of a predefined constant.

## SERVICE\_FLAG\_ISFONTREG



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript Object*

SERVICE\_FLAG\_ISFONTREG is a predefined constant that is used to represent a value that can be set as a bit flag in the system variable [ENABLED\\_ISERVICES](#). If the expression `ENABLED_ISERVICES & SERVICE_FLAG_ISFONTREG` is equal to a non-zero value, global font registration is currently enabled. You cannot change the value of a predefined constant.

# SERVICE\_INTERACTIVE\_PROCESS



**Project** • This information applies to InstallScript projects.

SERVICE\_INTERACTIVE\_PROCESS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

## Used With

- [SERVICE\\_IS\\_PARAMS](#)

# SERVICE\_INTERROGATE



**Project** • This information applies to InstallScript projects.

SERVICE\_INTERROGATE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

## Used With

- [SERVICE\\_IS\\_PARAMS](#)

# SERVICE\_ISFONTREG

SERVICE\_ISFONTREG is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [Disable](#)

# SERVICE\_ISUPDATE



**Project** • This information applies to InstallScript projects.

This constant is obsolete. For information on adding FlexNet Connect support to an InstallScript project, consult the [Knowledge Base](#).

## SERVICE\_KERNEL\_DRIVER



**Project** • This information applies to InstallScript projects.

SERVICE\_KERNEL\_DRIVER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_PARAMS](#)

## SERVICE\_PAUSED



**Project** • This information applies to InstallScript projects.

SERVICE\_PAUSED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_STATUS](#)

## SERVICE\_PAUSE\_CONTINUE



**Project** • This information applies to InstallScript projects.

SERVICE\_PAUSE\_CONTINUE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_PARAMS](#)

## SERVICE\_PAUSE\_PENDING



**Project** • This information applies to InstallScript projects.

SERVICE\_PAUSE\_PENDING is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_STATUS](#)

## SERVICE\_QUERY\_CONFIG



**Project** • This information applies to InstallScript projects.

SERVICE\_QUERY\_CONFIG is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_PARAMS](#)

## SERVICE\_QUERY\_STATUS



**Project** • This information applies to InstallScript projects.

SERVICE\_QUERY\_STATUS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_PARAMS](#)

## SERVICE\_RECOGNIZER\_DRIVER



**Project** • This information applies to InstallScript projects.

SERVICE\_RECOGNIZER\_DRIVER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_PARAMS](#)

## SERVICE\_RUNNING



---

**Project** • This information applies to InstallScript projects.

SERVICE\_RUNNING is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_STATUS](#)

## SERVICE\_START



---

**Project** • This information applies to InstallScript projects.

SERVICE\_START is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_PARAMS](#)

## SERVICE\_START\_PENDING



---

**Project** • This information applies to InstallScript projects.

SERVICE\_START\_PENDING is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_STATUS](#)

## SERVICE\_STOP



---

**Project** • This information applies to InstallScript projects.

SERVICE\_STOP is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_PARAMS](#)

## SERVICE\_STOPPED



---

**Project** • This information applies to InstallScript projects.

SERVICE\_STOPPED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_STATUS](#)

## SERVICE\_STOP\_PENDING



---

**Project** • This information applies to InstallScript projects.

SERVICE\_STOP\_PENDING is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_STATUS](#)

## SERVICE\_SYSTEM\_START



---

**Project** • This information applies to InstallScript projects.

SERVICE\_SYSTEM\_START is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_PARAMS](#)

# SERVICE\_USER\_DEFINED\_CONTROL



**Project** • This information applies to InstallScript projects.

SERVICE\_USER\_DEFINED\_CONTROL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

## Used With

- [SERVICE\\_IS\\_PARAMS](#)

# SERVICE\_WIN32\_OWN\_PROCESS



**Project** • This information applies to InstallScript projects.

SERVICE\_WIN32\_OWN\_PROCESS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

## Used With

- [SERVICE\\_IS\\_PARAMS](#)

# SERVICE\_WIN32\_SHARE\_PROCESS



**Project** • This information applies to InstallScript projects.

SERVICE\_WIN32\_SHARE\_PROCESS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

## Used With

- [SERVICE\\_IS\\_PARAMS](#)

# SETUPTYPE

SETUPTYPE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [SetupType](#)



## SETUPTYPE\_INFO\_DESCRIPTION

SETUPTYPE\_INFO\_DESCRIPTION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureSetupTypeGetData](#)

## SETUPTYPE\_INFO\_DISPLAYNAME

SETUPTYPE\_INFO\_DISPLAYNAME is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureSetupTypeGetData](#)

## SETUPTYPE\_STR\_COMPACT

SETUPTYPE\_STR\_COMPACT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureStandardSetupTypeSet](#)

## SETUPTYPE\_STR\_COMPLETE

SETUPTYPE\_STR\_COMPLETE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureStandardSetupTypeSet](#)

## SETUPTYPE\_STR\_CUSTOM

SETUPTYPE\_STR\_CUSTOM is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureStandardSetupTypeSet](#)

## SETUPTYPE\_STR\_TYPICAL

SETUPTYPE\_STR\_TYPICAL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [FeatureStandardSetupTypeSet](#)

## SETUP\_PACKAGE



---

**Project** • This information applies to InstallScript projects.

SETUP\_PACKAGE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [Is](#)

## SEVERE

SEVERE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [MessageBox](#)
- [SprintfBox](#)

## SHAREDFILE

SHAREDFILE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [XCopyFile](#)
- [VerUpdateFile](#)
- [DeinstallStart](#)
- [InstallationInfo](#)
- [System](#)

# SILENTMODE

SILENTMODE is a predefined constant that can be used to test whether or not a setup is running in silent mode. For more information, refer to the InstallShield system variable [MODE](#).

# SKIN\_LOADED



**Project** • This information applies to InstallScript projects.

SKIN\_LOADED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [Is](#)

# SQL\_BATCH\_INSTALL

SQL\_BATCH\_INSTALL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [SQLRTGetBatchList](#)

# SQL\_BATCH\_UNINSTALL

SQL\_BATCH\_UNINSTALL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [SQLRTGetBatchList](#)

# SQL\_BROWSE\_ALIAS

SQL\_BROWSE\_ALIAS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [SQLRTGetBrowseOption](#)
- [SQLRTSetBrowseOption](#)

## SQL\_BROWSE\_ALL

SQL\_BROWSE\_ALL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SQLRTGetBrowseOption](#)
- [SQLRTSetBrowseOption](#)

## SQL\_BROWSE\_LOCAL

SQL\_BROWSE\_LOCAL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SQLRTGetBrowseOption](#)
- [SQLRTSetBrowseOption](#)

## SQL\_BROWSE\_REMOTE

SQL\_BROWSE\_REMOTE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SQLRTGetBrowseOption](#)
- [SQLRTSetBrowseOption](#)

## SQL\_ERROR\_GET\_SCHEMA\_VERSION

SQL\_ERROR\_GET\_SCHEMA\_VERSION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SQLRTGetComponentScriptError](#)
- [SQLRTGetComponentScriptError2](#)

## SQL\_ERROR\_SCRIPT\_COMMAND\_ERROR

SQL\_ERROR\_SCRIPT\_COMMAND\_ERROR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [SQLRTGetComponentScriptError](#)
- [SQLRTGetComponentScriptError2](#)

## SQL\_ERROR\_SCRIPT\_CONNECTION\_NOT\_OPEN

SQL\_ERROR\_SCRIPT\_CONNECTION\_NOT\_OPEN is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [SQLRTGetComponentScriptError](#)
- [SQLRTGetComponentScriptError2](#)

## SQL\_ERROR\_SCRIPT\_UNABLE\_OPEN\_FILE

SQL\_ERROR\_SCRIPT\_UNABLE\_OPEN\_FILE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [SQLRTGetComponentScriptError](#)
- [SQLRTGetComponentScriptError2](#)

## SQL\_ERROR\_SET\_SCHEMA\_VERSION

SQL\_ERROR\_SET\_SCHEMA\_VERSION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [SQLRTGetComponentScriptError](#)
- [SQLRTGetComponentScriptError2](#)

## SRCINSTALLDIR




---

**Note** • *SRCTARGETDIR* replaces *SRCINSTALLDIR*.

SRCINSTALLDIR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [VarRestore](#)
- [VarSave](#)

## SRCTARGETDIR



**Note** • *SRCTARGETDIR* replaces *SRCINSTALLDIR*.

SRCTARGETDIR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [VarRestore](#)
- [VarSave](#)

## SSP\_PROPERTY\_NO\_NEW\_INSTALL\_HIGHLIGHT

SSP\_PROPERTY\_NO\_NEW\_INSTALL\_HIGHLIGHT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetShortcutProperty](#)

## SSP\_PROPERTY\_NO\_STARTSCREEN\_PIN

SSP\_PROPERTY\_NO\_STARTSCREEN\_PIN is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetShortcutProperty](#)

## SSP\_PROPERTY\_PREVENT\_PINNING

SSP\_PROPERTY\_PREVENT\_PINNING is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetShortcutProperty](#)

## STANDARD\_RIGHTS\_ALL

STANDARD\_RIGHTS\_ALL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## STANDARD\_RIGHTS\_EXECUTE

STANDARD\_RIGHTS\_EXECUTE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## STANDARD\_RIGHTS\_READ

STANDARD\_RIGHTS\_READ is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## STANDARD\_RIGHTS\_REQUIRED

STANDARD\_RIGHTS\_REQUIRED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_PARAMS](#)
- [SetObjectPermissions](#)

## STANDARD\_RIGHTS\_WRITE

STANDARD\_RIGHTS\_WRITE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SetObjectPermissions](#)

## STATUS

STATUS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [PlaceWindow](#)
- [Enable](#)
- [Disable](#)

## STATUSBAR

STATUSBAR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetColor](#)

## STATUSBBD

STATUSBBD is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Enable](#)
- [Disable](#)

## STATUSDLG

STATUSDLG is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [PlaceWindow](#)
- [Enable](#)
- [Disable](#)



## STATUSEX

STATUSEX is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [PlaceWindow](#)
- [Enable](#)
- [Disable](#)

## STATUSOLD

STATUSOLD is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [PlaceWindow](#)
- [Enable](#)
- [Disable](#)

## STRINGLIST

STRINGLIST is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ListCreate](#)

## STYLE\_BOLD

STYLE\_BOLD is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetFont](#)
- [SetFont](#)

## STYLE\_ITALIC

STYLE\_ITALIC is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [GetFont](#)
- [SetFont](#)

## STYLE\_NORMAL

STYLE\_NORMAL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [GetFont](#)
- [SetFont](#)

## STYLE\_SHADOW

STYLE\_SHADOW is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [SetFont](#)

## STYLE\_UNDERLINE

STYLE\_UNDERLINE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [GetFont](#)
- [SetFont](#)

## SW\_MAXIMIZE

SW\_MAXIMIZE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

**Used With**

- [ShowProgramFolder](#)

## SW\_MINIMIZE

SW\_MINIMIZE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ShowProgramFolder](#)

## SW\_RESTORE

SW\_RESTORE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ShowProgramFolder](#)

## SW\_SHOW

SW\_SHOW is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ShowProgramFolder](#)

## SYNCHRONIZE

SYNCHRONIZE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_PARAMS](#)
- [SetObjectPermissions](#)

## SYS\_BOOTMACHINE

SYS\_BOOTMACHINE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [System](#)
- [RebootDialog](#)

- [SdFinishReboot](#)

## SYSTEM\_DPI

SYSTEM\_DPI is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## SYSTEM\_DPI\_SCALING

SYSTEM\_DPI\_SCALING is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## TBYTES

TBYTES is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [ConvertSizeToUnits](#)
- [StrConvertSizeUnit](#)

## TILED

TILED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [PlaceBitmap](#)

## TIME

TIME is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

# TRUE

TRUE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [AskOptions](#)
- [CtrlSetMultCurSel](#)
- [DialogSetInfo](#)
- [FeatureAddItem](#)
- [FeatureGetData](#)
- [FeatureIsItemSelected](#)
- [FeatureSelectItem](#)
- [FeatureTotalSize](#)
- [LongPathToQuote](#)
- [SdDiskSpace2](#)
- [SelectDir](#)
- [SdShowMsg](#)
- [SQLDatabaseBrowse](#)
- [SQLRTConnect](#)
- [SQLRTConnect2](#)
- [SQLRTConnectDB](#)
- [SQLRTGetDatabases](#)
- [SQLRTGetServers](#)
- [SQLRTGetServers2](#)
- [SQLRTPutConnectionAuthentication](#)
- [SQLRTTestConnection](#)
- [SQLRTTestConnection2](#)
- [SQLServerSelectLogin](#)
- [SQLServerSelectLogin2](#)

# TTFONTFILEINFO\_FONTTITLE

TTFONTFILEINFO\_FONTTITLE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetTrueTypeFontFileInfo](#)

## TYPICAL

TYPICAL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SdSetupType](#)
- [SetupType](#)

## UPDATE\_SERVICE\_INSTALL



---

**Project** • This information applies to InstallScript projects.

This constant is obsolete. For information on adding FlexNet Connect support to an InstallScript project, consult the [Knowledge Base](#).

## UPDATESERVICECOMPONENT



---

**Project** • This information applies to InstallScript projects.

This constant is obsolete. For information on adding FlexNet Connect support to an InstallScript project, consult the [Knowledge Base](#).

## UPPER\_LEFT

UPPER\_LEFT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [PlaceBitmap](#)
- [PlaceWindow](#)

## UPPER\_RIGHT

UPPER\_RIGHT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [PlaceBitmap](#)
- [PlaceWindow](#)

## URL



**Project** • This information applies to InstallScript projects.

URL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Is](#)

## USER\_ADMINISTRATOR

USER\_ADMINISTRATOR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Is](#)

## USER\_INADMINGROUP

USER\_INADMINGROUP is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Is](#)

## USER\_POWERUSER



**Project** • This information applies to InstallScript projects.

USER\_POWERUSER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Is](#)

## USE\_LOADED\_SKIN



**Project** • This information applies to InstallScript projects.

USE\_LOADED\_SKIN is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Disable](#)
- [Enable](#)

## VALID\_PATH

VALID\_PATH is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Is](#)

## VERSION\_COMPARE\_RESULT\_NEWER



**Project** • This information applies to InstallScript projects.

VERSION\_COMPARE\_RESULT\_NEWER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [VerProductCompareVersions](#)

## VERSION\_COMPARE\_RESULT\_NEWER\_NOT\_SUPPORTED



**Project** • This information applies to InstallScript projects.

VERSION\_COMPARE\_RESULT\_NEWER\_NOT\_SUPPORTED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



### Used With

- [VerProductCompareVersions](#)

## VERSION\_COMPARE\_RESULT\_NOT\_INSTALLED



**Project** • This information applies to InstallScript projects.

VERSION\_COMPARE\_RESULT\_NOT\_INSTALLED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [VerProductCompareVersions](#)

## VERSION\_COMPARE\_RESULT\_OLDER



**Project** • This information applies to InstallScript projects.

VERSION\_COMPARE\_RESULT\_OLDER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [VerProductCompareVersions](#)

## VERSION\_COMPARE\_RESULT\_SAME



**Project** • This information applies to InstallScript projects.

VERSION\_COMPARE\_RESULT\_SAME is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [VerProductCompareVersions](#)

## VERSION\_PREVIOUS\_VERSION\_DELIMITER



**Project** • This information applies to InstallScript projects.

VERSION\_PREVIOUS\_VERSION\_DELIMITER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

#### Used With

- [VerProductCompareVersions](#)

## VER\_DLL\_NOT\_FOUND

VER\_DLL\_NOT\_FOUND is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

#### Used With

- [VerSearchAndUpdateFile](#)

## VER\_UPDATE\_ALWAYS

VER\_UPDATE\_ALWAYS is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

#### Used With

- [VerUpdateFile](#)
- [VerSearchAndUpdateFile](#)

## VER\_UPDATE\_COND

VER\_UPDATE\_COND is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

#### Used With

- [VerSearchAndUpdateFile](#)

## VIDEO

VIDEO is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

#### Used With

- [GetSystemInfo](#)

# VIRTUAL\_MACHINE\_TYPE

VIRTUAL\_MACHINE\_TYPE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetSystemInfo](#)

# VOLUMELABEL

VOLUMELABEL is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [GetSystemInfo](#)

# WARNING

WARNING is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [MessageBox](#)
- [SprintfBox](#)

# WEB\_BASED\_SETUP



---

**Project** • This information applies to InstallScript projects.

WEB\_BASED\_SETUP is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

## Used With

- [Is](#)

# WELCOME

WELCOME is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Welcome](#)

## WHITE

WHITE is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetTitle](#)

## WILL\_REBOOT

WILL\_REBOOT is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or event handlers. You cannot change the value of a predefined constant.

### Used With

- [RebootDialog](#)
- [SdFinishReboot](#)

## WINDOWS\_SHARED

WINDOWS\_SHARED is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Is](#)

## WINMAJOR

WINMAJOR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## WINMINOR

WINMINOR is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [GetSystemInfo](#)

## WOW64FSREDIRECTION



**Project** • This information applies to *InstallScript* projects.

WOW64FSREDIRECTION is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [Disable](#)
- [Enable](#)

## WRITE\_DAC

WRITE\_DAC is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_PARAMS](#)
- [SetObjectPermissions](#)

## WRITE\_OWNER

WRITE\_OWNER is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions or assigned to one or more system variables. You cannot change the value of a predefined constant.

### Used With

- [SERVICE\\_IS\\_PARAMS](#)
- [SetObjectPermissions](#)

## YELLOW

YELLOW is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [SetColor](#)
- [SetTitle](#)

## YES

YES is a predefined constant that is used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

### Used With

- [AskYesNo](#)
- [SdLicense](#)
- [SdConfirmNewDir](#)
- [SdConfirmRegistration](#)

## \_MAX\_PATH



---

**Project** • This information applies to *InstallScript* projects.

MAX\_PATH is a predefined constant that is used to represent the maximum length of a path variable that is passed to a Windows API function. The following sample code lines illustrate possible uses of \_MAX\_PATH:

```
string szPath[_MAX_PATH]; /* variable declaration */
...
Kernel32.GetTempPathA( _MAX_PATH, szPath ); /* Windows API function call */
```

# Predefined Script Variables

This section includes a list of predefined script variables that are reserved during script compilation.

- `__FILE__`
- `__LINE__`
- `BASICMSI`
- `INSTALLSCRIPTMSI`
- `INSTALLSCRIPTMSIEEUI`
- `ISUS_PRODUCT_CODE`
- `SERVICE_IS_PARAMS`
- `SERVICE_IS_STATUS`
- `SUITE_HOSTED`

## `__FILE__`

During script compilation this reserved identifier is replaced by a string that contains the fully qualified name of the source file in which `__FILE__` resides. `__FILE__` can be specified anywhere in a script where a string constant is allowed, but it is most useful when used with `__LINE__` for simple debugging. For example, by constructing a statement like the one below and copying it to strategic locations in your source files during testing, you can easily associate specific parts of your setup with specific sections of your script as you observe your setup run.

```
SprintfBox (INFORMATION, "", "File: %s\nLine:%ld",
            __FILE__, __LINE__);
```

Note that the path will be the location from which the file was compiled, not the location from which the setup is being run. If necessary, the `ParsePath` function can be called with `__FILE__` in the second parameter to extract parts of the fully qualified file name. The code fragment below extracts the file name and displays it.

```
ParsePath (svReturnString, __FILE__, FILENAME);
MessageBox (svReturnString, INFORMATION);
```

For more complete and powerful debugging, use the InstallScript Debugger by clicking Debug on the Build menu in InstallShield. For more information, see the InstallScript Debugger Help.

## `__LINE__`

During setup compilation, this reserved identifier is replaced by the number of the source file line in which `__LINE__` resides. Note that `__LINE__` can be specified anywhere in a script where a number constant is allowed, but it is most useful when used with `__FILE__` for simple debugging. For example, by constructing a statement like the one below and copying it to strategic locations in your source files during testing, you can easily associate specific parts of your setup with specific sections of your script as you observe your setup run.

```
SprintfBox (INFORMATION, "", "File: %s\nLine:%ld", __FILE__, __LINE__);
```

For more complete and powerful debugging, use the InstallScript Debugger by clicking Debug on the Build menu in InstallShield. For more information, see the InstallScript Debugger Help.

## BASICMSI

The BASICMSI script variable is defined for Basic MSI projects, but it is undefined and evaluates as zero in InstallScript MSI projects and in InstallScript projects.



**Note** • *BASICMSI is not a preprocessor switch; therefore, you can use this script variable to create script code that will work differently in different project types without being recompiled.*

You can use BASICMSI to write a single script that produces different behavior in the different project types by including code such as the following in your script:

```
if( BASICMSI ) then
    //Code for Basic MSI projects
else
    //Code for InstallScript MSI or InstallScript projects
endif;
```

## INSTALLSCRIPTMSI

INSTALLSCRIPTMSI is defined in InstallScript MSI and Basic MSI projects, but it is undefined and evaluates as zero in InstallScript projects.



**Note** • *INSTALLSCRIPTMSI is not a preprocessor switch; therefore, you can use this script variable to create script code that will work differently in the two project types without being recompiled.*

You can use INSTALLSCRIPTMSI to write a single script that produces different behavior in the different project types by including code such as the following in your script:

```
if( INSTALLSCRIPTMSI ) then
    //Code for InstallScript MSI and Basic MSI projects...
else
    //Code for InstallScript projects...
endif;
```

## INSTALLSCRIPTMSIEEUI

The INSTALLSCRIPTMSIEEUI variable is set to enable an installation to determine at run time whether the InstallScript engine is used as the embedded user interface (UI) handler for an InstallScript MSI installation. This implementation is also known as the new style of InstallScript UI.

If the InstallScript engine is used as the embedded user interface handler for an InstallScript MSI installation, INSTALLSCRIPTMSIEEUI is set to TRUE. If it is not an embedded user interface handler, this variable is set to FALSE.



**Tip** • *For information on using the InstallScript engine as an embedded user interface handler for an InstallScript MSI installation, see [Using the InstallScript Engine as an External vs. Embedded UI Handler for InstallScript MSI Installations](#).*

You can use INSTALLSCRIPTMSIEEUI to write a single script that produces different behavior for the different user interface styles by including code such as the following in your script:



```

if( INSTALLSCRIPTMSIEUI ) then
    //Code for new-style InstallScript MSI installations
    //(InstallScript engine as an embedded UI handler)...
else
    //Code for traditional-style InstallScript MSI installations
    (InstallScript engine as an external UI handler)...
endif;

```

## ISUS\_PRODUCT\_CODE

The ISUS\_PRODUCT\_CODE variable is a read-write script variable which is set to PRODUCT\_GUID during initialization. Like PRODUCT\_GUID, this variable is not linked to a text-sub. Therefore, if customized, this script variable must be customized each time the setup is run, even during maintenance mode.

## SERVICE\_IS\_PARAMS



**Project** • This information applies to InstallScript projects.

The SERVICE\_IS\_PARAMS variable is initialized automatically during installation initialization by a call to [ServiceInitParams](#).



**Note** • Certain InstallScript service functions internally call the Windows API functions *OpenSCManager*, *CreateService*, or *ChangeServiceConfig*. The following members of the structured variable SERVICE\_IS\_PARAMS specify the corresponding arguments for these Windows API functions:

- SERVICE\_IS\_PARAMS.lpMachineName
- SERVICE\_IS\_PARAMS.lpDatabaseName
- SERVICE\_IS\_PARAMS.dwDesiredAccess
- SERVICE\_IS\_PARAMS.dwServiceType
- SERVICE\_IS\_PARAMS.dwStartType
- SERVICE\_IS\_PARAMS.dwErrorControl
- SERVICE\_IS\_PARAMS.lpLoadOrderGroup
- SERVICE\_IS\_PARAMS.lpdwTagId
- SERVICE\_IS\_PARAMS.lpDependencies
- SERVICE\_IS\_PARAMS.lpServiceStartName
- SERVICE\_IS\_PARAMS.lpPassword

The following members of SERVICE\_IS\_PARAMS control how the installation behaves when waiting for a service to reach a desired state. See the descriptions for each member for additional information.



- SERVICE\_IS\_PARAMS.nWaitHintMin
- SERVICE\_IS\_PARAMS.nWaitHintMax
- SERVICE\_IS\_PARAMS.nStartServiceWaitCount
- SERVICE\_IS\_PARAMS.nStopServiceWaitCount

The SERVICE\_IS\_PARAMS script variable has the following members:

**Table 1 • SERVICE\_IS\_PARAMS Parameters**

Member	Description
<b>dwServiceType</b>	<p>This member can be set to these predefined constants:</p> <ul style="list-style-type: none"> <li>• SERVICE_WIN32_OWN_PROCESS</li> <li>• SERVICE_WIN32_SHARE_PROCESS</li> <li>• SERVICE_KERNEL_DRIVER</li> <li>• SERVICE_FILE_SYSTEM_DRIVER</li> <li>• SERVICE_ADAPTER</li> <li>• SERVICE_RECOGNIZER_DRIVER</li> <li>• SERVICE_INTERACTIVE_PROCESS</li> </ul>
<b>dwStartType</b>	<p>This member can be set to these predefined constants:</p> <ul style="list-style-type: none"> <li>• SERVICE_BOOT_START</li> <li>• SERVICE_SYSTEM_START</li> <li>• SERVICE_AUTO_START</li> <li>• SERVICE_DEMAND_START</li> <li>• SERVICE_DISABLED</li> </ul>
<b>dwErrorControl</b>	<p>This member can be set to these predefined constants:</p> <ul style="list-style-type: none"> <li>• SERVICE_ERROR_IGNORE</li> <li>• SERVICE_ERROR_NORMAL</li> <li>• SERVICE_ERROR_SEVERE</li> <li>• SERVICE_ERROR_CRITICAL</li> </ul>
<b>nWaitHintMin</b>	<p>Specifies the minimum dwWaitHint wait time (in milliseconds). If a service specifies a dwWaitHint smaller than nWaitHintMin, nWaitHintMin is used instead as the wait time. This applies to both starting and stopping services.</p> <p>The default value of this member variable is 1000 (1 second), and is set by calling ServiceInitParams. See the MSDN documentation for more information on how a service sets dwWaitHint.</p>
<b>nWaitHintMax</b>	<p>Specifies the maximum dwWaitHint wait time (in milliseconds). If a service specifies a dwWaitHint longer than nWaitHintMax, nWaitHintMax is used instead as the wait time. This applies to both starting and stopping services.</p> <p>The default value of this member variable is 10000 (10 seconds), and is set by calling ServiceInitParams. See the MSDN documentation for more information on how a service sets dwWaitHint.</p>

**Table 1 • SERVICE\_IS\_PARMS Parameters (cont.)**

Member	Description
<b>nStartServiceWaitCount</b>	<p>Specifies the service start timeout in seconds. This value can be set to a specific value to force the installation to stop waiting after the specified interval regardless of whether the service has reached the desired state.</p> <p></p> <p><b>Important</b> • Unlike <i>nWaitHintMax</i>, if a service specifies a long <i>dwWaitHint</i>, the installation does not interrupt this wait regardless of the value of this parameter. Therefore, it is recommended that this value not be changed from the default value of <i>INFINITE</i> set by <i>ServiceInitParams</i>. Instead, update <i>nWaitHintMax</i> to prevent an undesired wait interval.</p>
<b>nStopServiceWaitCount</b>	<p>Specifies the service stop timeout in seconds. This value can be set to a specific value to force the setup to stop waiting after the specified interval regardless of whether the service has reached the desired state.</p> <p></p> <p><b>Important</b> • Unlike <i>nWaitHintMax</i>, if a service specifies a long <i>dwWaitHint</i>, the installation does not interrupt this wait regardless of the value of this parameter. Therefore, it is recommended that this value not be changed from the default value of <i>INFINITE</i> set by <i>ServiceInitParams</i>. Instead, update <i>nWaitHintMax</i> to prevent an undesired wait interval.</p>

## Additional Information

For more information about the Windows API functions `OpenSCManager`, `CreateService`, or `ChangeServiceConfig`, consult the Windows API documentation.

# SERVICE\_IS\_STATUS



**Project** • This information applies to *InstallScript* projects.

When you call **ServiceGetServiceState**, this structured variable returns identification information about the service. This system variable is of type SERVICE\_IS\_STATUS; it has the following members:

**Table 2 • SERVICE\_IS\_STATUS Parameters**

Member	Meaning
<b>dwServiceType</b>	<p>The type of service. This member can be one of the following values.</p> <ul style="list-style-type: none"> <li>• <b>SERVICE_FILE_SYSTEM_DRIVER</b>—The service is a file system driver.</li> <li>• <b>SERVICE_KERNEL_DRIVER</b>—The service is a device driver.</li> <li>• <b>SERVICE_WIN32_OWN_PROCESS</b>—The service runs in its own process.</li> <li>• <b>SERVICE_WIN32_SHARE_PROCESS</b>—The service shares a process with other services.</li> </ul> <p>If the service type is either SERVICE_WIN32_OWN_PROCESS or SERVICE_WIN32_SHARE_PROCESS, the following type may also be specified.</p> <ul style="list-style-type: none"> <li>• <b>SERVICE_INTERACTIVE_PROCESS</b>—The service can interact with the desktop.</li> </ul>
<b>dwCurrentState</b>	<p>The current state of the service. This member can be one of the following values.</p> <ul style="list-style-type: none"> <li>• <b>SERVICE_CONTINUE_PENDING</b>—The service continue is pending.</li> <li>• <b>SERVICE_PAUSE_PENDING</b>—The service pause is pending.</li> <li>• <b>SERVICE_PAUSED</b>—The service is paused.</li> <li>• <b>SERVICE_RUNNING</b>—The service is running.</li> <li>• <b>SERVICE_START_PENDING</b>—The service is starting.</li> <li>• <b>SERVICE_STOP_PENDING</b>—The service is stopping.</li> <li>• <b>SERVICE_STOPPED</b>—The service is not running.</li> </ul>
<b>dwWin32ExitCode</b>	<p>A Win32 error code that the service uses to report an error that occurs when it is starting or stopping. To return an error code specific to the service, the service must set this value to ERROR_SERVICE_SPECIFIC_ERROR to indicate that the dwServiceSpecificExitCode member contains the error code. The service should set this value to NO_ERROR when it is running and on normal termination.</p>
<b>dwServiceSpecificExitCode</b>	<p>A service-specific error code that the service returns when an error occurs while the service is starting or stopping. This value is ignored unless the dwWin32ExitCode member is set to ERROR_SERVICE_SPECIFIC_ERROR.</p>

**Table 2 • SERVICE\_IS\_STATUS Parameters (cont.)**

Member	Meaning
<b>dwCheckPoint</b>	A value that the service increments periodically to report its progress during a lengthy start, stop, pause, or continue operation. For example, the service should increment this value as it completes each step of its initialization when it is starting up. The user interface program that invoked the operation on the service uses this value to track the progress of the service during a lengthy operation. This value is not valid and should be zero when the service does not have a start, stop, pause, or continue operation pending.
<b>dwWaitHint</b>	An estimate of the amount of time, in milliseconds, that the service expects a pending start, stop, pause, or continue operation to take before the service makes its next call to the Windows API function <code>SetServiceStatus</code> with either an incremented <code>dwCheckPoint</code> value or a change in <code>dwCurrentState</code> . If the amount of time specified by <code>dwWaitHint</code> passes, and <code>dwCheckPoint</code> has not been incremented, or <code>dwCurrentState</code> has not changed, the service control manager or service control program can assume that an error has occurred.
<b>dwControlsAccepted</b>	The control codes that the service will accept and process in its handler function. A user interface process can control a service by specifying a control command in the Windows API function <code>ControlService</code> . By default, all services accept the <code>SERVICE_CONTROL_INTERROGATE</code> value. Table 3 lists the possible control codes for this member.

### dwControlAccepted Control Codes

This table lists the possible control codes for `dwControlAccept` script variable.

**Table 3 • dwControlsAccepted Control Codes**

Control Code	Description
<b>SERVICE_ACCEPT_NETBINDCHANGE</b>	<p>The service is a network component that can accept changes in its binding without being stopped and restarted.</p> <p>This control code allows the service to receive <code>SERVICE_CONTROL_NETBINDADD</code>, <code>SERVICE_CONTROL_NETBINDREMOVE</code>, <code>SERVICE_CONTROL_NETBINDENABLE</code>, and <code>SERVICE_CONTROL_NETBINDDISABLE</code> notifications.</p>
<b>SERVICE_ACCEPT_PARAMCHANGE</b>	<p>The service can reread its startup parameters without being stopped and restarted.</p> <p>This control code allows the service to receive <code>SERVICE_CONTROL_PARAMCHANGE</code> notifications.</p>

**Table 3 •** dwControlsAccepted Control Codes (cont.)

Control Code	Description
<b>SERVICE_ACCEPT_PAUSE_CONTINUE</b>	The service can be paused and continued.  This control code allows the service to receive SERVICE_CONTROL_PAUSE and SERVICE_CONTROL_CONTINUE notifications.
<b>SERVICE_ACCEPT_SHUTDOWN</b>	The service is notified when system shutdown occurs.  This control code allows the service to receive SERVICE_CONTROL_SHUTDOWN notifications. Note that the Windows API function ControlService cannot send this notification; only the system can send it.
<b>SERVICE_ACCEPT_STOP</b>	The service can be stopped.  This control code allows the service to receive SERVICE_CONTROL_STOP notifications.

The dwControlAccept value can also contain the following extended control codes, which are supported only by service handler functions that are used with the Windows API function **RegisterServiceCtrlHandlerEx**.

**Table 4 •** Extended Control Codes for dwControlAccept

Control Code	Description
<b>SERVICE_ACCEPT_HARDWAREPROFILECHANGE</b>	The service is notified when the computer's hardware profile has changed. This enables the system to send SERVICE_CONTROL_HARDWAREPROFILECHANGE notifications to the service.
<b>SERVICE_ACCEPT_POWEREVENT</b>	The service is notified when the computer's power status has changed. This enables the system to send SERVICE_CONTROL_POWEREVENT notifications to the service.
<b>SERVICE_ACCEPT_SESSIONCHANGE</b>	Whistler: The service is notified when the computer's session status has changed. This enables the system to send SERVICE_CONTROL_SESSIONCHANGE notifications to the service.

## SUITE\_HOSTED



**Project •** This information applies to InstallScript projects.

Use the SUITE\_HOSTED variable to determine whether the InstallScript installation is running as an InstallScript package in an Advanced UI or Suite/Advanced UI installation.

If the InstallScript installation is running as an InstallScript package in an Advanced UI or Suite/Advanced UI installation, SUITE\_HOSTED is set to a non-zero value. If SUITE\_HOSTED is set to zero, the InstallScript installation is not running as an InstallScript package in an Advanced UI or Suite/Advanced UI installation. For example:

```
if SUITE_HOSTED then
    // Code for an InstallScript installation that is run
    // as an InstallScript package in an Suite/Advanced UI
    // or Advanced UI installation
else
    // Code for an InstallScript installation that is run
    // standalone, not as an InstallScript package in an
    // a Suite/Advanced UI or Advanced UI installation
endif;
```





# Data Types and Predefined Structures

## Data Types

InstallScript supports the following data types. Note that some data types can be entered in either lowercase or uppercase letters:


**Table 1 • Data Types**

Data Type	Description
<b>binary</b> <b>BINARY</b>	<p>Indicates binary data (specified in a string variable) to be passed to or retrieved from an external DLL function. Unlike the STRING or WSTRING data types, when the BINARY data type is specified, the InstallScript engine does not attempt to interpret the data as a string or perform any type of data conversion or validation; thus, this data type is used when passing binary data that may or may not consist of valid string characters.</p> <p>Note that this data type can be used only in a prototype of an external DLL function. If this data type is used for a variable instance or as a parameter for a non-DLL InstallScript function, compile error C8116 error occurs.</p> <p>If a standard InstallScript string is passed through the BINARY data type, the characters in the string are passed as ASCII characters. Therefore, the binary type is similar to the STRING data type, not the WSTRING data type, for valid string characters.</p>
<b>BOOL</b>	<p>Boolean data: either TRUE (1) or FALSE (0). Variables of this type should not be used to store any other values. Like C++, InstallScript evaluates non-zero values as TRUE; only the value of zero is evaluated as FALSE. Normally, the value of one is used to indicate TRUE.</p>
<b>char</b> <b>CHAR</b>	<p>Character data: a single 8-bit signed character. When a literal character appears in a script, it must be enclosed within single or double quotation marks. Note that you can assign a numeric ASCII value to a character value. To display a char variable as a character, use the <a href="#">Format Specifiers</a> "%c" with the function <a href="#">SprintfBox</a>. To display the numeric value of a char variable use the specifier "%d".</p> <p>InstallScript character variable types are signed; therefore extended ASCII characters will be interpreted as negative numbers when interpreted numerically. To avoid this problem, assign the value to a number variable; then AND (&amp;) the number variable with the value 255 before interpreting the number.</p>
<b>HWND</b>	<p>Handle to a window. The HWND variable type also can be used to store any other type of handle valid in Windows. HWND variables are normally initialized using the <a href="#">CmdGetHwndDlg</a> or <a href="#">GetWindowHandle</a> functions. Internally, HWND variables are equivalent to the data type NUMBER.</p>
<b>int</b> <b>INT</b>	<p>Equivalent to the number type; provided for convenience.</p>


**Table 1 • Data Types (cont.)**

Data Type	Description
<b>LIST</b>	Pointer to an InstallScript list. LIST variables are always initialized and uninitialized using the <a href="#">ListCreate</a> and <a href="#">ListDestroy</a> functions. Internally, LIST variables are equivalent to the data type NUMBER.
<b>long</b> <b>LONG</b>	Equivalent to the NUMBER type; provided for convenience.
<b>LPSTR</b>	Equivalent to the POINTER type; provided for convenience. For more information, see <a href="#">Pointers</a> .
<b>LPWSTR</b>	Equivalent to the WPOINTER type; provided for convenience. For more information, see <a href="#">Pointers</a> .
<b>number</b> <b>NUMBER</b>	Signed four-byte integer. Number is the recommended data type for storing numeric data. The data type is similar to the LONG variable type of other programming languages. It can hold any value between -2,147,483,648 and +2,147,483,647. Note that all numeric data types in InstallScript are equivalent to the NUMBER variable type.
<b>object</b> <b>OBJECT</b>	A reference to a COM object. The reference is returned by the <a href="#">CreateObject</a> function and assigned to the object variable using the <a href="#">set</a> keyword.
<b>pointer</b> <b>POINTER</b>	A pointer to data. In the case of a pointer to a string variable, the data pointed to is an ANSI string. Pointer variables are normally initialized by using the address-of (&) operator to assign the address of a variable to the pointer variable. For more information, see <a href="#">Pointers</a> .
<b>short</b> <b>SHORT</b>	Equivalent to the NUMBER type; provided for convenience.

**Table 1 • Data Types (cont.)**

Data Type	Description
<b>string</b> <b>STRING</b>	<p>An array of Unicode characters (two bytes per character). String variables, which are similar to arrays of characters in the C++ language, are NULL terminated. However, InstallShield does not support multiple NULL-terminated strings in the same string variable. String variables can be declared with an explicit size of up to 65,535 characters. Strings variables that are declared without an explicit size are automatically sized by InstallShield. Note that string concatenation in a setup script is performed by using the concatenation operator, which is a plus sign (+). Strings to be concatenated are positioned as operands on either side of the operator, as shown in the statement below, which appends the value of szLastName to the value of szFirstName and assigns the resulting string to szFullName:</p> <pre>szFullName = szFirstName + szLastName;</pre> <p>To display a string variable, use the <a href="#">SprintfBox</a> function with the format specifier "%s", or use the <a href="#">MessageBox</a> function.</p> <p></p> <p><b>Note •</b> You can declare string variables as <i>STRING</i> in InstallScript code; string variables that are declared this way in InstallScript code are stored as Unicode strings in string tables. However, if you want to store Unicode strings in structures that are passed outside of the InstallScript code—for example, to a DLL function—you may need to use the <i>WSTRING</i> type when declaring the strings as structure members in your InstallScript code. For more information, see <a href="#">Data Structures</a>.</p>
<b>variant</b> <b>VARIANT</b>	<p>Any kind of data: character, string, numeric, object reference, and so on. It is recommended that you use the other data types whenever possible; the VARIANT data type is necessary only when declaring a script-defined function that takes an array as an argument, for example:</p> <pre>prototype number AverageValue( variant );  function OnBegin( )     number nAverage , nArray(10); begin     /* Assign values to array elements here. */      /* Pass the array to the function. */     nAverage = AverageValue( nArray ); end;</pre> <p>Note that the VARIANT data type cannot be defined within a data structure.</p>

**Table 1 • Data Types (cont.)**

Data Type	Description
<b>void</b> <b>VOID</b>	<p>Void is not a true data type, in the sense that a variable cannot be declared as type void. Void is only used in function prototypes to indicate that the function does not return a value, as in the following:</p> <pre> prototype void Subroutine(int);  function void Subroutine(int); begin     // perform operations, but     // do not return a value end; </pre>
<b>wpointer</b> <b>WPOINTER</b>	<p>A pointer to string data. The POINTER type should be used in all cases except when a pointer to Unicode string data is needed. Pointer variables are normally initialized by using the address-of (&amp;) operator to assign the address of a variable to the pointer variable.</p> <p>For more information, see <a href="#">Pointers</a>.</p>
<b>wstring</b> <b>WSTRING</b>	<p>Same as STRING except that, unlike STRING, it can be used when declaring DLL function calls in which a wide-character string argument or Unicode string argument is expected. For example:</p> <pre> prototype long Kernel32.GetWindowsDirectoryW(BYREF wstring, int); </pre> <p>You can pass a string variable in a WSTRING argument; for example:</p> <pre> wstring svWinDir; ... GetWindowsDirectoryW(svWinDir, 1024); </pre> <div>  <p><b>Note •</b> You can declare string variables as <i>STRING</i> in InstallScript code; string variables that are declared this way in InstallScript code are stored as Unicode strings in string tables. However, if you want to store Unicode strings in structures that are passed outside of the InstallScript code—for example, to a DLL function—you may need to use the <i>WSTRING</i> type when declaring the strings as structure members in your InstallScript code. For more information, see <a href="#">Data Structures</a>.</p> </div>



**Note •** InstallScript does not provide unsigned or floating-point data types.

## Predefined Structures

InstallScript supports the following predefined structures:

**Table 2 • Predefined Structures**

Predefined Structure	Description
<b>_FONTFILEINFO</b>	<p>This is a data structure that is passed to the <a href="#">OnInstalledFontFile</a> and <a href="#">OnUninstallingFontFile</a> event handlers. It has the following members:</p> <ul style="list-style-type: none"> <li>• <b>string szFileName[_MAX_PATH]</b>—The complete path to the font file being installed on the system</li> <li>• <b>string szFaceName[_MAX_PATH]</b>—The face name of the font being installed (if you specify the name for the font file in InstallShield)</li> </ul> <p>Note that the information in the structure is passed to the event handler so that the event handler can use the information. Thereafter, the values of the structure are used by the installation. Therefore, changing the structure members does not have any effect on the installation.</p>
<b>_LAAW_PARAMETERS</b>	For a list of this data structure's members and their purposes and permitted values, see <a href="#">LAAW_PARAMETERS</a> .
<b>_SERVICE_IS_PARAMS</b>	For a list of this data structure's members and their purposes and permitted values, see <a href="#">SERVICE_IS_PARAMS</a> .
<b>_SERVICE_IS_STATUS</b>	For a list of this data structure's members and their purposes and permitted values, see <a href="#">SERVICE_IS_STATUS</a> .
<b>ISURL_COMPONENTS</b>	<p>This is a data structure that contains the constituent parts of a URL. This structure is similar to the Windows API structure URL_COMPONENTS. For an example of the use of this structure, see <a href="#">ParseUrl</a>. This structure has the following members:</p> <ul style="list-style-type: none"> <li>• <b>szScheme</b>—String value that contains the scheme name.</li> <li>• <b>nInternetScheme</b>—Numeric value that indicates the Internet protocol scheme; equals 3 for HTTP and 4 for HTTPS.</li> <li>• <b>szUserName</b>—String value that contains the user name.</li> <li>• <b>szPassword</b>—String value that contains the password.</li> <li>• <b>szHostName</b>—String value that contains the host name.</li> <li>• <b>nInternetPort</b>—Port number.</li> <li>• <b>szUrlPath</b>—String value that contains the URL path.</li> <li>• <b>szExtraInfo</b>—String value that contains the extra information (for example, ?something or #something).</li> </ul>
<b>PROCESS_INFORMATION</b>	For a list of this data structure's members and their purposes and permitted values, see <a href="#">LAAW_PROCESS_INFORMATION</a> .

**Table 2 •** Predefined Structures (cont.)

Predefined Structure	Description
<b>STARTUPINFO</b>	For a list of this data structure's members and their purposes and permitted values, see <a href="#">LAAW_STARTUPINFO</a> .

## Arrays

You can declare and use any InstallScript data type as an array. To declare a variable as an array, append an open parenthesis and a close parenthesis, with an array size optionally between the parentheses, to the variable name in the declaration. For example, the following declares a variable called `nArray` to be an array containing 10 `NUMBER` elements:

```
NUMBER nArray(10);
```

If you do not declare an array size, for example,

```
NUMBER nArray( );
```

the array size defaults to zero. You can resize arrays in the script by using the `Resize` operator. You can get the size of an array by using the `SizeOf` operator.

Use the following syntax to assign values to array elements:

### Syntax

```
<array variable name>(<array index>) = <value>;
```

For example:

```
nArray(0) = 1; /* Array indexing begins with zero. */
nArray(5) = 17;
```

When declaring a script-defined function that takes an array as an argument, do not use an array as the parameter data type; for example:

```
prototype NUMBER AverageValue( NUMBER( ) ); /* This will not compile. */
```

Instead, use the `VARIANT` data type; for example:

```
prototype NUMBER AverageValue( VARIANT );

function OnBegin( )
    NUMBER nAverage, nArray(10);
begin
    /* Assign values to array elements here. */

    /* Pass the array to the function. */
    nAverage = AverageValue( nArray );
end;
```

## Constant Data

A constant is a named data item with a defined value. InstallShield supports two types of constants:

- Predefined constants, such as TRUE and RESET, are part of InstallScript. These constants, which are used as function parameters and return values for built-in functions, cannot be redefined in the script. Attempting to redefine a predefined constant results in a compiler error.
- User-defined constants are declared by the programmer as needed for individual scripts. Although a user-defined constant can be redefined after the initial declaration, it is generally not considered good programming practice to do so.

User-defined constants are declared with a `#define` preprocessor statement. (InstallScript does not support the `const` keyword for declaring variable constants as the C++ language does.) String constants must be enclosed within quotation marks; numeric constants are defined without quotation marks and contain only numeric characters. Once declared, a string constant can be used anywhere that a string literal can be used. Likewise, number constants can be used anywhere that a numeric literal can be used.

In the following example, a string constant and a numeric constant are declared:

```
#define COMPANY_NAME "Example_Company"
#define MAXCOUNT 1000
```

A constant name must follow the rules for InstallScript identifiers. By convention, constant identifiers are created with all uppercase characters. InstallScript's predefined constants follow that convention.

## Data Structures

A data structure is a named data item that consists of logically related variables, called members. In many programming languages, data structures are called records, and the variables within records are called fields. In InstallScript, data structures are similar in form and function to structures in C. They can include members of different data types, and those members within the data structure can be referenced directly by using the [member operator](#) (`.`).

### Defining a Data Structure

To define a data structure, use the keyword `typedef` and follow it with the name of the data structure. Fields in the structure must be defined within a `begin...end` block, as shown in the example below, which defines a structure called `EMPLOYEE`. The data structure `EMPLOYEE` includes three variables: a string variable for an employee's name, a string variable for an employee's department, and a number variable for an employee's phone extension.

```
typedef EMPLOYEE
begin
    STRING szName[50];
    STRING szDepartment[50];
    NUMBER nExtension;
end;
```

When you define a data structure, you are actually defining a new data type. To use a data structure within a program, you must first declare a variable of that type. To do that, use the name of the defined data structure as the data type and follow it with an identifier, as shown in the example below, which creates a variable of type `EMPLOYEE`.

```
EMPLOYEE structEmployee;
```

To reference members of the structure variable, use the member operator (`.`). In the example below, literal values are assigned to each member of `structEmployee`.

```
structEmployee.szName = "I. S. Coder";
structEmployee.szDepartment = "Development";
structEmployee.nExtension = 555;
```

## Restrictions

The following restrictions apply to structures:

- You cannot assign the contents of one structure to another structure with the assignment operator, as in `newstruct = struct1`. Instead, you must copy the structure one element at a time.
- You must specify the size of all `STRING` declarations in a structure—the autosizing functionality in InstallScript does not work in typedef statements.
- You cannot declare a structure within a function.
- You cannot use the `BYREF` operator with a structure, nor can you pass a structure member in a parameter that was declared with the `BYREF` operator. To modify a member of a user-defined structure in a user-defined function, pass a pointer to the structure and then use the [Structure Pointer Operator \(->\)](#) to access the data within the function.
- Referencing a pointer before the address of a data structure has been assigned to the pointer results in a run-time error.

## Unicode Support for Structures

Structures in InstallScript can contain any basic data type, including strings and pointers, or other structures. If a structure needs to contain a Unicode string and the structure is passed to an external DLL, the InstallScript engine can distinguish between string member types in a structure, and then size the structure and calculate member offsets correctly. String members that need to be stored and passed as Unicode can be declared with the `WSTRING` type.

Note that if a Unicode string is declared with the `STRING` type and the string is used in a structure, the InstallScript engine treats the string as ANSI when passing it to an external DLL. As a result, the size of the structure and member offsets in the structure could be wrong, causing the DLL to incorrectly read or write data related to the structure.

Pointer members in structures can be declared as `WPOINTER`. This enables you to store pointers to Unicode strings in a structure.

## Examples

Like C, InstallScript allows you to nest or embed data structures. For example, suppose you wanted to create a structure that could be used to define the upper-left and lower-right coordinates of a rectangle. Each coordinate consists of two values: an x value and a y value. You could define a structure that consists of four members: the x and y positions of the upper-left corner and the x and y positions of the lower-right corner.

However, since each x and y pair is a logical unit, you might first define a structure called `POINT` that has two members defining a vertical and horizontal position. Then you could define a structure called `RECT` that includes two members of type `POINT`, one to define the upper left coordinate, another to define the lower right coordinate. These two structures are shown below:

```
// Define a point structure.
typedef POINT
begin
    SHORT nX;
    SHORT nY;
end;

// Use nested point structures to define a rectangle structure.
typedef RECT
```



```

begin
    POINT UpperLeft;
    POINT LowerRight;
end;

```

When a structure is to be referenced by a pointer to that structure, you must use the structure pointer operator ( -> ) to address members of the structure. In the example below, a variable of type RECT is declared, a pointer to the structure is declared, and then the address of the RECT variable is assigned to the pointer. Finally, the structure pointer operator is used to initialize each member to 0.

```

RECT Rectangle;
RECT POINTER pRect;

pRect = &Rectangle;
pRect->UpperLeft.nX = 0;
pRect->UpperLeft.nY = 0;
pRect->LowerRight.nX = 0;
pRect->LowerRight.nY = 0;

```

The following script presents a more complete demonstration of structure pointers, nested structures, and structure pointer dereferencing with the structure pointer operator.

```

// Use a structure to define a point.
typedef POINT
begin
    SHORT nX;
    SHORT nY;
end;

// Use nested structures to define a rectangle.
typedef RECT
begin
    POINT UpperLeft;
    POINT LowerRight;
end;

// Declare a rectangle structure variable.
RECT Rectangle;

// Define a pointer to a RECT structure.
RECT POINTER pRect;

// Declare a function to display structure contents.
prototype ShiftRectBy2(RECT POINTER);

. . .

// Get a pointer to the Rectangle structure.
pRect = &Rectangle;

// Define the points that define the rectangle.
pRect->UpperLeft.nX = 100;
pRect->UpperLeft.nY = 400;
pRect->LowerRight.nX = 200;
pRect->LowerRight.nY = 100;

// Display point x and y values before calling ShiftRectBy2.
SprintfBox (INFORMATION,
    "BEFORE calling ShiftRectBy2",

```

```

        "pRect->UpperLeft.nX  = %d\n" +
        "pRect->UpperLeft.nY  = %d\n" +
        "pRect->LowerRight.nX = %d\n" +
        "pRect->LowerRight.nY = %d\n",
        pRect->UpperLeft.nX,
        pRect->UpperLeft.nY,
        pRect->LowerRight.nX,
        pRect->LowerRight.nY
    );

    // Shift the rectangle 2 up and 2 to the right.
    ShiftRectBy2(pRect);

    // Display point x and y values after calling ShiftRectBy2.
    sprintfBox (INFORMATION,
        "AFTER calling ShiftRectBy2",
        "pRect->UpperLeft.nX  = %d\n" +
        "pRect->UpperLeft.nY  = %d\n" +
        "pRect->LowerRight.nX = %d\n" +
        "pRect->LowerRight.nY = %d\n",
        pRect->UpperLeft.nX,
        pRect->UpperLeft.nY,
        pRect->LowerRight.nX,
        pRect->LowerRight.nY
    );

    // Define the rectangle shifting function.
    function ShiftRectBy2(pR)
    begin
        pR->UpperLeft.nX = pR->UpperLeft.nX + 2;
        pR->UpperLeft.nY = pR->UpperLeft.nY + 2;
        pR->LowerRight.nX = pR->LowerRight.nX + 2;
        pR->LowerRight.nY = pR->LowerRight.nY + 2;
    end;

```

## Language IDs

InstallShield provides language constants for all languages supported by Windows. However, most of these constants are not supported for the designation of language-specific components and language filtering.

InstallScript constants can be used in the following contexts.

### Parameter for FeatureFilterLanguage

InstallScript language constants are used as the second parameter to the function [FeatureFilterLanguage](#). In this context, the language constant specifies which files to filter or unfilter. Only supported language constants, which are listed in the supported language table should be used for this purpose. Using an unsupported language constant for component filtering has no effect since components designated for an unsupported language are filtered (not included) during the media build and so cannot be installed.

### Return Value for GetSystemInfo

A language constant serves as the value returned in `nvResult` by function [GetSystemInfo](#) when it is called with the constant `LANGUAGE` in the `nltem` parameter. In this context, any of the language constants that are listed in **ISRTDefs.h** can be returned because Windows supports all the language constants.



**Note** • If your installation includes language filtering based on these return values, you must use a switch statement to convert constants returned by this function into one of the constants supported for language filtering.

## Language Constant Reference

For a complete list of language constants that are supported by InstallShield, as well as their numeric equivalents, refer to the file **ISRTDefs.h**, which is located in the **Script\Include** folder within the InstallShield Program Files folder.

For a list of supported languages and the appropriate InstallScript constants, see Language Support for InstallScript.



**Note** • The language that the setup is using to display prompts and messages is stored in the system variable **SELECTED\_LANGUAGE**.

When the language selection dialog is used by a multilanguage installation to allow the end user to select the installation language during setup initialization, the language dialog will display the Windows equivalent names. Because these names are generated by Windows, they will be localized to the version of Windows under which the installation is being run.

# Pointers

A pointer is a variable that contains the address of another variable. To declare a pointer, use the keyword **POINTER** or **WPOINTER** followed by a variable name, as shown in the following two sample lines of code:

```
POINTER pPointerName;
WPOINTER pwPointerName;
```

To declare a pointer that will be used to access the members of a data structure, precede the keyword **POINTER** or **WPOINTER** with the structure type:

```
typedef RECT
begin
    SHORT sX;
    SHORT sY;
end;

RECT Rectangle;

RECT POINTER pRect;
```

Use the address operator (&) to assign the address of a variable to a pointer variable:

```
pPointerName = &MyStructure;
pNum = &nvNumber;
pString = &svString;
```

When you are defining a function that takes a pointer to a structure as a parameter, use the structure name with **POINTER** or **WPOINTER** in the function prototype, as shown below. Note that any function prototype that specifies a pointer to a structure as one of its parameters must be declared after the structure declaration.

```
typedef RECT
begin
    SHORT sX;
```

```

    SHORT sY;
end;

RECT Rectangle;
RECT POINTER pRect;

prototype SizeRectangle(RECT POINTER);

. . .

pRect = &Rectangle;
SizeRectangle(pRect);

. . .

function SizeRectangle(pRectangle)
begin
    pRectangle->sX = 10;
    pRectangle->sY = 5;
end;

```

## Passing Pointers to Strings to Functions that Are Implemented Outside InstallScript Code

The InstallScript compiler lets you pass pointers to Unicode or ANSI strings to functions that are implemented outside script. For example, if you want to call a DLL function that accepts pointers to strings in its parameters, the DLL function prototype would look something like the following in C or C++:

```
void __stdcall MyDllFunction(LPCSTR pszString);
```

In InstallScript, the function could be prototyped as follows:

```
prototype DLL.MyDllFunction(POINTER);
```

You can use the address-of operator (&) to call the function and pass a pointer to a string:

```
DLL.MyDllFunction(&myString);
```

When the script engine makes this function call, the data in string **myString** is passed through a pointer value to **MyDllFunction**. **MyDllFunction** receives a pointer to an ANSI representation of the string that is contained in **myString**.

The pointer type WPOINTER (or optionally wpointer or LPWSTR) lets you pass pointers to Unicode strings to functions outside of script. For example, if the DLL uses Unicode strings, you could change its prototype in C or C++ to the following:

```
void __stdcall MyDllFunction(LPCWSTR pszString);
```

In InstallScript, the only change that is required to pass a Unicode string pointer to a DLL that uses Unicode strings is the prototype, which would contain the WPOINTER type, as follows:

```
prototype DLL.MyDllFunction(WPOINTER);
```

When the DLL function is called in the running script, the engine passes a pointer to a Unicode copy of the string that is stored in **myString** instead of an ANSI version.

## Using STRING and WSTRING Instead of Pointers

In most cases, you do not need to use pointers to pass strings to external DLL functions. The `STRING` and `WSTRING` types may be used in place of `POINTER` or `WPOINTER`. If a DLL function expects an ANSI string, you can use the `STRING` type; if a DLL function expects a Unicode string, you can use the `WSTRING` type. Using `BYREF` and `BYVAL` will allow for passing a string that either can be modified by the external DLL function or not.

Thus, if you use the following prototype for the function, an ANSI string could be passed by value or by references (modifying the prototype to `BYREF` as needed):

```
prototype DLL.MyDllFunction(byval string);
```

Changing the parameter type to `BYVAL WSTRING` allows a Unicode version of the string to be passed instead of the ANSI version.



# Variable Data

A variable is a named data item whose value can change during program execution.

## Variable Declaration

### Format

A variable must be declared in the following format:

```
data_type VariableName1[, VariableName2 [,...]];
```

### Rules

Variable declaration must follow these rules:

- A variable name can have a maximum of 32 characters.
- When more than one variable name is specified in a single declaration, the names must be separated by commas.
- Each variable declaration must be terminated with a semicolon.




---

**Caution** • The names of InstallScript variables and functions are case sensitive. For example, `svItemCounter` is not equivalent to `svITEMCOUNTER`.

## Variable Declaration Example

In the following example, seven variables are declared. Note that the last declaration creates three numeric variables.

```
BOOL bValidEntry;

LONG lPopulation;

//String explicitly sized
STRING szUserName[128];

//String autosized
STRING szMessage;

NUMBER nFileSize, nDirSize, nDiskSpace;
```

## Declaring String Variables

You can declare a string variable with or without an explicit size. String variables that are not declared with an explicit size are then sized automatically during a setup to accommodate the values assigned to them. Autosizing is recommended for all string variables except those that will be passed to an external (DLL or Windows API) function—in these cases, the string's size *must* be declared explicitly. The maximum size of a string is 65534 characters.

# Global vs. Local Variables

Variables may be either global or local:

- A variable is global if it is declared outside of the main program block and not within a function. Global variables are visible and available to all statements in a setup script that follow its declaration.
- A variable is local if it is declared between the function declaration and the keyword *begin* within that function. Local variables are visible and available only within the function where they are declared.




---

**Project** • *InstallScript events are not used in Basic MSI and merge module projects; therefore, all InstallScript code for these project types must be written in InstallScript custom actions. Global variables do not share state between these custom action invocations.*




---

**Note** • *InstallScript system variables are global and therefore are visible to the main program and to all functions in a script.*

In the following example, the variable `nVisibleEverywhere` can be referenced by any statement in the script. The variable `nVisibleOnlyToFunctions` may be referenced only by the functions. The variable `nVisibleOnlyToSecondFunction` cannot be referenced by the main program or by `FirstFunction`. The variable `szString` is local to `FirstFunction`.

```

prototype FirstFunction();
prototype SecondFunction();

NUMBER nVisibleEverywhere;

. . .

nVisibleEverywhere = 10;

FirstFunction();
SecondFunction();

. . .

NUMBER nVisibleOnlyToFunctions;

function FirstFunction()
    STRING szString;
begin
    szString = "Local to FirstFunction";
    nVisibleOnlyToFunctions = 20;
end;

    NUMBER nVisibleOnlyToSecondFunction;

function SecondFunction()
begin
    nVisibleOnlyToSecondFunction = 30;
end;

```

Although identifiers in a script must be unique, it is valid for a local variable to have a name identical to that of a global variable, or for one function to declare a local variable that has the same name as a local variable declared in another function. These exceptions are allowed because InstallShield qualifies local variable names based on the



function with which they are associated. In the example below, the global variable `szVal` is not affected by the action of `AFunction`, which has a local variable of the same name; the function `MessageBox` displays the string "YES," which was the value assigned to the global variable `szVal`.

```

STRING szVal;

prototype AFunction();

. . .

szVal = "YES";
AFunction();
MessageBox(szVal, INFORMATION);

. . .

function AFunction()
  STRING szVal;
begin
  szVal = "NO";
end;
```

Parameter names in function definitions are considered to be local variables. If a global variable is passed to a function whose parameter has the same name as the global variable, the value of that global variable will not be changed (unless the parameter was specified with the `BYREF` operator in the function prototype). In the following example, `AFunction` has no effect on the global variable `szVal`; the script displays the string "YES".

```

STRING szVal;
prototype AFunction(STRING);

. . .

szVal = "YES";
AFunction(szVal);
MessageBox(szVal, INFORMATION);

. . .

function AFunction(szVal)
begin
  szVal = "NO";
end;
```

## String Variables

For information about string variables, refer to the following topics:

- [String Indexing](#)
- [String Size and Autosize](#)

## String Indexing

A string variable is an array of Unicode characters with a null terminator. You can reference individual characters within a string by specifying the string name followed by an index value within square brackets. Note that the first character in a string is in position 0.

In the example below, the function `BlankLeadingZeros` uses the string indexing technique to replace leading zeros in the string representation of a number with blank characters.

```

        prototype BlankLeadingZeros(BYREF STRING);

function BlankLeadingZeros(szString)
    INT iVal, iLength;
begin
    iVal = 0;
    iLength = StrLength (szString);

    while (szString[iVal] = "0") && (iVal <= iLength)
        szString[iVal] = " ";
        iVal++;
    endwhile;
end;
```

## String Size and Autosize

### InstallShield Autosizing

When you declare a string variable without a size specification, InstallShield automatically sizes a string buffer for that variable. The allocation for the buffer occurs when you first assign a string to the variable. If later, you assign a longer string to that variable, InstallShield increases the memory allocation to accommodate the longer string—up to the amount of available memory. However, if you later assign a shorter string to an autosized variable, InstallShield does not decrease the memory allocation.



**Caution** • *InstallShield's autosizing feature does not work in typedef statements; you must specify the size of all `STRING` declarations in a structure.*

### Specifying a String Size

When specifying a string size, you must declare one character position for the null terminator. For example, if you want your string to hold up to 128 characters, you must declare it with a length of 129 to allow room for the null terminator. For this reason, the minimum size of a string is 2.

When you are using a string that you have declared with a size, you must be aware of how that string might be used with other strings. For example, consider the following function call:

```

        STRING szQuestion[20], szDefault[20], svResult[50];

begin
    szQuestion = "Enter company name";
    szDefault  = "My Software Company";
    AskText (szQuestion, szDefault, svResult);
```

The size of the string `svResult` should be greater than or equal to the size of the string `szDefault`. If not, then `szDefault`, if accepted, will not fit into the `svResult` variable returned by the function. The easiest way to avoid conflicts is to let InstallShield autosize all strings (except those in a typedef statement).




---

**Caution** • An autosized string variable that is passed by reference to a function is not autosized within the called function. If the function attempts to assign a value whose length is greater than the current size of that parameter, run-time error 401 occurs.

## System Variables

System variables are predefined script variables that contain information such as the source path, the target path, the Windows folder, and the Windows system folder. The installation automatically initializes these system variables when the installation process begins, and it is not necessary to declare these variables in your script.




---

**Project** • Many Windows Installer directory properties—such as **INSTALLDIR**, **AppDataFolder**, and **TempFolder**—are available directly as variables in your InstallScript code for Basic MSI and InstallScript MSI projects.

### System Variables and Text Substitutions

Some of the system variables have corresponding text substitutions. The installation internally uses text substitution to set the values of certain system variables as shown in the following tables. You can use these text substitutions in your script in the same way that you use text substitutions that you have defined.

## Writable System Variables and Text Substitutions

**Table 1 •** Writable System Variables and Text Substitutions

Script Variable	Corresponding Text Substitution	Comments
<b>ALLUSERS</b>	<PERUSER_INSTALL>	
<b>DISK1TARGET</b>	<DISK1TARGET>	
<b>IFX_COMPANY_NAME</b>	<IFX_COMPANY_NAME>	<p>If this text substitution is defined in an object script, it applies to only that object, not to the main installation or any other objects in the installation.</p> <p>If it is defined in the main installation, it does not apply to any objects.</p>
<b>IFX_INSTALLED_DISPLAY_VERSION</b>	<IFX_INSTALLED_DISPLAY_VERSION>	<p>If this text substitution is defined in an object script, it applies to only that object, not to the main installation or any other objects in the installation.</p> <p>If it is defined in the main installation, it does not apply to any objects.</p>
<b>IFX_INSTALLED_VERSION</b>	<IFX_INSTALLED_VERSION>	<p>If this text substitution is defined in an object script, it applies to only that object, not to the main installation or any other objects in the installation.</p> <p>If it is defined in the main installation, it does not apply to any objects.</p>
<b>IFX_MULTI_INSTANCE_SUFFIX</b>	<IFX_MULTI_INSTANCE_SUFFIX>	
<b>IFX_PRODUCT_DISPLAY_NAME</b>	<IFX_PRODUCT_DISPLAY_NAME>	<p>If this text substitution is defined in an object script, it applies to only that object, not to the main installation or any other objects in the installation.</p> <p>If it is defined in the main installation, it does not apply to any objects.</p>

**Table 1 •** Writable System Variables and Text Substitutions (cont.)

Script Variable	Corresponding Text Substitution	Comments
<b>IFX_PRODUCT_DISPLAY_VERSION</b>	<IFX_PRODUCT_DISPLAY_VERSION>	<p>If this text substitution is defined in an object script, it applies to only that object, not to the main installation or any other objects in the installation.</p> <p>If it is defined in the main installation, it does not apply to any objects.</p>
<b>IFX_PRODUCT_KEY</b>	<IFX_PRODUCT_KEY>	<p>If this text substitution is defined in an object script, it applies to only that object, not to the main installation or any other objects in the installation.</p> <p>If it is defined in the main installation, it does not apply to any objects.</p>
<b>IFX_PRODUCT_NAME</b>	<IFX_PRODUCT_NAME>	<p>If this text substitution is defined in an object script, it applies to only that object, not to the main installation or any other objects in the installation.</p> <p>If it is defined in the main installation, it does not apply to any objects.</p>
<b>IFX_PRODUCT_VERSION</b>	<IFX_PRODUCT_VERSION>	<p>If this text substitution is defined in an object script, it applies to only that object, not to the main installation or any other objects in the installation.</p> <p>If it is defined in the main installation, it does not apply to any objects.</p>
<b>IFX_SETUP_TITLE</b>	<IFX_SETUP_TITLE>	<p>If this text substitution is defined in an object script, it applies to only that object, not to the main installation or any other objects in the installation.</p> <p>If it is defined in the main installation, it does not apply to any objects.</p>

**Table 1 •** Writable System Variables and Text Substitutions (cont.)

Script Variable	Corresponding Text Substitution	Comments
<b>IFX_SUPPORTED_VERSIONS</b>	<IFX_SUPPORTED_VERSIONS>	<p>If this text substitution is defined in an object script, it applies to only that object, not to the main installation or any other objects in the installation.</p> <p>If it is defined in the main installation, it does not apply to any objects.</p>
<b>SHELL_OBJECT_FOLDER</b>	<SHELL_OBJECT_FOLDER>	
<b>SRCDIR</b> (Read from local)	<SRCDIR>	
<b>SRCDISK</b> (Read from local)	<SRCDISK>	
<b>TARGETDIR</b>	<TARGETDIR>	
<b>TARGETDISK</b>	<TARGETDISK>	
<b>UNINST</b>	<UNINST>	
<b>UNINSTALL_STRING</b>	<UNINSTALL_STRING>	

## Read-Only System Variables and Text Substitutions

**Table 2 •** Read-Only System Variables and Text Substitutions

Script Variable	Corresponding Text Substitution	Comments
<b>COMMONFILES</b>	<COMMONFILES>	
<b>DISK1SETUPEXENAME</b>	<DISK1SETUPEXENAME>	
<b>ENGINECOMMONDIR</b>	<ENGINECOMMONDIR>	
<b>ENGINEDIR</b>	<ENGINEDIR>	
<b>FOLDER_APPDATA</b>	<FOLDER_APPDATA>	
<b>FOLDER_DOTNET_10</b>	<FOLDER_DOTNET_10>	
<b>FOLDER_DOTNET_11</b>	<FOLDER_DOTNET_11>	
<b>FOLDER_DOTNET_20</b>	<FOLDER_DOTNET_20>	
<b>FOLDER_DOTNET_30</b>	<FOLDER_DOTNET_30>	
<b>FOLDER_DOTNET_35</b>	<FOLDER_DOTNET_35>	
<b>FOLDER_DOTNET_40</b>	<FOLDER_DOTNET_40>	
<b>FOLDER_PERSONAL</b>	<PERSONALDIR>	
<b>FOLDER_TEMP</b>	<FOLDER_TEMP>	
<b>ISRES</b>	<ISRES>	<p>If this text substitution is defined in an object script, it applies to only that object, not to the main installation or any other objects in the installation.</p> <p>If it is defined in the main installation, it does not apply to any objects.</p>
<b>ISUSER</b>	<ISUSER>	<p>If this text substitution is defined in an object script, it applies to only that object, not to the main installation or any other objects in the installation.</p> <p>If it is defined in the main installation, it does not apply to any objects.</p>

**Table 2 •** Read-Only System Variables and Text Substitutions (cont.)

Script Variable	Corresponding Text Substitution	Comments
<b>MULTI_INSTANCE_COUNT</b>	<MULTI_INSTANCE_COUNT>	
<b>PACKAGE_LOCATION</b>	<PACKAGE_LOCATION>	
<b>PROGRAMFILES</b>	<PROGRAMFILES>	
<b>SELECTED_LANGUAGE</b>	<SELECTED_LANGUAGE>	
<b>SHAREDSUPPORTDIR</b>	<SHOW_PASSWORD_DIALOG>	
<b>SHOW_PASSWORD_DIALOG</b>	<SHOW_PASSWORD_DIALOG>	
<b>SUPPORTDIR</b>	<SUPPORTDIR>	
<b>WINDIR</b>	<WINDIR>	
<b>WINDISK</b>	<WINDISK>	
<b>WINSYSDIR</b>	<WINSYSDIR>	
<b>WINSYSDISK</b>	<WINSYSDISK>	

## ADDREMOVE



**Project •** This information applies to InstallScript projects.

The ADDREMOVE system variable is set equal to a non-zero value if the setup is run from Add or Remove Programs in the Control Panel and is set equal to FALSE otherwise. This system variable is read-only; if you attempt to assign a value to it, a compiler error results.

## ADDREMOVE\_COMBINEDBUTTON



**Project •** This information applies to InstallScript projects.

This system variable's value is used by the MaintenanceStart function to specify the existence of and data in the application uninstallation registry key's ModifyPath and UninstallString values, so as to specify whether the application's Add or Remove Programs entry displays separate Change and Remove buttons or a combined Change/Remove button. For more information on this variable, see [MaintenanceStart](#).

This system variable is initialized to FALSE.



## ADDREMOVE\_HIDECHANGEOPTION



**Project** • This information applies to InstallScript projects.

This system variable's value is used by the [MaintenanceStart](#) function to specify the data in the application uninstallation registry key's ModifyPath, NoModify, and UninstallString values. The NoModify registry value specifies whether a Change button is displayed for the application in Add or Remove Programs in the Control Panel. The ModifyPath and UninstallString registry values specify the behaviors of the Change and Remove buttons.

This system variable is initialized based on the value that you specify in the Disable Change Button setting in the General Information view.

## ADDREMOVE\_HIDEREMOVEOPTION



**Project** • This information applies to InstallScript projects.

This system variable's value is used by the [MaintenanceStart](#) function to specify the data in the application uninstallation registry key's ModifyPath, NoRemove, and UninstallString values. The NoRemove registry value specifies whether a Remove button is displayed for the application in Add or Remove Programs in the Control Panel. The ModifyPath and UninstallString registry values specify the behaviors of the Change and Remove buttons.

This system variable is initialized based on the value that you specify in the Disable Remove Button setting in the General Information view.

## ADDREMOVE\_STRING\_REMOVEONLY



**Project** • This information applies to InstallScript projects.

This system variable's value is used by the MaintenanceStart function to specify the data in the application uninstallation registry key's UninstallString value, which specifies the behavior of the Remove button (if any) in the application's Add or Remove Programs entry. For more information on this variable, see [MaintenanceStart](#).

This system variable is initialized to the string value "-removeonly".

## ADDREMOVE\_SYSTEMCOMPONENT



**Project** • This information applies to InstallScript projects.

This system variable's value is used by the [MaintenanceStart](#) function to specify the data in the application uninstallation registry key's SystemComponent value, which specifies whether an entry is displayed for the application in Add or Remove Programs in the Control Panel.

This system variable is initialized to FALSE, meaning that an entry is displayed.

## ALLUSERS

The ALLUSERS system variable is the key to installations that allow installing the application to the current user or all users on the target system. The value of ALLUSERS determines the following:

- The root key that is used by registry functions that are called after RegDBSetDefaultRoot(HKEY\_USER\_SELECTABLE) is called
- The root key under which the application uninstallation key is created
- The location of registry entries that are specified in a registry set's HKEY\_USER\_SELECTABLE root key
- The value of the [DISK1TARGET](#) system variable (which specifies the path to the folder in which copies of certain installation files are placed to enable maintenance installations and uninstallation)
- The default value of the [TARGETDIR](#) system variable that is set by the default OnFirstUIBefore code
- Whether a shortcut that is defined in InstallShield with its Type property set to Automatic appears in the list of personal shortcuts or common shortcuts when the shortcut is created
- The default option selection in the [SdCustomerInformation](#) and [SdCustomerInformationEx](#) dialogs

ALLUSERS has no effect on the registration of COM DLL files.

The following sections explain how the value of ALLUSERS is determined and set in different project types.

### InstallScript Installations

If an installation is running as a first-time installation, the InstallScript engine determines during initialization the most appropriate value for the ALLUSERS variable and initializes it to that value:

- If the user does not have administrator privileges, ALLUSERS is set to 0. This results in a per-user installation.
- Otherwise, ALLUSERS is set to 1. This results in a per-machine installation by default.

If an installation is running in maintenance mode, the InstallScript engine determines the value of the ALLUSERS variable based on whether the initial installation was installed as per-user or per-machine (based on the location of where the uninstallation information was installed).

For an example of an InstallScript installation that allows installing the application to the current user or all users on the target system, see the sample project in the ALLUSERS Sample Project folder. This folder is a subfolder in the Samples folder of your InstallShield Program Files folder. The default location is:

**C:\Program Files\InstallShield\2016\Samples\InstallScript\ALLUSERS Sample Project**

## InstallScript Custom Actions in Basic MSI and InstallScript MSI Installations

### Getting the Value of the ALLUSERS Variable

The ALLUSERS InstallScript variable is determined by querying the Windows Installer property **ALLUSERS**:

**Table 3 •** Getting the Value of the ALLUSERS InstallScript Variable

Windows Installer Property	InstallScript Variable	Notes
""	0	Any per-user or per-machine dependent script code in the custom action results in per-user behavior.
1	1	Any per-user or per-machine dependent script code in the custom action results in per-machine behavior.
Any other value	The InstallScript engine attempts to determine the appropriate value.	

If the Windows Installer property **ALLUSERS** cannot be determined because the InstallScript engine determines that a deferred custom action is running, an unexpected property value is returned, or **MsiGetProperty** returns an error value. Therefore, the InstallScript engine attempts to determine the best value for the variable.

Note that the InstallScript engine uses **MsiGetMode** with the MSIRUNMODE\_SCHEDULED, MSIRUNMODE\_ROLLBACK, and MSIRUNMODE\_COMMIT flags to determine if a deferred custom action is running. If **MsiGetMode** returns true for any of the above values, the custom action is assumed to be deferred and the InstallScript mechanism is used.

Note that if a Basic MSI installation does not have a value for **ALLUSERS** in the **Property** table, any InstallScript custom action that runs before the installation displays an **ALLUSERS** dialog (such as the CustomerInformation dialog, which sets the **ALLUSERS** Windows Installer property) has the ALLUSERS InstallScript variable set to 0. Therefore, the InstallScript custom action exhibits per-user behavior. Thus, it is recommended that all Basic MSI installations have a default value for **ALLUSERS** in the **Property** table.

### Setting the Value of the ALLUSERS Variable

When the ALLUSERS InstallScript variable is set in script, the InstallScript engine first determines whether the platform and privilege level allow the ALLUSERS InstallScript variable to be changed. (In scenarios where the end user is not an administrator or power user, ALLUSERS cannot be changed.)

If the ALLUSERS InstallScript variable can be changed, the InstallScript engine attempts to update the **ALLUSERS** Windows Installer property appropriately as follows:

**Table 4 •** Setting the Value of the ALLUSERS InstallScript Variable

InstallScript Variable	Windows Installer Property
1	1

**Table 4 •** Setting the Value of the ALLUSERS InstallScript Variable (cont.)

InstallScript Variable	Windows Installer Property
0	""

Note that the InstallScript engine sets the ALLUSERS InstallScript variable even if the Windows Installer property cannot be set. This could result in synchronization problems between the Windows Installer property and the InstallScript variable. Therefore, if you change the ALLUSERS InstallScript variable in a custom action, it is recommended that you set the Windows Installer property manually as well to ensure that the property can be changed successfully.

The following table shows the behavior for various scenarios on Windows Vista with User Account Control (UAC) enabled.

**Table 5 •** ALLUSERS Value on Windows Vista with UAC Enabled

Type of Custom Action	Manifest	ALLUSERS Value in Property Table	Windows Installer Property	Resulting InstallScript Variable	Notes
<b>Immediate</b>	Highest available	2	1	1	
<b>Immediate</b>	Invoker	2	""	0	The InstallScript variable cannot be changed. Per-machine InstallScript actions are not possible.
<b>Immediate</b>	Highest available	1	1	1	
<b>Immediate</b>	Invoker	1	1	1	The InstallScript variable cannot be changed. Per-machine InstallScript actions fail.
<b>Immediate, before CustomerInformation dialog (or end user selects to install per user)</b>	Invoker	""	""	0	The InstallScript variable cannot be changed. Per-machine InstallScript actions are not possible.
<b>Immediate, after CustomerInformation dialog</b>	Highest available	""	1	1	

**Table 5 • ALLUSERS Value on Windows Vista with UAC Enabled (cont.)**

Type of Custom Action	Manifest	ALLUSERS Value in Property Table	Windows Installer Property	Resulting InstallScript Variable	Notes
<b>Deferred</b>	Invoker	Any	Cannot be determined	0	The InstallScript variable cannot be changed. InstallScript method for determining ALLUSERS is used.
<b>Deferred</b>	Highest available	Any	Cannot be determined	1	InstallScript method for determining ALLUSERS is used.

The following table shows the behavior for various scenarios on pre-Windows Vista systems and on Windows Vista systems with User Account Control (UAC) disabled.

**Table 6 • ALLUSERS Value on Systems Earlier than Windows Vista and on Windows Vista with UAC Disabled**

Type of Custom Action	User Privileges	ALLUSERS Value in Property Table	Windows Installer Property	Resulting InstallScript Variable	Notes
<b>Immediate</b>	Administrator	2	1	1	
<b>Immediate</b>	Limited	2	""	0	The InstallScript variable cannot be changed.
<b>Immediate</b>	Administrator	1	1	1	
<b>Immediate</b>	Limited	1	1	1	The InstallScript variable cannot be changed. The installation fails during UI sequence, and per-machine InstallScript custom actions fail.

**Table 6 • ALLUSERS Value on Systems Earlier than Windows Vista and on Windows Vista with UAC Disabled (cont.)**

Type of Custom Action	User Privileges	ALLUSERS Value in Property Table	Windows Installer Property	Resulting InstallScript Variable	Notes
<b>Immediate, before CustomerInformation dialog (or end user selects to install per user)</b>	Administrator	""	""	0	The InstallScript variable cannot be changed. Per-machine InstallScript actions are not possible.
<b>Immediate, after CustomerInformation dialog</b>	Administrator	""	1	1	
<b>Deferred</b>	Administrator	Any	Cannot be determined	1	InstallScript method for determining ALLUSERS is used.
<b>Deferred</b>	Limited	Any	Cannot be determined	0	The InstallScript variable cannot be changed. InstallScript method for determining ALLUSERS is used.
<b>Any on Windows 9x</b>	Not applicable	Any	Any	1	The InstallScript variable cannot be changed. InstallScript method for determining ALLUSERS is used.

If the resulting InstallScript variable is 0, InstallScript custom actions are used in a per-user context. If the resulting InstallScript variable is 1, InstallScript custom actions are used in a per-machine context.

### Event-Driven InstallScript Code in InstallScript MSI Installations

InstallScript MSI installations work similarly to InstallScript installations except that when the ALLUSERS InstallScript variable is changed, the installation attempts to update the Windows Installer property **ALLUSERS** as described for InstallScript custom actions.

During InstallScript MSI installations, the Windows Installer property **ALLUSERS** is not queried to determine the appropriate value of the ALLUSERS InstallScript variable; the InstallScript engine always attempts to determine the value, as described for InstallScript installations.

## How an InstallScript Installation Works by Default Depending on ALLUSERS



**Project** • This information applies to InstallScript projects.

The following table provides information on how an installation is installed based on the ALLUSERS system variable.

**Table 7 • ALLUSERS**

	Admin & Power User	User and Guest
<b>Default setting of ALLUSERS</b>	True	False
<b>Automatic setting will check ALLUSERS property</b>	Per machine	Per user
<b>HKEY_USER_SELECTABLE_AUTO</b>	HKEY_LOCAL_MACHINE	HKEY_CURRENT_USER
<b>Registry Set Data</b>	HKEY_LOCAL_MACHINE	HKEY_CURRENT_USER
<b>DISK1TARGET</b>	Program Files\InstallShield Installation Information\GUID	My Docs\InstallShield Installation Information\GUID
<b>Uninstall Registry Key</b>	HKEY_LOCAL_MACHINE\...\Uninstall\GUID	HKEY_CURRENT_USER\...\Uninstall\GUID
<b>TARGETDIR</b>	Program Files\Company Name\Product Name	My Docs\Company Name\Product Name
<b>Engine Installation</b>	Program Files\Common Files	My Docs\...
<b>Register COM Information (.dll, .ocx, .exe)</b>	Supported	Not supported



**Note** • My Docs refers to a destination location to which the user has rights. This value depends on the operating system.

## ADMINUSER



**Project** • This information applies to the following project types:

- Basic MSI projects with InstallScript custom actions
- InstallScript MSI projects with InstallScript custom actions

This information does not apply to InstallScript projects or to event-driven InstallScript code in InstallScript MSI projects.

The ADMINUSER system variable is set to the value of the Windows Installer property AdminUser.

## BATCH\_INSTALL



**Project** • This information applies to the following project types:

- InstallScript
- InstallScript MSI—if the InstallScript user interface (UI) style is the traditional style (which uses the InstallScript engine as an external UI handler)

*This information does not apply to InstallScript MSI projects in which the InstallScript UI style is the new style (which uses the InstallScript engine as an embedded UI handler). To learn more, see [Using the InstallScript Engine as an External vs. Embedded UI Handler for InstallScript MSI Installations](#).*

The BATCH\_INSTALL system variable is set to a non-zero value to indicate that one or more operations need to be performed after the target system restarts. BATCH\_INSTALL can be set to a non-zero value for any of the following reasons:

- The installation determines that a file cannot be installed because the file already exists on the target system and it is locked.
- BATCH\_INSTALL is set to non-zero manually through script. Note that this can occur in some objects if the object determines that an embedded installation needs a reboot to complete.
- LAAW\_OPTION\_SET\_BATCH\_INSTALL is used when calling **LaunchApplication**, and the function determines that the launched installation needs a reboot to complete.
- The installation attempted to update a Windows service (through **ServiceAddService** and the related functions), but it could not stop the existing service.
- The DIFx integration indicated that a reboot is needed, due to the installation of a DIFx driver.

If BATCH\_INSTALL is set to FALSE, no locked files were found and the installation process can end normally.

For more information, see [Understanding When an Installation or Uninstallation Restarts the Target System](#).

## CMDLINE

The CMDLINE variable varies, depending on the project type.

### CMDLINE in InstallScript Projects

In InstallScript projects, **Setup.exe** accepts all user-defined command-line arguments and assigns them to the system string variable CMDLINE at run time.



**Note** • Note the following about CMDLINE in InstallScript projects:

- CMDLINE stores only user-defined command-line arguments. InstallShield command-line arguments (predefined arguments) are not copied to CMDLINE.



- The installation converts all characters in user-defined command-line arguments to lowercase when copying them to CMDLINE. Use case insensitive logic when processing CMDLINE.

## CMDLINE in InstallScript Installations That Are Included in Advanced UI and Suite/Advanced UI Projects as InstallScript Packages

Data can be passed to an InstallScript package through the package's Command Line and Silent Command Line settings in the Advanced UI or Suite/Advanced UI project that contains the InstallScript package. The data can then be parsed from the CMDLINE variable in the Advanced UI or Suite/Advanced UI script events. For more information, see [OnSuiteShowUI](#).

## CMDLINE in InstallScript MSI Projects

In InstallScript MSI projects, any command-line data that are passed to **Setup.exe** with the /z switch are stored in the system string variable CMDLINE. For example, if the user runs the following command line, CMDLINE is set to the string **My custom data**.

```
Setup.exe /z"My custom data"
```

## CMDLINE in Basic MSI Projects

In Basic MSI projects, you can pass public properties through **Setup.exe** to **Msiexec.exe** using the /v command-line argument.

# COMMONFILES

The COMMONFILES system variable contains the fully qualified name of the folder that is defined by Windows and that stores files shared by applications that are installed on the system. In English Windows, that folder is named Common Files, and it is located in the Program Files folder. (In other language versions of Windows, the common files folder name is localized appropriately by default.) The common files folder is the recommended default location for files and folders that are shared by applications.

On 64-bit Windows systems, this folder stores common files for 32-bit applications; common files for 64-bit applications should be installed to the COMMONFILES64 folder.



**Project** • During setup initialization in InstallScript installations, the value of the COMMONFILES variable is obtained by calling the Windows API function **SHGetSpecialFolderPath** with the CSIDL\_COMMON\_FILES parameter.

In Basic MSI and InstallScript MSI installations, the value of the COMMONFILES variable is initialized based on the Windows Installer property **CommonFilesFolder**. Note that deferred, commit, and rollback custom actions do not have access to this property. Therefore, the corresponding COMMONFILES variable is empty in deferred, commit, and rollback custom actions. To learn more, see *Accessing or Setting Windows Installer Properties Through Deferred, Commit, and Rollback Custom Actions*.

## COMMONFILES64

The COMMONFILES64 system variable contains the fully qualified name of the folder that is defined by Windows and that stores files shared by 64-bit applications. In English Windows, that folder is named Common Files, and it is located in the PROGRAMFILES64 folder. (In other language versions of Windows, the common files folder name is localized appropriately by default.) The common files folder is the recommended default location for files and folders that are shared by applications.



**Project** • During setup initialization in InstallScript installations, the value of the COMMONFILES64 variable is obtained by calling the Windows API function **SHGetSpecialFolderPath** with the CSIDL\_COMMON\_FILES parameter from a 64-bit executable file.

In Basic MSI and InstallScript MSI installations, the value of the COMMONFILES64 variable is initialized based on the Windows Installer property **CommonFiles64Folder**. Note that deferred, commit, and rollback custom actions do not have access to this property. Therefore, the corresponding COMMONFILES64 variable is empty in deferred, commit, and rollback custom actions. To learn more, see *Accessing or Setting Windows Installer Properties Through Deferred, Commit, and Rollback Custom Actions*.

## DISK1SETUPEXENAME

DISK1SETUPEXENAME is a read-only system variable that contains the file name and file name extension—but not the path—of the setup launcher, the installation's executable file. The default value is *Setup.exe*.



**Project** • DISK1SETUPEXENAME is intended for use in InstallScript projects only. Using this variable with other project types may result in different behavior or scenarios where the variable is not set as expected.

## DISK1TARGET

This system variable contains the path to the folder in which copies of certain of the installation's files (such as the compiled script file) are placed to enable maintenance installations and uninstallation.

## ENABLED\_SERVICES



**Project** • This information applies to InstallScript projects.

This system variable contains a set of bit flags indicating which InstallShield services are currently enabled. For example, if the expression **ENABLED\_SERVICES & SERVICE\_FLAG\_ISFONTREG** is equal to a non-zero value, global font registration is currently enabled.

## ENGINECOMMONDIR



**Project** • This information applies to InstallScript projects.

The ENGINECOMMONDIR system variable stores the fully qualified path to the folder that contains the run-time files that are used by all 6.x, 7.x, and 9.x InstallScript (not InstallScript MSI) setups running on the system.

The value of this system variable is shared among object scripts and between object scripts and the main setup script. This system variable is read-only; if you attempt to assign a value to it, a compiler error results.

## ENGINEDIR



**Project** • This information applies to InstallScript projects.

The ENGEDIR system variable stores the fully qualified path to the folder that contains the run-time files specific to the version of the engine being used by the setup, i.e., 7.00, 7.01, or 9.00, but not by InstallShield Professional 6.x setups.

The value of this system variable is shared among object scripts and between object scripts and the main setup script. This system variable is read-only; if you attempt to assign a value to it, a compiler error results.

## ERRORFILENAME

This system variable stores the name of the file that was involved in an error. For example, if an error occurs while copying a specific file with a built-in function, InstallShield sets ERRORFILENAME to the name of the file that caused the error. Not all file-operation functions use ERRORFILENAME.

## FOLDER\_APPDATA

The FOLDER\_APPDATA system variable stores the fully qualified path to the folder that is defined by the operating system and that serves as a common repository for application-specific data.

This system variable is read-only; if you attempt to assign a value to it, a compiler error results. The value of this system variable is shared among object scripts and between object scripts and the main setup script.



**Project** • During setup initialization in InstallScript installations, the value of the FOLDER\_APPDATA variable is obtained by calling the Windows API function **SHGetSpecialFolderPath** with the CSIDL\_APPDATA value for LPITEMIDLIST.

In Basic MSI and InstallScript MSI installations, the value of the FOLDER\_APPDATA variable is initialized based on the Windows Installer property **AppDataFolder** or **LocalAppDataFolder**. Note that deferred, commit, and rollback custom actions do not have access to these properties. Therefore, the corresponding FOLDER\_APPDATA variable is empty in deferred, commit, and rollback custom actions. To learn more, see *Accessing or Setting Windows Installer Properties Through Deferred, Commit, and Rollback Custom Actions*.

## FOLDER\_APPLICATIONS

The FOLDER\_APPLICATIONS system variable stores the fully qualified path to the root folder for application folders. The value of this system variable is equal to the value of the system variable [PROGRAMFILES](#) when the value of the system variable [ALLUSERS](#) is non-zero; when ALLUSERS is FALSE, the value of this system variable is equal to the value of the system variable [FOLDER\\_APPDATA](#).

This system variable is read-only; if you attempt to assign a value to it, a compiler error results. The value of this system variable is shared among object scripts and between object scripts and the main setup script.

## FOLDER\_APPLICATIONS64

The FOLDER\_APPLICATIONS64 system variable stores the fully qualified path to the root folder for application folders on a 64-bit system. The value of this system variable is equal to the value of the system variable [PROGRAMFILES64](#) when the value of the system variable [ALLUSERS](#) is non-zero; when ALLUSERS is FALSE, the value of this system variable is equal to the value of the system variable [FOLDER\\_APPDATA](#).

This system variable is read-only; if you attempt to assign a value to it, a compiler error results. The value of this system variable is shared among object scripts and between object scripts and the main setup script.

## FOLDER\_COMMON\_APPDATA

The FOLDER\_COMMON\_APPDATA system variable stores the fully qualified path to the folder that is defined by the operating system and that serves as a common repository for application-specific data.

This system variable is read-only; if you attempt to assign a value to it, a compiler error results. The value of this system variable is shared among object scripts and between object scripts and the main setup script.




---

**Project** • During setup initialization in InstallScript installations, the value of the FOLDER\_COMMON\_APPDATA variable is obtained by calling the Windows API function **SHGetSpecialFolderPath** with the CSIDL\_COMMON\_APPDATA value for LPITEMIDLIST.

In Basic MSI and InstallScript MSI installations, the value of the FOLDER\_COMMON\_APPDATA variable is initialized based on the Windows Installer property **CommonAppDataFolder**. Note that deferred, commit, and rollback custom actions do not have access to this property. Therefore, the corresponding FOLDER\_COMMON\_APPDATA variable is empty in deferred, commit, and rollback custom actions. To learn more, see *Accessing or Setting Windows Installer Properties Through Deferred, Commit, and Rollback Custom Actions*.

## FOLDER\_DESKTOP

The FOLDER\_DESKTOP system variable stores the fully qualified path to the Desktop folder, which holds the program folders and items that are displayed on the end user's desktop.

To ensure that groups and folders are created at the proper location, the location to which FOLDER\_DESKTOP points changes when the default group or folder type is changed from Common to Personal or from Personal to Common when the system variable ALLUSERS is changed.

## FOLDER\_DOTNET\_10

The FOLDER\_DOTNET\_10 system variable stores the fully qualified path of the folder where the Microsoft .NET Framework 1.0 redistributable files are located:

**<WINDIR>\Microsoft.NET\Framework\v1.0.3705\**

This system variable is read-only. If you attempt to assign a value to it, a compiler error results.

## FOLDER\_DOTNET\_11

The FOLDER\_DOTNET\_11 system variable stores the fully qualified path of the folder where the Microsoft .NET Framework 1.1 redistributable files are located:

**<WINDIR>\Microsoft.NET\Framework\v1.1.4322\**

This system variable is read-only. If you attempt to assign a value to it, a compiler error results.

## FOLDER\_DOTNET\_20

The FOLDER\_DOTNET\_20 system variable stores the fully qualified path of the folder where the Microsoft .NET Framework 2.0 redistributable files are located:

**<WINDIR>\Microsoft.NET\Framework\v2.0.50727\**

This system variable is read-only. If you attempt to assign a value to it, a compiler error results.

## FOLDER\_DOTNET\_30

The FOLDER\_DOTNET\_30 system variable stores the fully qualified path of the folder where the Microsoft .NET Framework 3.0 redistributable files are located:

**<WINDIR>\Microsoft.NET\Framework\v3.0**

This system variable is read-only. If you attempt to assign a value to it, a compiler error results.

## FOLDER\_DOTNET\_35

The FOLDER\_DOTNET\_35 system variable stores the fully qualified path of the folder where the Microsoft .NET Framework 3.5 redistributable files are located:

**<WINDIR>\Microsoft.NET\Framework\v3.5**

This system variable is read-only. If you attempt to assign a value to it, a compiler error results.

## FOLDER\_DOTNET\_40

The FOLDER\_DOTNET\_40 system variable stores the fully qualified path of the folder where the Microsoft .NET Framework 4.0 redistributable files are located:

**<WINDIR>\Microsoft.NET\Framework\v4.0.30319**

This system variable is read-only. If you attempt to assign a value to it, a compiler error results.

## FOLDER\_FONTS



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript Object*

The FOLDER\_FONTS system variable stores the fully qualified path of the Windows font folder.

This system variable is read-only; if you attempt to assign a value to it, a compiler error results. The value of this system variable is shared among object scripts and between object scripts and the main setup script.

## FOLDER\_LOCAL\_APPDATA

The FOLDER\_LOCAL\_APPDATA system variable stores the fully qualified path to the folder that is defined by the operating system and that serves as a common repository for application-specific data. Common values are **C:\Users\<User>\Application Data** on Windows Vista and later and **C:\Documents and Settings\<User>\Application Data** on earlier systems.

This system variable is read-only; if you attempt to assign a value to it, a compiler error results. The value of this system variable is shared among object scripts and between object scripts and the main setup script.



**Project** • During setup initialization in InstallScript installations, the value of the FOLDER\_LOCAL\_APPDATA variable is obtained by calling the Windows API function **SHGetSpecialFolderPath** with the **CSIDL\_LOCAL\_APPDATA** value for **LPITEMIDLIST**.

In Basic MSI and InstallScript MSI installations, the value of the FOLDER\_LOCAL\_APPDATA variable is initialized based on the Windows Installer property **LocalAppDataFolder**. Note that deferred, commit, and rollback custom actions do not have access to this property. Therefore, the corresponding FOLDER\_LOCAL\_APPDATA variable is empty in deferred, commit, and rollback custom actions. To learn more, see *Accessing or Setting Windows Installer Properties Through Deferred, Commit, and Rollback Custom Actions*.

## FOLDER\_PERSONAL



**Project** • This information applies to InstallScript projects.

The FOLDER\_PERSONAL system variable stores the fully qualified path to the folder that is defined by the operating system and that serves as a common repository for application-specific data. Common values are **C:\Users\<User>\Application Data** on Windows Vista and later and **C:\Documents and Settings\<User>\Application Data** on earlier systems.

This system variable is read-only; if you attempt to assign a value to it, a compiler error results. The value of this system variable is shared among object scripts and between object scripts and the main setup script.



**Project** • During setup initialization in InstallScript installations, the value of the `FOLDER_PERSONAL` variable is obtained by calling the Windows API function **SHGetSpecialFolderPath** with the `CSIDL_PERSONAL` value for `LPITEMIDLIST`.

In Basic MSI and InstallScript MSI installations, the value of the `FOLDER_PERSONAL` variable is initialized based on the Windows Installer property **PersonalFolder**. Note that deferred, commit, and rollback custom actions do not have access to this property. Therefore, the corresponding `FOLDER_PERSONAL` variable is empty in deferred, commit, and rollback custom actions. To learn more, see *Accessing or Setting Windows Installer Properties Through Deferred, Commit, and Rollback Custom Actions*.

## FOLDER\_PROGRAMS

The `FOLDER_PROGRAMS` system variable stores the fully qualified path to the Start Menu\Programs folder, which is displayed when you select Programs from the Start Menu.

To ensure that groups and folders are created at the proper location, the location to which `FOLDER_PROGRAMS` points changes when the default group or folder type is changed from Common to Personal or from Personal to Common when the system variable `ALLUSERS` is changed.

## FOLDER\_STARTMENU

The `FOLDER_STARTMENU` system variable stores the fully qualified path to the Start Menu folder, which is displayed when you click the Windows Start button.

To ensure that groups and folders are created at the proper location, the location to which `FOLDER_STARTMENU` points changes when the default group or folder type is changed from Common to Personal or from Personal to Common when the system variable `ALLUSERS` is changed.

## FOLDER\_STARTUP

The `FOLDER_STARTUP` system variable stores the fully qualified path to the Startup folder, which contains the program folders and items that are launched when Windows starts.

To ensure that groups and folders are created at the proper location, the location to which `FOLDER_STARTUP` points changes when the default group or folder type is changed from Common to Personal or from Personal to Common when the system variable `ALLUSERS` is changed.

## FOLDER\_TEMP



**Project** • This information applies to InstallScript projects.

The `FOLDER_TEMP` system variable stores the fully qualified path of the folder designated for temporary files. This folder is used by Windows and most applications on the system and is not created or deleted by the installation. (The folder whose path is stored in the system variable `SUPPORTDIR` is created by the installation to store installation-specific files and is deleted after the installation completes.)

This system variable is read-only; if you attempt to assign a value to it, a compiler error results. The value of this system variable is shared among object scripts and between object scripts and the main setup script.



**Project** • During setup initialization in InstallScript installations, the value of the `FOLDER_TEMP` variable is obtained by calling the Windows API function **GetTempPath**.

In Basic MSI and InstallScript MSI installations, the value of the `FOLDER_TEMP` variable is initialized based on the Windows Installer property **TempFolder**. Note that deferred, commit, and rollback custom actions do not have access to this property. Therefore, the corresponding `FOLDER_TEMP` variable is empty in deferred, commit, and rollback custom actions. To learn more, see *Accessing or Setting Windows Installer Properties Through Deferred, Commit, and Rollback Custom Actions*.

## HKEYCURRENTROOTKEY

The value of this system variable is the root key that is used by the general registry-related functions. The system variable's possible values are the following:

- `HKEY_CLASSES_ROOT`
- `HKEY_CURRENT_USER`
- `HKEY_LOCAL_MACHINE`
- `HKEY_USERS`
- `HKEY_CURRENT_CONFIG`
- `HKEY_DYN_DATA`

You can set the default root key by setting `HKEYCURRENTROOTKEY` equal to one of the preceding predefined constants, the predefined constant `HKEY_USER_SELECTABLE`, or the system variable [HKEY\\_USER\\_SELECTABLE\\_AUTO](#).

Unlike [RegDBGetDefaultRoot](#), the value of `HKEYCURRENTROOTKEY` is never `HKEY_USER_SELECTABLE`. If you most recently set the default root key by using `HKEY_USER_SELECTABLE`, the value of `HKEYCURRENTROOTKEY` is `HKEY_LOCAL_MACHINE` if the [ALLUSERS](#) system variable is non-zero or `HKEY_CURRENT_USER` if `ALLUSERS` is FALSE.

## HKEY\_USER\_SELECTABLE\_AUTO

The value of this system variable is `HKEY_LOCAL_MACHINE` if the [ALLUSERS](#) system variable is non-zero or `HKEY_CURRENT_USER` if `ALLUSERS` is FALSE.

## IFX\_COMPANY\_NAME



**Project** • This information applies to InstallScript projects.

This system variable is automatically initialized to the value of the string entry `COMPANY_NAME`, if that entry exists; if that entry does not exist, `IFX_COMPANY_NAME` is initialized to the company name that you specified in the Project Settings property sheet's Application page.



## IFX\_DISK1INSTALLED



**Project** • This information applies to InstallScript projects.

This system variable is set equal to zero at the beginning of a setup and is reset to a non-zero value if the setup installs or reinstalls the feature with the files needed for maintenance setups and uninstallation. (Note that this feature is automatically placed in your .cab files by the media builder and is not displayed in the IDE.)

## IFX\_INITIALIZED



**Project** • This information applies to InstallScript projects.

This system variable is set equal to a non-zero value if the setup is event based and is set equal to FALSE if the setup is procedural (that is, has a program...endprogram block).

## IFX\_INSTALLED\_DISPLAY\_VERSION



**Project** • This information applies to InstallScript projects.

The IFX\_INSTALLED\_DISPLAY\_VERSION system variable replaces the placeholder %VI in Sd dialog static text fields and string passed to the **SdSubstituteProductInfo** function. This system variable is automatically initialized to the value of the system variable IFX\_INSTALLED\_VERSION; if you assign a new value to IFX\_INSTALLED\_VERSION the value of IFX\_INSTALLED\_DISPLAY\_VERSION does not automatically change.

## IFX\_INSTALLED\_VERSION



**Project** • This information applies to InstallScript projects.

This system variable is automatically initialized to the string equivalent of the data in the Version value of the application uninstallation registry key if that data is a packed DWORD; if the key or value does not exist or the data is not a packed DWORD, IFX\_INSTALLED\_VERSION is initialized to a null string ("").

## IFX\_KEYPATH\_PRODUCT\_INFO

This system variable specifies the registry location of the application information key that is created by [CreateInstallationInfo](#) and whose values are read by [RegDBGetAppInfo](#) and modified by [RegDBSetAppInfo](#). This system variable is initialized to a value of:

**Software\<IFX\_COMPANY\_NAME>\<IFX\_PRODUCT\_NAME>\<IFX\_PRODUCT\_VERSION>\.**

## IFX\_MULTI\_INSTANCE\_SUFFIX



---

**Project** • This information applies to InstallScript projects.

The IFX\_MULTI\_INSTANCE\_SUFFIX system variable is set in the default code for the [OnFirstUIBefore](#) event handler function. IFX\_MULTI\_INSTANCE\_SUFFIX is used in that handler function to construct a unique target folder name for a multi-instance installation. It is also used in the [OnCustomizeUninstInfo](#) handler function to construct a unique uninstallation display name for a multi-instance installation.

## IFX\_PRODUCT\_COMMENTS



---

**Project** • This information applies to InstallScript projects.

If this system variable's value is not a null string (""), its value is used by the [MaintenanceStart](#) function to specify the data in the application uninstallation registry key's Comments value. This registry value provides information about the application to Add or Remove Programs in the Control Panel.

This system variable is initialized to the value that you specify in ARP Comments setting in the General Information view.

## IFX\_PRODUCT\_DISPLAY\_NAME

This system variable replaces the placeholder %P in Sd dialog static text fields and string passed to the **SdSubstituteProductInfo** function. This system variable is automatically initialized to the value of the system variable IFX\_PRODUCT\_NAME; if you assign a new value to IFX\_PRODUCT\_NAME the value of IFX\_PRODUCT\_DISPLAY\_NAME does not automatically change.



---

**Note** • The system variable IFX\_SETUP\_TITLE specifies the text in the title bar of all built-in dialogs.

## IFX\_PRODUCT\_DISPLAY\_VERSION



---

**Project** • This information applies to InstallScript projects.

This system variable replaces the placeholder %VS in Sd dialog static text fields and string passed to the **SdSubstituteProductInfo** function. This system variable is automatically initialized to the value of the system variable IFX\_PRODUCT\_VERSION; if you assign a new value to IFX\_PRODUCT\_VERSION the value of IFX\_PRODUCT\_DISPLAY\_VERSION does not automatically change.

## IFX\_PRODUCT\_ICON



**Project** • This information applies to InstallScript projects.

If this system variable's value is not a null string (""), its value is used by the [MaintenanceStart](#) function to specify the data in the application uninstallation registry key's DisplayIcon value. This registry value specifies the icon that is displayed for the application in Add or Remove Programs in the Control Panel.

This system variable is initialized to the value that you specify for the Display Icon setting in the Add or Remove Programs area of the General Information view.

## IFX\_PRODUCT\_KEY



**Project** • This information applies to InstallScript projects.

This system variable is automatically initialized to the value of the string entry PRODUCT\_KEY, if that entry exists; if that entry does not exist, IFX\_PRODUCT\_KEY is initialized to the executable file name that you specified in the Project Settings property sheet's Application page.

## IFX\_PRODUCT\_NAME



**Project** • This information applies to InstallScript projects.

This system variable is automatically initialized to the value of the string entry PRODUCT\_NAME, if that entry exists; if that entry does not exist, IFX\_PRODUCT\_NAME is initialized to the product name that you specified in the Project Settings property sheet's Application page.

## IFX\_PRODUCT\_README



**Project** • This information applies to InstallScript projects.

If this system variable's value is not a null string (""), its value is used by the [MaintenanceStart](#) function to specify the data in the application uninstallation registry key's Readme value. This registry value provides information about the application to Add or Remove Programs in the Control Panel.

This system variable is initialized to the value that you specify in the Read Me setting in the General Information view.

## IFX\_PRODUCT\_REGISTEREDCOMPANY

If the `IFX_PRODUCT_REGISTEREDCOMPANY` system variable's value is not a null string (""), its value is used by the [MaintenanceStart](#) function to specify the data in the application uninstallation registry key's RegCompany value. This registry value provides information about the application to Add or Remove Programs in the Control Panel.

This system variable is initialized to the data in the registry value `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows key\CurrentVersion\RegisteredOrganization`, where *Windows key* is Windows NT if the target operating system is Windows XP or later, or Windows for other Windows operating systems. This system variable's value is modified by end-user input in the Company Name edit box of the [SdRegisterUser](#), [SdRegisterUserEx](#), [SdCustomerInformation](#), and [SdCustomerInformationEx](#) dialogs.



**Project** • In an InstallScript MSI installation, if the value of `IFX_PRODUCT_REGISTEREDCOMPANY` is set, the Windows Installer property **COMPANYNAME** is automatically updated.

## IFX\_PRODUCT\_REGISTEREDOWNER

If the `IFX_PRODUCT_REGISTEREDOWNER` system variable's value is not a null string (""), its value is used by the [MaintenanceStart](#) function to specify the data in the application uninstallation registry key's RegOwner value. This registry value provides information about the application to Add or Remove Programs in the Control Panel.

This system variable is initialized to the data in the registry value `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows key\CurrentVersion\RegisteredOwner`, where *Windows key* is Windows NT if the target operating system is Windows XP or later, or Windows for other Windows operating systems. This system variable's value is modified by end-user input in the User Name edit box of the [SdRegisterUser](#), [SdRegisterUserEx](#), [SdCustomerInformation](#), and [SdCustomerInformationEx](#) dialogs.



**Project** • In an InstallScript MSI installation, if the value of `IFX_PRODUCT_REGISTEREDOWNER` is set, the Windows Installer property **USERNAME** is automatically updated.

## IFX\_PRODUCT\_REGISTEREDSERIALNUM

If the `IFX_PRODUCT_REGISTEREDSERIALNUM` system variable's value is not a null string (""), its value is used by the [MaintenanceStart](#) function to specify the data in the application uninstallation registry key's ProductId value. This registry value provides information about the application to Add or Remove Programs in the Control Panel.

This system variable's value is modified by end-user input in the Serial Number edit box of the [SdCustomerInformation](#), and [SdCustomerInformationEx](#) dialogs.

## IFX\_PRODUCT\_SUPPORT\_CONTACT



**Project** • This information applies to InstallScript projects.

If this system variable's value is not a null string (""), its value is used by the [MaintenanceStart](#) function to specify the data in the application uninstallation registry key's Contact value. This registry value provides information about the application to Add or Remove Programs in the Control Panel.

This system variable is initialized to the value that you specify in the Support Contact setting in the General Information view.

## IFX\_PRODUCT\_SUPPORT\_PHONE



---

**Project** • This information applies to InstallScript projects.

If this system variable's value is not a null string (""), its value is used by the [MaintenanceStart](#) function to specify the data in the application uninstallation registry key's HelpTelephone value. This registry value provides information about the application to Add or Remove Programs in the Control Panel.

This system variable is initialized to the value that you specify in the Support Phone Number setting in the General Information view.

## IFX\_PRODUCT\_SUPPORT\_URL



---

**Project** • This information applies to InstallScript projects.

If this system variable's value is not a null string (""), its value is used by the [MaintenanceStart](#) function to specify the data in the application uninstallation registry key's HelpLink value. This registry value provides information about the application to Add or Remove Programs in the Control Panel.

This system variable is initialized to the value that you specify in the Support URL setting in the General Information view.

## IFX\_PRODUCT\_UPDATE\_URL



---

**Project** • This information applies to InstallScript projects.

If this system variable's value is not a null string (""), its value is used by the [MaintenanceStart](#) function to specify the data in the [application uninstallation registry key's URLUpdateInfo](#) value. This registry value provides information about the application to Add or Remove Programs in the Control Panel.

This system variable is initialized to the value that you specify in the Product Update URL setting in the General information view.

## IFX\_PRODUCT\_URL



---

**Project** • This information applies to InstallScript projects.

If this system variable's value is not a null string (""), its value is used by the [MaintenanceStart](#) function to specify the data in the application uninstallation registry key's URLInfoAbout value. This registry value provides information about the application to Add or Remove Programs in the Control Panel.

This system variable is initialized to the value that you specify in the Publisher/Product URL setting in the General Information view.

## IFX\_PRODUCT\_VERSION



**Project** • This information applies to InstallScript projects.

This system variable is automatically initialized to the value of the string entry `PRODUCT_VERSION`, if that entry exists; if that entry does not exist, `IFX_PRODUCT_VERSION` is initialized to the product version that you specified in the Project Settings property sheet's Application page.

## IFX\_SETUP\_TITLE

This system variable specifies the text in the title bar of all built-in dialogs (except dialogs generated directly by Windows API function calls) and all message boxes generated by the `MessageBox` function. This system variable is automatically initialized to the value of the string entry `TITLE_CAPTIONBAR`, if that entry exists; if that entry does not exist, `IFX_SETUP_TITLE` is initialized with the following internal code:

```
Sprintf( IFX_SETUP_TITLE, SdLoadString( IDS_IFX_FORMAT_SETUP_TITLE ), IFX_PRODUCT_DISPLAY_NAME );
```

Changing the value of `IFX_SETUP_TITLE` automatically resets the title of all dialogs displayed by the setup.

## IFX\_SUPPORTED\_VERSIONS



**Project** • This information applies to InstallScript projects.

This system variable is automatically initialized to a pipe(|)-delimited list of the versions of your product to which the update can be applied, which you specified in the media property sheet's Update page or the Media Wizard's Update panel.

## INFOFILENAME

When you use [BatchFileSave](#) to save a batch file or [ConfigFileSave](#) to save a `Config.sys` file, you can specify that InstallShield create a backup of the file as it existed before you updated it. InstallShield assigns the name of that backup file to the system variable `INFOFILENAME`. If you want to alert the user to the existence of the backup file, use the function `MessageBox` to display the value of `INFOFILENAME`.

## INSTALLDIR



**Project** • This information applies to the following project types:

- Basic MSI
- InstallScript MSI

In an InstallScript project, use [TARGETDIR](#).

During setup initialization, the installation assigns to the system variable `INSTALLDIR` the fully qualified path to a target folder on the hard drive. The `INSTALLDIR` path is resolved based on the destination that is specified in the `INSTALLDIR` setting in the General Information view. By default, the `INSTALLDIR` path is resolved based on the `[ProgramFilesFolder]ISYourCompanyDir\ISYourProductDir` entries in the **Directory** table of your .msi package.

## INSTANCE\_GUID



**Project** • This information applies to InstallScript projects.

This system variable contains a globally unique identifier (GUID) for the installation, which is used as the name of the application uninstallation registry key. This variable's value is set at run time to equal `PRODUCT_GUID` except in a multi-instance installation.

The value of this system variable is shared among object scripts and between object scripts and the main installation script. You cannot assign a new value to this system variable.

## ISDIFXAPPID

This predefined global system variable determines the application to associate when installing or uninstalling device drivers. `ISDIFXAPPID` is set to `PRODUCT_GUID` by default during initialization and can be changed as desired to specify an alternate application ID.



**Note** • See the DIFxAPI documentation for the `INSTALLERINFO` structure for more information regarding specifying application association.

## ISMSI\_HANDLE

This system variable is set to the handle of the currently running .msi database, and can be used in event-handler functions as an argument to Windows Installer API functions that require a handle to the currently running database.

For example, to retrieve the value of the `USERNAME` property in the `OnBegin` event handler, you can use code similar to the following:

```
function OnBegin( )
    STRING svUsername[256];
    NUMBER nBuffer;
begin
    nBuffer = 256;
    MsiGetProperty(ISMSI_HANDLE, "USERNAME", svUsername, nBuffer);
    MessageBox("USERNAME = " + svUsername, INFORMATION);
end;
```



**Project** • `ISMSI_HANDLE` is not supported in Basic MSI projects, and is not supported in InstallScript custom actions.

## IS\_NULLSTR\_PTR

You can use the IS\_NULLSTR\_PTR variable to pass a null pointer to an external DLL function or Windows API through a parameter that has been prototyped as an InstallScript string. This functionality works for byval string, byref string, wstring, and binary data types.

This functionality does not apply to byref number parameters. If you want to specify a NULL pointer for byref number parameters, you must prototype the parameter as a pointer data type and pass the address of the number variable or NULL, as needed.

IS\_NULLSTR\_PTR is a global string variable instance with the value of <IS\_NULLSTR\_PTR>. A statement that assigns a new value to this variable compiles; however, the assignment does not have any effect. The variable's value remains <IS\_NULLSTR\_PTR>.

If you pass this variable to a non-DLL function, the function receives the string <IS\_NULLSTR\_PTR>.

If you pass a string that has the value <IS\_NULLSTR\_PTR> to an external DLL function, the result is the same as if you used IS\_NULLSTR\_PTR.

### Using the IS\_NULLSTR\_PTR Variable to Pass a Null Pointer to a Windows API

The Windows function **WritePrivateProfileString** lets you flush the INI file buffer on Windows 9x by specifying NULL for the first three parameters. However, since this function is prototyped as follows, there does not appear to be any way to accomplish this:

```
prototype number    KERNEL32.WritePrivateProfileString (byval string, byval string, byval string, byval
string);
```

Using the pointer data type allows NULL to be specified, but it causes problems when trying to specify a valid string.

If you want the InstallScript engine to pass a null pointer to the function, you can use the following code:

```
KERNEL32.WritePrivateProfileString (IS_NULLSTR_PTR, IS_NULLSTR_PTR, IS_NULLSTR_PTR, szFile);
```

### Using the IS\_NULLSTR\_PTR Variable to Pass a Null Pointer to an External DLL Function

You can use IS\_NULLSTR\_PTR with any external DLL function that expects a string. In this case, the DLL function receives a NULL pointer.

## ISRES

During setup initialization, the installation decompresses **\_isres.dll** from your setup and copies it to a temporary folder on the target system, giving it a unique name so that it does not interfere with other InstallShield installations. The fully qualified name of this file, which contains setup resources, is assigned to the system variable ISRES.

## ISUSER

During setup initialization, the installation decompresses **\_isuser.dll**, if present, from your setup and copies it to the temporary folder SUPPORTDIR on the target system, giving the file a unique name so that it does not interfere with other InstallShield installations. The fully qualified name of this file, which contains user-defined setup resources, is assigned to the system variable ISUSER.



## ISVERSION

When the setup script starts running, the installation gets the version of **Setup.exe** that is running and assigns it to the system variable ISVERSION. The version number also appears in the Setup program's About box.

## LAAW\_PARAMETERS


If you call **LaunchApplication** without LAAW\_OPTION\_USE\_SHELLEXECUTE or you call **LaunchAppAndWait** or **LaunchApp**, these functions internally call the Windows API function **CreateProcess**. The LAAW\_PARAMETERS structured variable specifies certain arguments for **CreateProcess**, and whether to display a text window while the launched application is running. For information on **CreateProcess**, consult the Windows API documentation.

The LAAW\_PARAMETERS system variable is initialized automatically during setup initialization by a call to [LaunchAppAndWaitInitStartupInfo](#).

**Table 8 • LAAW\_PARAMETERS**

Member	Description
<b>bCallbackEndedWait</b>	Indicates that <b>WaitForApplication</b> ended the wait because the callback function returned LAAW_CALLBACK_RETURN_END_WAIT.
<b>bInheritHandles</b>	Sets the corresponding argument to <b>CreateProcess</b> .
<b>dwCreationFlags</b>	Sets the corresponding argument to <b>CreateProcess</b> .
<b>lpCurrentDirectory</b>	Sets the corresponding argument to <b>CreateProcess</b> . This member is set to the szDirectory parameter of <b>LaunchApplication</b> or <b>LaunchAndAppAndWait</b> .  Note that setting the value of lpCurrentDirectory manually has no effect on <b>LaunchApplication</b> or <b>LaunchAndAppAndWait</b> .
<b>lpEnvironment</b>	Sets the corresponding argument to <b>CreateProcess</b> .
<b>lpProcessAttributes</b>	Sets the corresponding argument to <b>CreateProcess</b> .
<b>lpThreadAttributes</b>	Sets the corresponding argument to <b>CreateProcess</b> .
<b>nCallbackInterval</b>	This member defines the callback interval in milliseconds. It is set to 1000 (1 second) by default when you call the <b>LaunchApplicationInit</b> or <b>LaunchAppAndWaitInitStartupInfo</b> functions.
<b>nLaunchResult</b>	If the application cannot be launched, the nLaunchResult member contains the result of calling <b>GetLastError</b> after the <b>CreateProcess</b> call. If <b>LaunchApp</b> , <b>LaunchAppAndWait</b> , or <b>LaunchApplication</b> is successful and the LAAW_OPTION_WAIT option was specified, the nLaunchResult member contains the return code of the launched application.

**Table 8 •** LAAW\_PARAMETERS (cont.)

Member	Description
<b>nTimeout</b>	Indicates the timeout value used internally by <b>LaunchApplication</b> or <b>LaunchAndAppAndWait</b> when <b>WaitForApplication</b> is called. The default value is INFINITE. You can customize this value to set a wait timeout for <b>LaunchApplication</b> or <b>LaunchAndAppAndWait</b> .
<b>nTimeoutCheckInterval</b>	Indicates the interval for how often the installation checks whether the timeout interval has elapsed while waiting for an application through <b>WaitForApplication</b> (or through <b>LaunchApplication</b> or <b>LaunchAndAppAndWait</b> , which may call <b>WaitForApplication</b> internally). This value is not used if nTimeout is set to INFINITE and LAAW_USE_CALLBACK is not specified. If LAAW_USE_CALLBACK is specified, the timeout/callback check interval is the lower of the two values for LAAW_PARAMETERS.nTimeoutCheckInterval and LAAW_PARAMETERS.nCallbackInterval. The default value is 1000.
<b>nWaitForInputIdleMax</b>	<p>Indicates the maximum amount of time (in milliseconds) to wait for the application to complete its initialization through the Windows API <b>WaitForInputIdle</b>. The default value for this structure member is 2000. You can set it to 0 to indicate that the installation should not wait for the application to initialize before beginning the wait for the application to complete.</p> <p>Since <b>LaunchApplication</b> and <b>LaunchAndAppAndWait</b> wait for the application to initialize only if LAAW_OPTION_WAIT is specified, this value is only used if LAAW_OPTION_WAIT is specified.</p>
<b>nWaitResult</b>	Indicates additional information about the last wait that occurred as a result of calling <b>WaitForApplication</b> . For more information, see <a href="#">WaitForApplication</a> .
<b>szCommandLineResult</b>	Contains the resulting command line used as the lpCommandLine parameter in the internal call to <b>CreateProcess</b> . This member is populated when <b>LaunchApplication</b> or <b>LaunchAndAppAndWait</b> is called, so setting its value directly before or after this function is called has no effect. Also, this member is set to null ("" ) when you call the <b>LaunchApplicationInit</b> or <b>LaunchAppAndWaitInitStartupInfo</b> functions.
<b>szStatusText</b>	 <p><b>Project •</b> The <i>szStatusText</i> member is not available for use in <i>InstallScript</i> actions that are called in <i>Suite/Advanced UI</i> installations.</p> <p>If this member is set to anything other than a null string ("" ), the installation displays its contents in a text window (by calling <b>SdShowMsg</b>) while the launched application is running. Note that <i>szStatusText</i> cannot contain more than 4 kilobytes of data.</p>

## LAAW\_PROCESS\_INFORMATION

When you call **LaunchApplication**, **LaunchAndAppAndWait**, or **LaunchApp**, this structured variable returns identification information about the launched process. The PROCESS\_INFORMATION system variable has the following members:

**Table 9 •** LAAW\_PROCESS\_INFORMATION

Member	Description
<b>hProcess</b>	A handle to the newly created process. The handle is used to specify the process in all functions that perform operations on the process object.
<b>hThread</b>	A handle to the primary thread of the newly created process. The handle is used to specify the thread in all functions that perform operations on the thread object.
<b>dwProcessId</b>	A global process identifier that can be used to identify a process. The value is valid from the time the process is created until the time the process is terminated.
<b>dwThreadId</b>	A global thread identifiers that can be used to identify a thread. The value is valid from the time the thread is created until the time the thread is terminated.

## LAAW\_SHELLEXECUTEINFO

The LAAW\_SHELLEXECUTEINFO script variable is an instance of a SHELLEXECUTEINFO structure used by the **LaunchApplication** function when the **ShellExecuteEx** function is called. You can customize the members of this structure to affect how using **LaunchApplication** with the LAAW\_OPTION\_USE\_SHELLEXECUTE parameter works.

### SHELLEXECUTEINFO Structure

```
typedef SHELLEXECUTEINFO
begin
    int      cbSize;
    int      fMask;
    HWND     hwnd;
    pointer  lpVerb;
    pointer  lpFile;
    pointer  lpParameters;
    pointer  lpDirectory;
    int      nShow;
    HWND     hInstApp;
    pointer  lpIDLlist;
    pointer  lpClass;
    HWND     hkeyClass;
    int      dwHotKey;
```

```

        HWND    hIconMonitor;

        HWND    hProcess;

    end;

```

## LAAW\_SHELLEXECUTEVERB

The LAAW\_SHELLEXECUTEVERB script variable is a string that indicates the verb used by **LaunchApplication** when calling **ShellExecuteEx**. The default value is **open**. The lpVerb member of LAAW\_SHELLEXECUTEINFO points to this string by default.



**Tip** • If you are using LAAW\_OPTION\_USE\_SHELLEXECUTE on systems running Windows Vista or later and you want to launch the application using the full administrator account (similar to right-clicking the executable file to be run and clicking Run as Administrator), set LAAW\_SHELLEXECUTEVERB to **runas** before using **LaunchApplication** in your script:

```
LAAW_SHELLEXECUTEVERB = "runas";
```

This ensures that the application is always run with full administrator privileges regardless of whether the application to be launched has an application manifest with relevant settings. Note that this may trigger a User Account Control (UAC) prompt for consent or credentials.

On systems running operating systems earlier than Windows Vista, if **runas** is used, a Run As dialog box is displayed. The behavior is similar to right-clicking the executable file to be run and clicking Run As. This dialog box enables the end user to select the user account that should be used to run the application.

## LAAW\_STARTUPINFO

When you call **LaunchApplication**, **LaunchAndAppAndWait**, or **LaunchApp**, the LAAW\_STARTUPINFO structured variable specifies main window properties if a new window is created for the launched process. This system variable is initialized automatically during installation initialization by a call to [LaunchAppAndWaitInitStartupInfo](#).

The STARTUPINFO system variable has the following members:

**Table 10** • LAAW\_STARTUPINFO

Member	Description
<b>cb</b>	Specifies the size, in bytes, of the structure.
<b>lpReserved</b>	Reserved. Set this member to NULL.
<b>lpDesktop</b>	Pointer to a null-terminated string that specifies either the name of the desktop only or the name of both the desktop and window station for this process. A backslash in the string pointed to by lpDesktop indicates that the string includes both desktop and window station names. If lpDesktop is NULL, the new process inherits the desktop and window station of its parent process. If lpDesktop is an empty string, the process does not inherit the desktop and window station of its parent process; instead, the system determines if a new desktop and window station need to be created. If the impersonated user already has a desktop, the system will use the existing desktop.

**Table 10 •** LAAW\_STARTUPINFO (cont.)

Member	Description
<b>lpTitle</b>	For console processes, this is the title displayed in the title bar if a new console window is created. If NULL, the name of the executable file is used as the window title instead. This parameter must be NULL for GUI or console processes that do not create a new console window.
<b>dwX</b>	Ignored unless dwFlags specifies STARTF_USEPOSITION. Specifies the x offset, in pixels, of the upper left corner of a window if a new window is created. The offset is from the upper left corner of the screen. For GUI processes, the specified position is used the first time the new process calls the Windows API function <b>CreateWindow</b> to create an overlapped window if the x parameter of <b>CreateWindow</b> is CW_USEDEFAULT.
<b>dwY</b>	Ignored unless dwFlags specifies STARTF_USEPOSITION. Specifies the y offset, in pixels, of the upper left corner of a window if a new window is created. The offset is from the upper left corner of the screen. For GUI processes, the specified position is used the first time the new process calls the Windows API function <b>CreateWindow</b> to create an overlapped window if the y parameter of <b>CreateWindow</b> is CW_USEDEFAULT.
<b>dwXSize</b>	Ignored unless dwFlags specifies STARTF_USESIZE. Specifies the width, in pixels, of the window if a new window is created. For GUI processes, this is used only the first time the new process calls the Windows API function <b>CreateWindow</b> to create an overlapped window if the nWidth parameter of <b>CreateWindow</b> is CW_USEDEFAULT.
<b>dwYSize</b>	Ignored unless dwFlags specifies STARTF_USESIZE. Specifies the height, in pixels, of the window if a new window is created. For GUI processes, this is used only the first time the new process calls <b>CreateWindow</b> to create an overlapped window if the nHeight parameter of <b>CreateWindow</b> is CW_USEDEFAULT.
<b>dwXCountChars</b>	Ignored unless dwFlags specifies STARTF_USECOUNTCHARS. For console processes, if a new console window is created, dwXCountChars specifies the screen buffer width in character columns. This value is ignored in a GUI process.
<b>dwYCountChars</b>	Ignored unless dwFlags specifies STARTF_USECOUNTCHARS. For console processes, if a new console window is created, dwYCountChars specifies the screen buffer height in character rows. This value is ignored in a GUI process.
<b>dwFillAttribute</b>	Ignored unless dwFlags specifies STARTF_USEFILLATTRIBUTE. Specifies the initial text and background colors if a new console window is created in a console application. These values are ignored in GUI applications. This value can be any combination of the following values: FOREGROUND_BLUE, FOREGROUND_GREEN, FOREGROUND_RED, FOREGROUND_INTENSITY, BACKGROUND_BLUE, BACKGROUND_GREEN, BACKGROUND_RED, and BACKGROUND_INTENSITY. For information on using these Windows constants, which are not defined in <b>ISRTWindows.h</b> , see Using Windows Constants in a Script. For example, the following combination of values produces red text on a white background:  FOREGROUND_RED   BACKGROUND_RED   BACKGROUND_GREEN   BACKGROUND_BLUE

**Table 10 •** LAAW\_STARTUPINFO (cont.)

Member	Description
<b>dwFlags</b>	<p>This is a bit field that determines whether certain STARTUPINFO members are used when the process creates a window. Any combination of the following values can be specified:</p> <ul style="list-style-type: none"> <li>• <b>STARTF_FORCEONFEEDBACK</b>—Indicates that the cursor is in feedback mode for two seconds after <b>LaunchApplication</b> or <b>LaunchAndAppAndWait</b> is called. If during those two seconds the process makes the first GUI call, the system gives five more seconds to the process. If during those five seconds the process shows a window, the system gives five more seconds to the process to finish drawing the window.</li> </ul> <p>The system turns the feedback cursor off after the first call to the Windows API function <b>GetMessage</b>, regardless of whether the process is drawing.</p> <ul style="list-style-type: none"> <li>• <b>STARTF_FORCEOFFFEEDBACK</b>—Indicates that the feedback cursor is forced off while the process is starting. The normal cursor is displayed.</li> <li>• <b>STARTF_RUNFULLSCREEN</b>—Indicates that the process should be run in full-screen mode, rather than in windowed mode.</li> </ul> <p>This flag is only valid for console applications running on an x86 computer.</p> <ul style="list-style-type: none"> <li>• <b>STARTF_USECOUNTCHARS</b>—If this value is not specified, the dwXCountChars and dwYCountChars members are ignored.</li> <li>• <b>STARTF_USEFILLATTRIBUTE</b>—If this value is not specified, the dwFillAttribute member is ignored.</li> <li>• <b>STARTF_USEPOSITION</b>—If this value is not specified, the dwX and dwY members are ignored.</li> <li>• <b>STARTF_USESHOWWINDOW</b>—If this value is not specified, the wShowWindow member is ignored.</li> <li>• <b>STARTF_USESIZE</b>—If this value is not specified, the dwXSize and dwYSize members are ignored.</li> <li>• <b>STARTF_USESTDHANDLES</b>—Sets the standard input, standard output, and standard error handles for the process to the handles specified in the hStdInput, hStdOutput, and hStdError members of the STARTUPINFO structure.</li> </ul> <p>LAAW_PARAMETERS.bInheritHandles must be set to TRUE for this to work properly.</p> <p>If this value is not specified, the hStdInput, hStdOutput, and hStdError members of the STARTUPINFO structure are ignored.</p>
<b>wShowWindow</b>	<p>Ignored unless dwFlags specifies STARTF_USESHOWWINDOW. The wShowWindow member can be any of the SW_ constants defined in Winuser.h. For GUI processes, wShowWindow specifies the default value the first time the Windows API function <b>ShowWindow</b> is called. The nCmdShow parameter of <b>ShowWindow</b> is ignored. In subsequent calls to <b>ShowWindow</b>, the wShowWindow member is used if the nCmdShow parameter of <b>ShowWindow</b> is set to SW_SHOWDEFAULT.</p>
<b>cbReserved2</b>	Reserved; must be zero.

**Table 10 •** LAAW\_STARTUPINFO (cont.)

Member	Description
<b>lpReserved2</b>	Reserved; must be NULL.
<b>hStdInput</b>	Ignored unless dwFlags specifies STARTF_USESTDHANDLES. Specifies a handle that will be used as the standard input handle to the process if STARTF_USESTDHANDLES is specified.
<b>hStdOutput</b>	Ignored unless dwFlags specifies STARTF_USESTDHANDLES. Specifies a handle that will be used as the standard output handle to the process if STARTF_USESTDHANDLES is specified.
<b>hStdError</b>	Ignored unless dwFlags specifies STARTF_USESTDHANDLES. Specifies a handle that will be used as the standard error handle to the process if STARTF_USESTDHANDLES is specified.

## Example

To specify that the launched application should be displayed at coordinates (0,0), before calling **LaunchAppAndWait**, you would customize the structure as follows:

```
LAAW_STARTUPINFO.dwFlags = LAAW_STARTUPINFO.dwFlags | STARTF_USEPOSITION;
LAAW_STARTUPINFO.dwX = 0;
LAAW_STARTUPINFO.dwY = 0;
```

## MAINTENANCE

This system variable is set to TRUE if your installation program is running in maintenance mode, or set to FALSE for a first-time installation.

## MAINT\_OPTION



**Project •** This information applies to InstallScript projects.

The MAINT\_OPTION system variable is set to one of the following values, corresponding to the maintenance option that you set for the Maintenance Experience setting in the General Information view:

- MAINT\_OPTION\_STANDARD
- MAINT\_OPTION\_MULTI\_INSTANCE
- MAINT\_OPTION\_NONE

## MEDIA

This system variable stores the name of the current file media library or script-created feature set. During setup initialization, MEDIA is assigned the value of 'DATA', which corresponds to the DATAx.cab file that was created by the media build. If you change the value of this system variable to refer to a script-created component set, you must change the value back to 'DATA' before calling [FeatureMoveData](#).

## MODE

The system variable MODE holds one of the following constant values (note that the value cannot be changed at run time):

**Table 11 • MODE**

Constant	Meaning
<b>SILENTMODE</b>	Indicates that Setup.exe is running in silent mode. (That is, the user ran Setup.exe with the /s argument.)
<b>NORMALMODE</b>	Indicates Setup.exe is running in normal mode.
<b>RECORDMODE</b>	Indicates Setup.exe is automatically generating a silent setup file (.iss file), which is a record of the setup input, by default in the Windows folder. (That is, when you run Setup.exe with the /r argument.)

You can use the system variable MODE in if statements to control the flow of your script based on mode, as shown below:

```
if (MODE = SILENTMODE) then
    // Perform silent setup actions and events.
else
    // Perform normal setup actions and events.
endif;
```



**Note** • For a Basic MSI project, you can find if the user is running the installation in silent mode with the Windows Installer condition "UILevel=2".

## MSI\_TARGETDIR

MSI\_TARGETDIR represents the destination of an administrative installation (when a user runs Setup.exe with the /a argument) for an InstallScript MSI project.

For a Basic MSI project, the TARGETDIR *property* (not the InstallScript variable) contains the destination of an administrative installation.

## MULTI\_INSTANCE\_COUNT



**Project** • This information applies to InstallScript projects.

This system variable is set equal to the number of instances of the currently running multi-instance setup that are already installed on the target system. You cannot assign a new value to this system variable.



## PACKAGE\_LOCATION



**Project** • The `PACKAGE_LOCATION` system variable applies to InstallScript projects only.

This system variable contains the fully qualified file name of the installation's self-extracting executable file if the installation is running from a self-extracting executable file that was created from within InstallShield, or a null string ("" ) value otherwise.

## PRODUCT\_GUID

This read-only system variable contains the GUID for the setup, which is initialized to the value of the project's `ProductCode` property. By default, `PRODUCT_GUID` is used as part of the `UNINSTALLKEY` variable and also as part of the `DISK1TARGET` and `SUPPORTDIR` directories.

## PRODUCT\_INSTALLED



**Project** • This information applies to InstallScript projects.

This system variable is set to a non-zero value if a valid log file exists for the installation. If the installation is running with the standard maintenance option, this variable is equal to the `MAINTENANCE` system variable.

## PROGRAMFILES

The `PROGRAMFILES` system variable contains the fully qualified name of the folder defined by Windows to store applications. In English Windows, that folder is named Program Files, and it is located off the root of the drive on which Windows is installed. (In other language versions of Windows, the folder name is localized appropriately by default.) The program files folders is the recommended default location for application folders.

On 64-bit Windows systems, this folder is for 32-bit applications only and has the name Program Files (x86) by default; 64-bit applications should be installed to the `PROGRAMFILES64` folder.



**Tip** • If your company distributes more than one application, you may prefer to create a company folder inside the program files folder and then create application folders within the company folder.

This system variable is read-only; if you attempt to assign a value to it, a compiler error results. The value of this system variable is shared among object scripts and between object scripts and the main setup script.



**Project** • During setup initialization in InstallScript installations, the value of the `PROGRAMFILES` variable is obtained by calling the Windows API function **`SHGetSpecialFolderPath`** with the `CSIDL_PROGRAM_FILES` parameter.

In Basic MSI and InstallScript MSI installations, the value of the `PROGRAMFILES` variable is initialized based on the Windows Installer property **`ProgramFilesFolder`**. Note that deferred, commit, and rollback custom actions do not have access to this property. Therefore, the corresponding `PROGRAMFILES` variable is empty in deferred, commit, and

rollback custom actions. To learn more, see *Accessing or Setting Windows Installer Properties Through Deferred, Commit, and Rollback Custom Actions*.

## PROGRAMFILES64

The PROGRAMFILES64 system variable contains the fully qualified name of the folder defined by Windows to store 64-bit applications on a 64-bit system. (Note that 32-bit applications are stored under the PROGRAMFILES folder.) In English Windows, that folder is named Program Files, and it is located off the root of the drive on which Windows is installed. (In other language versions of Windows, the folder name is localized appropriately by default.) The program files folders are the recommended default locations for application folders.



**Tip** • If your company distributes more than one application, you may prefer to create a company folder inside the program files folder and then create application folders within the company folder.

This system variable is read-only; if you attempt to assign a value to it, a compiler error results. The value of this system variable is shared among object scripts and between object scripts and the main setup script.



**Project** • During setup initialization in InstallScript installations, the value of the PROGRAMFILES64 variable is obtained by calling the Windows API function **SHGetSpecialFolderPath** with the CSIDL\_PROGRAM\_FILES parameter from a 64-bit executable file.

In Basic MSI and InstallScript MSI installations, the value of the PROGRAMFILES64 variable is initialized based on the Windows Installer property **ProgramFiles64Folder**. Note that deferred, commit, and rollback custom actions do not have access to this property. Therefore, the corresponding PROGRAMFILES64 variable is empty in deferred, commit, and rollback custom actions. To learn more, see *Accessing or Setting Windows Installer Properties Through Deferred, Commit, and Rollback Custom Actions*.

## REGDB\_OPTIONS

The REGDB\_OPTIONS system variable enables you to set various options for the general registry functions. The following table describes the options that you can specify:

**Table 12 • REGDB\_OPTIONS**

Option	Meaning
<b>REGDB_OPTION_DISABLETEXTSUBS</b>	Disables text substitutions in strings that are passed to registry functions. Use this option when working with registry function strings that contain opening angle brackets (<) and closing angle brackets (>) but that should not be interpreted as text substitutions.

**Table 12 • REGDB\_OPTIONS (cont.)**

Option	Meaning
<b>REGDB_OPTION_NO_DELETE_OLD_MAJMIN_VERSION</b>	<p>Prevents the <b>MaintenanceStart</b> function from deleting the legacy values of the following constants:</p> <ul style="list-style-type: none"> <li>REGDB_VALUENAME_UNINSTALL_MAJORVERSION (the major version value name under the application uninstallation key), whose value was MajorVersion for installations that were created with InstallShield 2009 and earlier</li> <li>REGDB_VALUENAME_UNINSTALL_MINORVERSION (the minor version value name under the application uninstallation key), whose value was MinorVersion for installations that were created with InstallShield 2009 and earlier</li> </ul> <p>For more information, see the “Changes to the Major and Minor Version Registry Entries for the Uninstall Key of InstallScript Installations” section in Upgrading Projects from InstallShield 2009 or Earlier.</p>
<b>REGDB_OPTION_WOW64_64KEY</b>	<p>Specifies that all future general registry operations affect the 64-bit parts of the registry instead of the 32-bit parts of the registry (on a 64-bit system). Setting this option on a 32-bit system has no effect.</p> <p>For more information about installing to 64-bit registry locations, see Targeting 64-Bit Operating Systems with InstallScript Installations.</p>
<b>REGDB_OPTION_USE_DEFAULT_OPTIONS</b>	Resets (clears) all previously set options.

To add options, combine one or more options using bitwise OR (|) operator as shown:

```
REGDB_OPTIONS = REGDB_OPTIONS | REGDB_OPTION_WOW64_64KEY
```

To remove options, specify the option to remove using the bitwise AND (&) operator and the bitwise NOT (~) operator as shown:

```
REGDB_OPTIONS = REGDB_OPTIONS & ~REGDB_OPTION_WOW64_64KEY
```



**Note •** When you enable the `REGDB_OPTION_WOW64_64KEY` option, this affects where registry entries from registry sets are created. For example, if this option is enabled when you call the [CreateRegistrySet](#) function, the registry set is created in the 64-bit part of the registry. If you enable this option for the specific 64-bit registry sets you want to install, it is recommended that you then disable the option so other registry entries or sets are not incorrectly created in the 64-bit part of the registry. For more information about installing to 64-bit registry locations, see *Targeting 64-Bit Operating Systems with InstallScript Installations*.

The InstallScript engine currently does not support installing Add or Remove Programs information for a product in the 64-bit part of the registry; therefore, the `REGDB_OPTION_WOW64_64KEY` option is not supported for the specific

registry functions such as **CreateInstallationInfo**, **MaintenanceStart**, **RegDBGetItem**, **RegDBSetItem**, **RegDBGetAppInfo**, **RegDBSetAppInfo**, and **RegGetUninstCmdLine**.

## REINSTALLMODE

This system variable is non-zero if one of the reinstall functions has been called in an InstallScript installation—that is, if **FeatureReinstall**, **FeatureUpdate**, or **FeaturePatch** have been called in the current instance of the installation.



**Project** • In an InstallScript MSI installation, this system variable is non-zero if the **FeatureReinstall** function is called. **FeatureUpdate** and **FeaturePatch** are not defined in an InstallScript MSI installation and should not be called.

## REMOVEALLMODE



**Project** • This information applies to InstallScript projects.

This system variable is non-zero if the application is being completely uninstalled—that is, if **FeatureRemoveAll**, **FeatureRemoveAllInMedia**, or **FeatureRemoveAllInMediaAndLog** have been called in the current instance of the setup—and FALSE otherwise. The value of this system variable is shared among object scripts and between object scripts and the main setup script.



**Note** • To execute script code only when the application is being completely uninstalled, enclose the code in the following if-then statement:

```
if REMOVEALLMODE!=0 then
    /* this code is executed only during uninstallation */
endif;
```

To perform specific uninstallation actions when a particular component is uninstalled, override the component's <ComponentName>\_Uninstalling event and perform the actions in this event.

## REMOVEONLY



**Project** • This information applies to the following project types:

- InstallScript
- InstallScript MSI

The REMOVEONLY system variable is set equal to a non-zero value if Setup.exe is run with the -removeonly option, and is set equal to FALSE otherwise. The default code for the OnMaintUIBefore event handler function conditionally displays the SdWelcomeMaint dialog, depending on the value of REMOVEONLY.

This system variable is read-only; if you attempt to assign a value to it, a compiler error results.

## SELECTED\_LANGUAGE



**Project** • The following project types support `SELECTED_LANGUAGE`:

- `InstallScript`
- `InstallScript MSI`

The numeric system variable `SELECTED_LANGUAGE` contains the ID of the language that the installation is using to display prompts and messages.

This system variable has a corresponding `<SELECTED_LANGUAGE>` text substitution, which contains the value of `SELECTED_LANGUAGE` formatted as a four-digit hexadecimal value (including the 0x prefix). For example, if `SELECTED_LANGUAGE` is `ISLANG_ENGLISH_UNITEDSTATES`, the value of the text substitution is `0x0409`.

## SHAREDSUPPORTDIR



**Project** • `InstallScript` projects support `SHAREDSUPPORTDIR`.

The read-only variable `SHAREDSUPPORTDIR` identifies the directory that contains all of the support files that are shared among an `InstallScript` installation and all of the `InstallScript` objects in that installation.

This system variable has a corresponding `<SHAREDSUPPORTDIR>` text substitution.

## SHELL\_OBJECT\_FOLDER



**Project** • The following project types support `SHELL_OBJECT_FOLDER`:

- `InstallScript`
- `InstallScript MSI`

The `SHELL_OBJECT_FOLDER` system variable is used to specify the name of a shell object folder—typically a Start Menu folder—at run time through script.

You can specify **`SHELL_OBJECT_FOLDER`** (for `InstallScript` or `InstallScript MSI` projects) or **`<SHELL_OBJECT_FOLDER>`** (for `InstallScript` projects) in the Display Name setting for a folder in the Shortcuts view. Then you can define the display name for the folder at run time by setting the `SHELL_OBJECT_FOLDER` variable in your script before the shortcut is created. The shortcut is typically created during file transfer.



**Project** • For `InstallScript` projects, you can specify either **`<SHELL_OBJECT_FOLDER>`** or **`SHELL_OBJECT_FOLDER`** in the Display Name setting. In both cases, text substitution is used. However, including the angle brackets—**`<SHELL_OBJECT_FOLDER>`**—is recommended.

For `InstallScript MSI` projects, you must specify **`SHELL_OBJECT_FOLDER`** (without the angle brackets).

To use this functionality in an `InstallScript MSI` installation, any letters that are specified for the Key Name setting of the folder in the Shortcuts view must be all uppercase (for example, `NEWFOLDER1`).

If the installation is not in maintenance mode, SHELL\_OBJECT\_FOLDER is initialized to the same value as IFX\_PRODUCT\_NAME during initialization of the InstallScript engine. Note that these variables are not synchronized once initialized; therefore, if you change one and want the other to change, you must change both manually. These types of manual changes are logged, and the changes are read from the log on subsequent maintenance operations. Therefore, a shortcut that uses the SHELL\_OBJECT\_FOLDER variable for its display name can be removed during uninstallation.

## SHOW\_PASSWORD\_DIALOG



**Project** • This information applies to InstallScript projects.

The SHOW\_PASSWORD\_DIALOG system variable is TRUE if the "Show Password dialog box during setup initialization" check box was checked in the Media Wizard's General Options panel or the media property sheet's General page, or FALSE otherwise.

## SRCDIR

This system variable contains the fully qualified path to the source folder that contains the Windows Installer package.

SRCDIR is initialized to the value of the Windows Installer property **SourceDir** when the sequence begins, and it cannot be assigned a new value in an InstallScript custom action.

## SRCDISK

This system variable contains the name of the drive with the source disk. During setup initialization, InstallShield assigns to SRCDISK the name of the drive that holds the disk containing the setup script file, Setup.inx. For example, if you start Setup.exe from a floppy disk in the A drive, then if that disk contains the file Setup.inx, InstallShield assigns the value "A:" to SRCDISK. Note that InstallShield includes the colon (:) with the drive letter.



**Note** • If you intend to reference the root folder of the drive specified by this variable, you must append a backslash to it (specified as two backslashes). For example, if the value of SRCDISK is A:, the following statement refers to the root folder of that drive: SRCDISK + "\\".

## SUPPORTDIR

During setup initialization, the installation locates a folder on the target system into which it can copy temporary files and files that were compressed into your installation. The installation sets the value of SUPPORTDIR to the fully qualified path for that folder.

In addition, files that you add to the Language Independent (or language-specific) file list in the Support Files/ Billboards view of InstallShield are decompressed into SUPPORTDIR when the installation initializes, and they are deleted when the installation is complete.

To access a particular support file in an InstallScript project, use the SUPPORTDIR variable directly and then append the file name to the SUPPORTDIR value to obtain the complete path of the file. Following is an example of InstallScript event code:

```
prototype STRING GetSupportFilePathIS(STRING);
function STRING GetSupportFilePathIS(szSupportFile)
begin
    return SUPPORTDIR ^ szSupportFile;
end;
```



**Note** • The value of the InstallScript variable SUPPORTDIR is not shared among InstallScript Object scripts or between InstallScript Object scripts and the main installation script.



**Project** • Note that the value of the InstallScript system variable SUPPORTDIR is not the same as the value of the Windows Installer property **SUPPORTDIR**.

*In event-driven InstallScript, the SUPPORTDIR system variable points to the folder that contains support files.*

*In Basic MSI or InstallScript MSI projects, each InstallScript custom action initializes its own engine. Each engine does not know where the primary SUPPORTDIR is, and each engine does not extract its own private copy of the support files. For instructions on locating the extracted support files from a custom action, see [Placing Files in the .msi Database and Extracting Them During Run Time](#).*

## SYSINFO

During setup initialization, the installation sets the members of the SYSINFO structure variable to identify the operating platform of the target computer. By inspecting the values assigned to members of this variable, your script can determine information such as the following:

- The operating system
- The major and minor version number of the operating system
- The subversion of the operating system
- The version of Internet Explorer
- The latest service pack that is installed
- If the end user has administrator rights under Windows NT
- If the end user is a power user
- If the system is 64-bit
- If the system is a virtual machine
- The language IDs of the system language, user language, and operating system language


The following table shows the meaning of each SYSINFO member:

**Table 13 • SYSINFO Members**

Member	Meaning
<b>SYSINFO.bIntel</b>	If TRUE, the processor is Intel.
<b>SYSINFO.bIsVirtualMachine</b>	If TRUE, a virtual machine is detected.  For more information, see <a href="#">Detecting Whether the Installation Is Being Run on a Virtual Machine</a> .
<b>SYSINFO.bIsWow64</b>	If the installation is running on a 64-bit platform, this value is non-zero.
<b>SYSINFO.bShellExplorer</b>	If TRUE, the shell is Explorer.
<b>SYSINFO.bWinServer2003R2</b>	If this member is TRUE, the operating system is Windows Server 2003 R2.




**Table 13 • SYSINFO Members (cont.)**

Member	Meaning
<b>SYSINFO.nISOSL</b>	<p>Value indicates the operating system of the target machine. Possible values are the following:</p> <ul style="list-style-type: none"> <li>• <b>ISOSL_WINXP</b>—Windows XP Edition</li> <li>• <b>ISOSL_WINSERVER2003</b>—Windows Server 2003</li> <li>• <b>ISOSL_WINVISTA_SERVER2008 (or ISOSL_WINVISTA)</b>—Note that Windows Vista and Windows Server 2008 use the same major and minor version numbers. Therefore, if you want to use InstallScript to distinguish between Windows Server 2008 and Windows Vista, check whether <code>SYSINFO.nOSProductType = VER_NT_WORKSTATION</code>; for Windows Vista, this is TRUE; for Windows Server 2008, it is FALSE.</li> <li>• <b>ISOSL_WIN7_SERVER2008R2</b>—Windows 7 or Windows Server 2008 R2</li> <li>• <b>ISOSL_WIN8</b>—Windows 8 or Windows Server 2012</li> <li>• <b>ISOSL_WIN81</b>—Windows 8.1 or Windows Server 2012 R2</li> <li>• <b>ISOSL_WIN10</b>—Windows 10</li> </ul>  <p><b>Note</b> • Several client and server versions of Windows use the same major and minor version numbers:</p> <ul style="list-style-type: none"> <li>• Windows 8.1 and Windows Server 2012 R2 use the same major and minor version numbers.</li> <li>• Windows 8 and Windows Server 2012 use the same major and minor version numbers.</li> <li>• Windows 7 and Windows Server 2008 R2 use the same major and minor version numbers.</li> <li>• Windows Vista and Windows Server 2008 have the same major and minor version numbers.</li> </ul> <p>Therefore, for these operating system versions, the installation considers the client versions to be the same as the equivalent server versions; thus, components that are marked for the client version are also installed on the server version. To distinguish between the client and server versions, you can check whether <code>SYSINFO.nOSProductType</code> is equal to <code>VER_NT_WORKSTATION</code>; on client versions, this is true. On server versions, this is false.</p>
<b>SYSINFO.nOSMajor</b>	Value indicates operating system's major version number.
<b>SYSINFO.nOSMinor</b>	Value indicates operating system's major version number.





**Table 13 • SYSINFO Members (cont.)**

Member	Meaning
<b>SYSINFO.nOSProductType</b>	<p>Value indicates the wProductType of the Windows OSVERSIONINFOEX structure as defined for the current platform. Possible values are the following:</p> <ul style="list-style-type: none"> <li>• <b>VER_NT_WORKSTATION</b></li> <li>• <b>VER_NT_DOMAIN_CONTROLLER</b></li> <li>• <b>VER_NT_SERVER</b></li> </ul> <p>You can also #define and test for any other constant supported by wProduct Type. To learn more, see <a href="#">OSVERSIONINFOEX Structure</a> on the MSDN Web site.</p>
<b>SYSINFO.nOSSuiteMask</b>	<p>Value indicates the wSuitesMask of the Windows OSVERSIONINFOEX structure as defined for the current platform. Possible values are the following:</p> <ul style="list-style-type: none"> <li>• <b>VER_SUITE_BACKOFFICE</b></li> <li>• <b>VER_SUITE_DATACENTER</b></li> <li>• <b>VER_SUITE_ENTERPRISE</b></li> <li>• <b>VER_SUITE_PERSONAL</b></li> <li>• <b>VER_SUITE_SMALLBUSINESS</b></li> <li>• <b>VER_SUITE_SMALLBUSINESS_RESTRICTED</b></li> <li>• <b>VER_SUITE_TERMINAL</b></li> </ul> <p>You can also #define and test for any other constant supported by wSuiteMask. To learn more, see <a href="#">OSVERSIONINFOEX Structure</a> on the MSDN Web site.</p>

**Table 13 • SYSINFO Members (cont.)**

Member	Meaning
<b>SYSINFO.nSuites</b>	<p>A combination of one or more bit flags that indicate the suite or suites on the target machine. Possible bit flags are the following:</p> <ul style="list-style-type: none"> <li>• <b>ISOS_ST_ALL</b></li> <li>• <b>ISOS_ST_XP_PRO</b></li> <li>• <b>ISOS_ST_XP_HOME</b></li> <li>• <b>ISOS_ST_SERVER</b></li> <li>• <b>ISOS_ST_SERVER2003_R2</b></li> <li>• <b>ISOS_ST_WORKSTATION</b></li> <li>• <b>ISOS_ST_BACKOFFICE</b></li> <li>• <b>ISOS_ST_DATACENTER</b></li> <li>• <b>ISOS_ST_ENTERPRISE</b></li> <li>• <b>ISOS_ST_SERVER2003_R2</b></li> <li>• <b>ISOS_ST_SMALLBUSINESS</b></li> <li>• <b>ISOS_ST_SMALLBUSINESS_RESTRICTED</b></li> <li>• <b>ISOS_ST_TERMINAL</b></li> <li>• <b>0 (zero)</b>—indicates that no suite is detected on the target machine</li> </ul> <p>To check whether a bit flag is set, use the bitwise AND (&amp;) operator as in the following example:</p> <pre>if (SYSINFO.nSuites &amp; ISOS_ST_XP_HOME) then     /* Perform operations that are        specific to Windows XP Home Edition. */ endif;</pre>  <p><b>Note •</b> The suites listed here are those that can be specified in the Windows API's OSVERSIONINFOEX data structure.</p>
<b>SYSINFO.nSystemDefaultUILangID</b>	Value indicates the installed operating system language's ID.
<b>SYSINFO.nSystemLangID</b>	Value indicates the system language's ID.
<b>SYSINFO.nUserLangID</b>	Value indicates the user language's ID.
<b>SYSINFO.nWinMajor</b>	Value indicates Windows major version number.
<b>SYSINFO.nWinMinor</b>	Value indicates Windows minor version number.

**Table 13 • SYSINFO Members (cont.)**

Member	Meaning
<b>SYSINFO.szInstalledIEVersion</b>	<p>Value indicates the Internet Explorer version on the system. This member is supported for versions 4 and later. If the version installed is earlier than 4, the value is null ("").</p>  <p><b>Note •</b> Do not rely on the fact that this value is null for versions earlier than 4. Instead, test specifically for 4 or later, since in the future, this member variable could support detecting Internet Explorer versions earlier than 4.</p>
<b>SYSINFO.WINNT.bAdmin_Logged_On</b>	<p>If this member is TRUE, the end user is logged on under NT with administrator rights.</p>
<b>SYSINFO.WINNT.bPowerUser_Logged_On</b>	<p>If this member is TRUE, the current user belongs to the power user group.</p>
<b>SYSINFO.WINNT.bWin10</b>	 <p><b>Note •</b> This member is applicable to event-based InstallScript code; it is not applicable to InstallScript custom actions.</p> <p>If this member is TRUE, the operating system is Windows 10.</p>
<b>SYSINFO.WINNT.bWin81</b>	 <p><b>Note •</b> This member is applicable to event-based InstallScript code; it is not applicable to InstallScript custom actions.</p> <p>If this member is TRUE, the operating system is Windows 8.1 or Windows Server 2012 R2.</p>
<b>SYSINFO.WINNT.bWin8</b>	 <p><b>Note •</b> This member is applicable to event-based InstallScript code; it is not applicable to InstallScript custom actions.</p> <p>If this member is TRUE, the operating system is Windows 8 or Windows Server 2012.</p>
<b>SYSINFO.WINNT.bWin7_Server2008R2</b>	 <p><b>Note •</b> This member is applicable to event-based InstallScript code; it is not applicable to InstallScript custom actions.</p> <p>If this member is TRUE, the operating system is Windows 7 or Windows Server 2008 R2.</p>
<b>SYSINFO.WINNT.bWinNT</b>	<p>If this member is TRUE, the operating system is Windows NT (includes Windows XP).</p>

**Table 13 • SYSINFO Members (cont.)**

Member	Meaning
<b>SYSINFO.WINNT.bWinVista_Server2008</b> <b>(SYSINFO.WINNT.bWinVista)</b>	If either SYSINFO.WINNT.bWinVista_Server2008 or SYSINFO.WINNT.bWinVista is TRUE, the operating system is Windows Vista or Windows Server 2008.  To distinguish between Windows Server 2008 and Windows Vista, check whether SYSINFO.nOSProductType is equal to VER_NT_WORKSTATION; for Windows Vista, this is TRUE; for Windows Server 2008, it is FALSE.
<b>SYSINFO.WINNT.bWinXP</b>	If this member is TRUE, the operating system is Windows XP.
<b>SYSINFO.WINNT.bWinServer2003</b>	If this member is TRUE, the operating system is Windows Server 2003 or Windows Server 2003 R2.
<b>SYSINFO.WINNT.nServicePack</b>	The number of the installed service pack.  The installation obtains this information by calling the Windows API <b>GetVersionEx</b> and reading the nServicePackMajor value.

## Example

The following code fragment displays a message box if the operating system on the target system is Windows XP.

```
if (SYSINFO.WINNT.bWinXP) then
    MessageBox("Installing on Windows XP", INFORMATION);
endif;
```

## SYSPROCESSORINFO

During setup initialization, the installation sets the members of this structure variable to identify information about the processor of the target computer. By inspecting the values assigned to members of this variable, your script can determine information such as the number of processors on the system and the type of processor.



The following table shows the meaning of each SYSPROCESSORINFO member:



**Note •** Each of these members corresponds to a member in the Windows *SYSINFO* structure. These are populated during initialization by calling the Windows API *GetSystemInfo* or *GetNativeSystemInfo* functions on 64-bit Windows

systems. Consult the documentation on this structure in the MSDN Library. Also, as documented by Microsoft, using *nProcessorType* is not recommended. Use *nProcessorLevel* and *nProcessorArchitecture* instead.

**Table 14 • SYSPROCESSORINFO**

Member	Meaning
<b>SYSPROCESSORINFO.nProcessorArchitecture</b>	<p>Indicates the processor architecture. Possible values are the following:</p> <p><b>PROCESSOR_ARCHITECTURE_INTEL</b></p> <p><b>PROCESSOR_ARCHITECTURE_IA64</b></p> <p><b>PROCESSOR_ARCHITECTURE_AMD64</b></p> <p><b>PROCESSOR_ARCHITECTURE_UNKNOWN</b></p> <p></p> <p><b>Note</b> • <i>InstallScript</i> includes constants for the most common values for this structure member. However, in unusual cases, this structure member could contain other values defined by Windows. Consult the Windows documentation on the corresponding <i>SYSINFO</i> member for more information on possible additional values.</p>
<b>SYSPROCESSORINFO.nNumberOfProcessors</b>	<p>Indicates the number of processors on the system.</p>
<b>SYSPROCESSORINFO.nProcessorType</b>	<p>Indicates the processor type. Possible values are the following:</p> <p><b>PROCESSOR_INTEL_386</b></p> <p><b>PROCESSOR_INTEL_486</b></p> <p><b>PROCESSOR_INTEL_PENTIUM</b></p> <p><b>PROCESSOR_INTEL_IA64</b></p> <p><b>PROCESSOR_AMD_X8664</b></p> <p></p> <p><b>Note</b> • <i>InstallScript</i> includes constants for the most common values for this structure member. However, in unusual cases, this structure member could contain other values defined by Windows. Consult the Windows documentation on the corresponding <i>SYSINFO</i> member for more information on possible additional values.</p>
<b>SYSPROCESSORINFO.nProcessorLevel</b>	<p>Indicates the system's architecture-dependent processor level. This is often defined by the vendor and should be used for display purposes. See the Windows documentation on the <i>SYSINFO</i> structure for further information on the meaning of these members.</p>
<b>SYSPROCESSORINFO.nProcessorRevision</b>	<p>Indicates the system's architecture-dependent processor revision. See the Windows documentation on the <i>SYSINFO</i> structure for further information on the meaning of these members.</p>

## TARGETDIR

During setup initialization, the installation assigns to the system variable TARGETDIR the fully qualified path to a target folder on the hard disk. This folder will be the one containing the file Win.ini, usually the Windows folder. Some InstallScript functions use this variable when performing file operations. You must set this variable to the folder you want to target before calling these functions. The default code for the OnFirstUIBefore event handler function assigns a value to TARGETDIR.

The value of this system variable is shared among object scripts and between object scripts and the main setup script. The value you assign to this system variable in any script is the value it has in the subsequently executed code in any script (until its value is explicitly reset).

## TARGETDISK

During setup initialization, the installation assigns the name of the target disk drive to the system variable TARGETDISK. This drive will be the one containing the file Win.ini, usually the C: drive. Note that InstallShield includes the colon (:) with the drive letter.



**Note** • If you intend to reference the root folder of the drive specified by this variable, you must append a backslash to it (specified as two backslashes). For example, if the value of TARGETDISK is C:, the following statement refers to the root folder of that drive: TARGETDISK + "\\".

## UNINST



**Project** • The following project types support UNINST:

- InstallScript
- InstallScript MSI

The UNINST system variable is provided for compatibility with earlier versions of InstallShield software. It contains the command line that launches the copy of **Setup.exe** that was placed on the target system to perform uninstallation. The default value is:

```
<UNINSTALL_STRING> -uninst
```

If you use this command line, the installation runs the OnUninstall event when the installation is launched. For more details, review the /uninst command-line parameter information for **Setup.exe**.

This command line is placed in the appropriate registry value by the [DeinstallStart](#) function, which is provided for compatibility with previous versions of InstallShield software.

The value of this system variable is shared among object scripts and between object scripts and the main setup script.

You can append your own custom command line switches to UNINST for processing by your script's uninstallation code. If you do so, and you change the value of the system variable [DISK1TARGET](#), be sure to change DISK1TARGET before appending to UNINST; UNINST incorporates DISK1TARGET and is automatically changed when you change DISK1TARGET.

## UNINSTALLKEY



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

*Note that project-specific differences are provided where applicable.*

The UNINSTALLKEY system variable contains the name of the registry key that is used to store your uninstallation information. The registry key is placed under SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall.

In an InstallScript installation, this key is placed under HKEY\_USER\_SELECTABLE\_AUTO, which is controlled by the value of the ALLUSERS script variable. In an InstallScript MSI installation, this key is placed under HKEY\_LOCAL\_MACHINE if the installation is being run by an administrator; otherwise, it is placed under HKEY\_CURRENT\_USER.

In an InstallScript installation, the default value is **INSTANCE\_GUID**. In an InstallScript MSI installation, the default value of UNINSTALLKEY is **InstallShield\_{ProductCode}**.

To specify a different uninstallation key, assign a new value to UNINSTALLKEY in your script, as in the following example:

```
UNINSTALLKEY = "Sample App";
```

Note that to avoid conflicts with other installed applications, be sure to use a value that is unique to your application.

If the UNINSTALLKEY variable has been changed from the default value, the installation automatically creates an additional registry key for the product:

- **For InstallScript installations**—REGDB\_KEYPATH\_ISUNINSTINFO ^ INSTANCE\_GUID
- **For InstallScript MSI installations**—REGDB\_KEYPATH\_ISUNINSTINFO ^ [ProductGuid]

This key contains a single string value with the name **UninstallKey** (REGDB\_VALUENAME\_UNINSTALLKEY); the value data contains the name of the product's uninstall key as determined by the UNINSTALLKEY variable. This value can be used to allow other products to find and use the uninstall information for the product without the product's custom uninstall key.

## UNINSTALL\_DISPLAYNAME

This system variable contains the displayed name of your product in Add or Remove Programs. This value is typically the product name you specified in the General Information view.

To specify a different uninstallation display name, assign a new value to UNINSTALL\_DISPLAYNAME in your script before data transfer, as in the following:

```
UNINSTALL_DISPLAYNAME = "Sample App";
```



**Project** • UNINSTALL\_DISPLAYNAME is used only in InstallScript and InstallScript MSI installation projects. Basic MSI projects use the value of the **ProductName** property as the product's display name in Add or Remove Programs.



## UNINSTALL\_STRING



**Project** • The following project types support `UNINSTALL_STRING`:

- `InstallScript`
- `InstallScript MSI`

The `UNINSTALL_STRING` system variable contains the command line that launches the setup launcher, the installation's executable file, which was placed on the target system to perform uninstallation. The default value is:

```
<DISK1TARGET>\<DISK1SETUPEXENAME> -runfromtemp -l<SELECTED_LANGUAGE>
```

The setup launcher automatically writes the `UNINSTALL_STRING` command line to the registry, unless you have hidden the Remove button (via the Disable Remove Button property) in Add or Remove Programs.

You can append your own custom command line switches to `UNINSTALL_STRING` for processing by your script's uninstallation code.

## UPDATEMODE

The `UPDATEMODE` system variable applies to `InstallScript` projects only.

This system variable is set by the [OnSetUpdateMode](#) event handler function and is used by the `OnShowUI` event handler to call the appropriate UI event handlers.

## WINDIR

The `WINDIR` system variable contains the fully qualified name of the folder that contains the main operating environment, for example `C:\Windows`.

This system variable is read-only; if you attempt to assign a value to it, a compiler error results. The value of this system variable is shared among object scripts and between object scripts and the main setup script.



**Project** • During setup initialization in `InstallScript` installations, the value of the `WINDIR` variable is obtained by calling the Windows API function **`GetWindowsDirectory`**.

In Basic MSI and `InstallScript MSI` installations, the value of the `WINDIR` variable is initialized based on the Windows Installer property **`WindowsFolder`**. Note that deferred, commit, and rollback custom actions do not have access to this property. Therefore, the corresponding `WINDIR` variable is empty in deferred, commit, and rollback custom actions. To learn more, see *Accessing or Setting Windows Installer Properties Through Deferred, Commit, and Rollback Custom Actions*.

## WINDISK

The `WINDISK` system variable contains the ID of the disk drive that contains the main operating environment. This drive is the one that contains the Windows program—usually drive C. Note that the `InstallScript` engine includes the colon (:) with the drive letter.



**Note** • If you intend to reference the root folder of the drive specified by this variable, you must append a backslash to it (specified as two backslashes). For example, if the value of `WINDISK` is `C:`, the following statement refers to the root folder of that drive: `WINDISK + "\\`.



**Project** • During setup initialization in InstallScript installations, the value of the `WINDIR` variable is obtained by calling the Windows API function **SHGetSpecialFolderPath** with the `CSIDL_WINDOWS` parameter.

In Basic MSI and InstallScript MSI installations, the value of the `WINDIR` variable is obtained by calling the InstallScript function **GetDisk** with `WINDIR`; if that fails, this variable is initialized based on the Windows Installer property **WindowsVolume**. Note that deferred, commit, and rollback custom actions do not have access to this property. Therefore, the corresponding `WINDIR` variable is empty in deferred, commit, and rollback custom actions. To learn more, see *Accessing or Setting Windows Installer Properties Through Deferred, Commit, and Rollback Custom Actions*.

## WINSYSDIR

The `WINSYSDIR` system variable contains the name of the System32 folder. This folder is used to store application extensions (DLLs), device drivers, and other Windows system files, depending on the version of Windows.

On 64-bit Windows systems, this variable points to the folder that stores Windows system files that are used by 32-bit applications. This folder is named `SysWOW64`. There is a different Windows system folder for system files to be used by 64-bit applications; the system variable `WINSYSDIR64` provides access to this folder.



**Project** • During setup initialization in InstallScript installations on 32-bit systems, the value of the `WINSYSDIR` variable is obtained by calling the Windows API **GetSystemFolder**. During setup initialization in InstallScript installations on 64-bit systems, the value of the `WINSYSDIR` variable is obtained by calling the Windows API **GetSystemWow64Directory** from a 64-bit executable file.

In Basic MSI and InstallScript MSI installations on 32-bit systems, the value of the `WINSYSDIR` variable is initialized based on the Windows Installer property **SystemFolder**. On 64-bit systems, the value is initialized based on the Windows Installer property **System64Folder**. Note that deferred, commit, and rollback custom actions do not have access to this property. Therefore, the corresponding `WINSYSDIR` variable is empty in deferred, commit, and rollback custom actions. To learn more, see *Accessing or Setting Windows Installer Properties Through Deferred, Commit, and Rollback Custom Actions*.

## WINSYSDIR64

The `WINSYSDIR64` system variable contains the name of the 64-bit System32 folder. This folder is used to store application extensions (DLLs), device drivers, and other Windows system files, depending on the version of Windows.

Although the `WINSYSDIR64` variable is set to the 64-bit System32 folder, 64-bit Windows includes functionality to automatically redirect 32-bit applications (such as the InstallScript engine) to the 32-bit `SysWOW64` folder. Therefore, if you are using InstallScript code to read or write to `WINSYSDIR64`, you may need to first disable file system redirection using the constant `WOW64FSREDIRECTION` with the functions **Disable** and **Enable**. Otherwise, reading from and writing to `WINSYSDIR64` are incorrectly redirected to the 32-bit `SysWOW64` folder. Since some

Windows functionality that could be used by the installation requires that redirection be enabled to work, Windows documentation recommends that you disable redirection only for as long as necessary. It is recommended that you then enable file system redirection as soon as you have completed reading from or writing to WINSYSDIR64.

The example code below shows how to disable and enable redirection before and after transferring a file to the WINSYSDIR64 through script:

```
Disable (WOW64FSREDIRECTION);
XCopyFile (SUPPORTDIR ^ "MyFile.dll", WINSYSDIR64, COMP_NORMAL);
Enable (WOW64FSREDIRECTION);
```



**Project** • InstallScript projects have support for installing files to the 64-bit System32 folder without requiring any script modifications with WOW64FSREDIRECTION. If you have files that need to be installed to this location, you can add the files and registry data to a component, and select Yes for that component's 64-Bit Component setting. At run time, the installation automatically disables file system redirection for the System32 files. To learn more, see *Targeting 64-Bit Operating Systems with InstallScript Installations*.

During setup initialization in InstallScript installations on 64-bit systems, the value of the WINSYSDIR64 variable is obtained by calling the Windows API **GetSystemFolder** from a 64-bit executable file.

In Basic MSI and InstallScript MSI installations on 64-bit systems, the value of the WINSYSDIR64 variable is initialized based on the Windows Installer property **System64Folder**. Note that deferred, commit, and rollback custom actions do not have access to this property. Therefore, the corresponding WINSYSDIR variable is empty in deferred, commit, and rollback custom actions. To learn more, see *Accessing or Setting Windows Installer Properties Through Deferred, Commit, and Rollback Custom Actions*.

## WINSYSDISK

The WINSYSDISK system variable contains the name of the disk drive that contains the Windows system folder—usually the C: drive. This folder is used to store application extensions (DLLs), device drivers, and other Windows system files, depending on the version of Windows. Note that the InstallScript engine includes the colon (:) with the drive letter. For information about the Windows system folder, see the documentation for the InstallScript system variable [WINSYSDIR](#).



**Note** • If you intend to reference the root folder of the drive specified by this variable, you must append a backslash to it (specified as two backslashes). For example, if the value of WINSYSDISK is C:, the following statement refers to the root folder of that drive: WINSYSDISK + "\\".



**Project** • During setup initialization in InstallScript installations, the value of the WINSYSDIR variable is obtained by calling the Windows API **GetSystemFolder**.

In Basic MSI and InstallScript MSI installations, the value of the WINSYSDIR variable is obtained by calling the InstallScript function **GetDisk** with WINSYSDIR; if that fails, this variable is initialized based on the Windows Installer property **WindowsVolume**. Note that deferred, commit, and rollback custom actions do not have access to this property. Therefore, the corresponding WINSYSDIR variable is empty in deferred, commit, and rollback custom actions. To learn more, see *Accessing or Setting Windows Installer Properties Through Deferred, Commit, and Rollback Custom Actions*.



# Preprocessor Directives

Preprocessor directives are instructions to the InstallScript compiler that are executed as the script is compiled. Preprocessor directives can instruct the compiler to include other source files in the compilation, to define constants, to include or exclude statements based on compile-time conditions, and to display a user-defined error message. Although InstallScript directives are similar to directives in the C language, they are not exactly alike.

Preprocessor directives begin with a pound sign (#) and can be inserted anywhere in a script. Each directive must appear on a line by itself and must *not* be terminated with a semicolon.

## Using Preprocessor Directives

Use the following guidelines when using preprocessor directives in your script. A preprocessor directive meets the following conditions:

- It does not end with a semicolon.
- It cannot line wrap.
- It must be less than 250 characters in length.



**Note** • The expressions used in conditional directives can include constants that have been defined with the `#define` directive. They cannot include variables.

## Using Boolean Operators with Preprocessor Directives

The following Boolean operators are supported in `#if`, `#ifdef`, `#ifndef`, and `#elif` statements:

- Logical OR ( `||` )
- Logical AND ( `&&` )
- Relational ( `=`, `!=`, `>`, `>=`, `<`, `<=` )

## InstallScript-Supported Preprocessor Directives

InstallScript supports the following preprocessor directives:

**Table 1** • Preprocessor Directives

Preprocessor Directive	Description
<code>#define</code>	Creates a symbolic constant.
<code>#elif</code>	Combines <code>#else</code> and <code>#if</code> into one statement.
<code>#else</code>	Indicates alternatives if the test fails.
<code>#endif</code>	Ends a preprocessor conditional directive ( <code>#if</code> , <code>#ifdef</code> , <code>#ifndef</code> ).
<code>#error</code>	Displays a user-defined error message.
<code>#if</code>	Compiles if the conditional statement is true.

**Table 1 •** Preprocessor Directives

Preprocessor Directive	Description
<b>#ifdef</b>	Compiles if a numeric constant is defined.
<b>#ifndef</b>	Compiles if a numeric constant is not defined.
<b>#include</b>	Includes the contents of another file.
<b>#undef</b>	Undefines a constant that was defined previously with #define.
<b>#warning</b>	Display a compiler warning and a user-defined error message.

## #define

Use #define to define a number or string constant. When you define a constant and assign it a value, InstallShield replaces the constant with that value wherever it appears. For example, the following #define statement sets the value of MAX\_SIZE to 145:

```
#define MAX_SIZE 145
```

The following example declares a string constant with the #define directive:

```
#define STR_MESSAGE "This is a message."
```

Once you have defined STR\_MESSAGE, you can use it anywhere in the script. A string message you want displayed in SprintfBox or MessageBox cannot be longer than 255 characters. If you want to display more than 255 characters, split your string into two or more parts before displaying it. The 255-character limit includes spaces, escape sequences, and other special characters.



**Note •** Another way to define constants is in the Preprocessor Defines field on the Compile tab of the Setup Settings dialog box. If you add or change a preprocessor define in the Setup Settings dialog box, you must recompile your setup before the changes will take effect.

## Restrictions

There are a few restrictions regarding the #define directive:

- InstallShield supports the use of #define to define only macros that involve the simple lexical substitution of a number or a string. You cannot define macros with expressions using multiple terms or operators.
- Constants you declare with a #define statement cannot begin with numbers.
- Many InstallShield functions use predefined constants. If you try to define one of the predefined constants, the InstallShield Script Compiler generates an error message.
- InstallShield assigns a value of zero (0) to an undefined constant.

## #elif

The `#elif` compile-time statement is similar in function to the `elseif` run-time statement. It combines the `#if` and `#else` statements, allowing you to specify another condition. For example:

```
#if (A = 1)
    // compile if A equals 1
    . . .
#elif (A = 2)
    // compile if A equals 2
    . . .
#elif (B = 3)
    // compile if B equals 3
    . . .
#else
    // if none of the #elif conditions are true, compile
    // the following portion
    . . .
#endif
```




---

**Note** • When using `#elif`, end the section with only one `#endif`.

## #error

Use the `#error` directive to abort compilation and display a user-defined error message. The message to display must be entered immediately after `#error` and be separated from the directive by at least one space.

In the example below, the constant `PRODUCTID` must equal 1 or 2. If the value of `PRODUCTID` is not within that range, the constant `PRODUCTNAME` is not defined and a user-defined error message is displayed.

```
#define PRODUCTID 1

#if (PRODUCTID = 1)
    #define PRODUCTNAME "Lite"
#elif (PRODUCTID = 2)
    #define PRODUCTNAME "Professional"
#endif

#ifndef PRODUCTNAME
    #error PRODUCTID out of range.
#endif
```

## #if...#else...#endif

Use the `#if` statement to select which lines to compile. You can switch various sections of the installation on and off, making the script more flexible. The `#if` statement works in the same manner as the run-time `if` statement:

```
#if (A = 1)
    // compile if A equals 1
    . . .
#else
    // compile
    . . .
```

#endif

Keep in mind the following restrictions when using #if:

- Note that the format of the #if statement is also identical to that of the run-time if statement; you must end the #if statement with the keyword #endif.
- You can test only numeric constants with an #if or #elif statement.
- InstallScript allows nesting of #if statements up to a maximum of 10 levels.

## #ifdef and #ifndef

Use the #ifdef statement when you want to compile a section only if a specified expression has been defined with #define. Use #ifndef when you want to compile a section only if a specified expression has not been defined.

### Example

```
#ifdef A
    // Compile if A is defined.
    . . .
#endif

#ifndef A
    // Skip if A is defined; otherwise compile.
    . . .
#endif

#ifdef A    // Compile if A is defined.    . . .#endif#ifndef A    // Skip if A is defined; otherwise
compile.    . . .#endif
```

You can also use #ifdef and #ifndef with #else and #elif:

```
#ifdef nFilePath
    //statements
#else
    //statements
#endif
```



**Note** • Preprocessor defines can be entered in the Project Settings dialog box's Compile/Link tab's Preprocessor Defines edit box. If you add or change a preprocessor definition in the Project Settings dialog box, you must recompile your setup for the changes to take effect.

### Restrictions

Keep in mind the following restrictions when using #ifdef and #ifndef statements:

- You cannot place comments on the same line as #ifdef and #ifndef directives.
- Do not test a constant that has a value of 0 (zero) with an #ifdef or #ifndef statement.
- You can test only numeric constants with an #ifdef or #ifndef statement.



# #include

Use the `#include` statement to include the contents of another script in the main installation script. When you use `#include`, the compiler treats the additional source script as if it were part of the main installation script. Additional scripts, or include files, may contain variable declarations, other compiler directives, and program statements.

For example, you can create a separate file that contains all the user-defined constant definitions and then insert it into the script file using the `#include` statement. If you need to redefine any of the constants at a later date, they are all in one central location.

When you compile from the InstallShield interface, InstallShield searches for include script files in the following order:

1. The project's script file directory
2. The directories that are specified in the Include Paths setting on the Compile/Link tab of the Settings dialog box
3. The InstallShield include directory

If two include files share the same name but they are stored in different locations, InstallShield links to the one that it finds first, according to the aforementioned order. In addition, if an include file is specified by a relative path, InstallShield searches for the path relative to the aforementioned directories, in order.

If the include file is not in any of these locations, specify a fully qualified file name in the `#include` statement. When using the `#include` statement, specify a file name or path by enclosing the file name or path in double quotation marks ("filename").

Note the following details when you are using `#include` directives in your script:

- InstallShield does not handle paths with more than 260 characters, including the file name.
- InstallScript allows nesting of include files up to a maximum of eight levels.
- The InstallShield preprocessor does not interpret a backslash character in an `#include` directive as a control character; when specifying a path, use a single backslash instead of double backslashes to separate folder names.
- Do not include C language header files in the script. The InstallShield compiler does not recognize some of C's constructs. Create header files using InstallScript only.

The following example shows a section of an installation script that uses the `#include` statement to include the contents of **Support.rul** and other files. Each of the source scripts referenced by the `#include` statements has been written for a specific purpose and is then added to the script when the script is compiled:

```
//The following include file contains installation-specific routines.
#include "SUPPORT.RUL"
```

```
// Local include file containing variable and prototype declarations.
#include "DECLARE.RUL"
```

```
// Include scripts from the LIBRARY directory
#include "..\LIBRARY\SYCHK.H"
```

```
// Include scripts from the DIALOGS directory.
#include "..\DIALOGS\WELCOME\WELCOME.H"
#include "..\DIALOGS\REGINS\REGINS.H"
#include "..\DIALOGS\ICONS\ICONS.H"
```

## #undef

Use the `#undef` directive to undefine a constant that was previously defined with `#define`. If you specify a constant with this directive that was not previously defined with the `#define` directive, an error will occur when you attempt to compile your script. In the following example, two constants are defined; then, the first constant is undefined if the second has been defined:

```
#define NORMSETUP
#define BONUSPAK

. . .

#ifdef NORMSETUP
    MessageBox('Compiling for normal setup .',INFORMATION);
#else
    MessageBox('Compiling for super setup.',INFORMATION);
#endif

#ifdef BONUSPAK
    #undef NORMSETUP
#endif

#ifdef NORMSETUP
    MessageBox('Compiling for normal setup .',INFORMATION);
#else
    MessageBox('Compiling for super setup.',INFORMATION);
#endif
```

## #warning

Use the `#warning` directive to display a compiler warning and a user-defined error message. The message to be displayed must be entered immediately after `#warning` and be separated from the directive by at least one space.

In the example below, the constant `PRODUCTID` must equal 1 or 2. If the value of `PRODUCTID` is not within that range, the constant `PRODUCTNAME` is not defined and a user-defined warning message is displayed.

```
#define PRODUCTID 1
#if (PRODUCTID = 1)
    #define PRODUCTNAME "Lite"
#elif (PRODUCTID = 2)
    #define PRODUCTNAME "Professional"
#endif
#ifndef PRODUCTNAME
    #warning PRODUCTID out of range.
#endif
```

# Event Handlers



**Project** • InstallScript event handlers are available in the following project types:

- InstallScript
- InstallScript MSI

Some exceptions exist for an InstallScript package that is included in an Advanced UI or Suite/Advanced UI installation. These are described below.

## About Event Handlers

InstallScript installations are driven by the InstallScript engine, which generates a series of events in a specific order. These events trigger software handlers that execute installation instructions. For example, shortly after an installation is loaded, it generates an event called Begin, which triggers the execution of an event handler called OnBegin. This handler specifies the instructions to carry out when the Begin event occurs. Other events occur in the installation to trigger other handlers. Together, the event handlers perform the work of installing the application.

A portion of an InstallScript MSI installation is driven by the InstallScript engine, and a portion is driven by the Windows Installer engine. The portion that is driven by the InstallScript engine uses a series of event handlers, similar to InstallScript installations.

Several types of event handlers are available:

- [Global Event Handlers](#)
- [Feature Event Handlers](#)
- [Miscellaneous Event Handlers](#)
- [Advanced Event Handlers](#)

When you create an InstallScript or InstallScript MSI project, InstallShield generates a set of default global event handlers, each of which is a function that is written in the InstallScript language. Likewise, when you add features to your project, InstallShield generates a set of default event handlers for that feature. You can override or customize any or all of the event handlers.

It is important to note that, in an event-driven script, event-handler functions are called even if they do not explicitly appear in the InstallScript view.

## Order of Event Handlers

Global and feature event handlers are always called in a specific order; which event handlers are called depends on the type of installation (normal installation, maintenance installation, administrative installation, or patch installation). Because miscellaneous event handlers respond to events that may not happen during installation, they are not necessarily called in a specific order, if they are called at all.



**Project** • When an InstallScript MSI major upgrade is uninstalling an earlier version of a product, none of the InstallScript event handlers are called.

### First-Time Installations

- [OnBegin](#)
- [OnCCPSearch](#)
- [OnAppSearch](#)
- [OnFirstUIBefore](#)
- [OnGeneratingMSIScript](#) (InstallScript MSI only)
- [OnMoving](#)
- feature Installing events
- [OnInstallFilesActionBefore](#)
- [OnGeneratedMSIScript](#) (InstallScript MSI only)
- [OnInstallFilesActionAfter](#)
- feature Installed events
- [OnMoved](#)
- [OnFirstUIAfter](#)
- [OnEnd](#)

### Resumed Installations

- [OnResumeUIAfter](#) (InstallScript MSI only)
- [OnResumeUIBefore](#) (InstallScript MSI only)

### Maintenance Installations

- [OnBegin](#)
- [OnMaintUIBefore](#)
- [OnGeneratingMSIScript](#) (InstallScript MSI only)
- [OnMoving](#)
- feature Installing or Uninstalling events
- [OnInstallFilesActionBefore](#)
- [OnInstallFilesActionAfter](#)
- feature Installed or Uninstalled events
- [OnMoved](#)
- [OnGeneratedMSIScript](#) (InstallScript MSI only)
- [OnMaintUIAfter](#)
- [OnEnd](#)

## Patch Installations

- [OnPatchUIBefore](#) (InstallScript MSI only)
- [OnGeneratingMSIScript](#) (InstallScript MSI only)
- [OnMoving](#)
- feature Installing or Uninstalling events
- [OnInstallFilesActionBefore](#)
- [OnInstallFilesActionAfter](#)
- feature Installed or Uninstalled events
- [OnMoved](#)
- [OnGeneratedMSIScript](#) (InstallScript MSI only)
- [OnPatchUIAfter](#) (InstallScript MSI only)

## Exceptions for an InstallScript Package That Is Included in an Advanced UI or Suite/ Advanced UI Installation

If you include an InstallScript installation as an InstallScript package in an Advanced UI or Suite/Advanced UI project, the Advanced UI or Suite/Advanced UI installation displays its own user interface (UI) while automatically suppressing the UI of the InstallScript package. To make these changes possible, the Advanced UI or Suite/Advanced UI installation uses several Advanced UI– or Suite/Advanced UI–specific InstallScript events and functions by default, and ignores some of the standard InstallScript events and functions. For more information, see [Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project](#).

Thus, depending on the installation state (first-time installation, maintenance, or update), [OnSuiteShowUI](#) ignores the UI events such as [OnFirstUIBefore](#) and [OnFirstUIAfter](#) and instead calls the following events:

- **First-time installation**—[OnSuiteInstallBefore](#), [OnSuiteInstallAfter](#)
- **Maintenance**—[OnSuiteMaintBefore](#), [OnSuiteMaintAfter](#)
- **Update**—[OnSuiteUpdateBefore](#), [OnSuiteUpdateAfter](#)

All other events and event call sequencing in an InstallScript package that is launched from an Advanced UI or Suite/Advanced UI installation remain the same as in an InstallScript installation that is launched separately from an Advanced UI or Suite/Advanced UI installation, or that is launched as an executable package from an Advanced UI or Suite/Advanced UI installation.

# Event Handler Index

Event handlers are InstallScript functions that are called in response to events that occur during setup. The names of these handlers are reserved. (It is not necessary to prototype these functions in your script.)



**Project** • Some of the following event handlers only apply to InstallScript or InstallScript MSI projects. Some apply to both types. Refer to the Project Type column to determine which projects are supported.

**Table 1** • Event Handlers Index

Event Handler	Project Type
<b>OnAbort</b>	InstallScript, InstallScript MSI
<b>OnAdminInstallUIAfter</b>	InstallScript MSI
<b>OnAdminInstallUIBefore</b>	InstallScript MSI
<b>OnAdminPatchUIAfter</b>	InstallScript MSI
<b>OnAdminPatchUIBefore</b>	InstallScript MSI
<b>OnAdvertisementAfter</b>	InstallScript MSI
<b>OnAdvertisementBefore</b>	InstallScript MSI
<b>OnAppSearch</b>	InstallScript, InstallScript MSI
<b>OnBegin</b>	InstallScript, InstallScript MSI
<b>OnCanceling</b>	InstallScript, InstallScript MSI
<b>OnCCPSearch</b>	InstallScript, InstallScript MSI
<b>OnCheckMediaPassword</b>	InstallScript
<b>OnComponentError</b>	InstallScript, InstallScript MSI
<b>OnCustomizeUninstInfo</b>	InstallScript
<b>OnDIFxLogCallback</b>	InstallScript
<b>OnEnd</b>	InstallScript, InstallScript MSI
<b>OnError</b>	InstallScript MSI
<b>OnException</b>	InstallScript MSI
<b>OnFileError</b>	InstallScript
<b>OnFileLocked</b>	InstallScript
<b>OnFileReadOnly</b>	InstallScript
<b>OnFilesInUse</b>	InstallScript MSI

**Table 1 •** Event Handlers Index (cont.)

Event Handler	Project Type
<b>OnFilterComponents</b>	InstallScript, InstallScript MSI
<b>OnFirstUIAfter</b>	InstallScript, InstallScript MSI
<b>OnFirstUIBefore</b>	InstallScript, InstallScript MSI
<b>OnGeneratedMSIScript</b>	InstallScript MSI
<b>OnGeneratingMSIScript</b>	InstallScript MSI
<b>OnHelp</b>	InstallScript, InstallScript MSI
<b>OnIISComponentInstalled</b>	InstallScript, InstallScript MSI
<b>OnIISInitialize</b>	InstallScript, InstallScript MSI
<b>OnIISUninitialize</b>	InstallScript, InstallScript MSI
<b>OnIISVRootUninstalling</b>	InstallScript, InstallScript MSI
<b>OnInstalled</b>	InstallScript, InstallScript MSI
<b>OnInstalledFile</b>	InstallScript
<b>OnInstalledFontFile</b>	InstallScript
<b>OnInstallFilesActionAfter</b>	InstallScript, InstallScript MSI
<b>OnInstallFilesActionBefore</b>	InstallScript, InstallScript MSI
<b>OnInstalling</b>	InstallScript, InstallScript MSI
<b>OnInstallingFile</b>	InstallScript
<b>OnInternetError</b>	InstallScript
<b>OnLaunchAppAndWaitCallback</b>	InstallScript, InstallScript MSI
<b>OnLogonUserSetMsiProperties</b>	InstallScript MSI
<b>OnMaintUIAfter</b>	InstallScript, InstallScript MSI
<b>OnMaintUIBefore</b>	InstallScript, InstallScript MSI
<b>OnMD5Error</b>	InstallScript
<b>OnMoved</b>	InstallScript, InstallScript MSI

**Table 1 •** Event Handlers Index (cont.)

Event Handler	Project Type
<b>OnMoveData</b>	InstallScript
<b>OnMoving</b>	InstallScript, InstallScript MSI
<b>OnMsiSilentInstall</b>	InstallScript MSI
<b>OnNetApiCreateUserAccount</b>	InstallScript, InstallScript MSI
<b>OnNextDisk</b>	InstallScript
<b>OnOutOfDiskSpace</b>	InstallScript MSI
<b>OnPatchUIAfter</b>	InstallScript MSI
<b>OnPatchUIBefore</b>	InstallScript MSI
<b>OnRebooted</b>	InstallScript  InstallScript MSI—if the InstallScript user interface (UI) style is the traditional style (which uses the InstallScript engine as an external UI handler)
<b>OnRemovingSharedFile</b>	InstallScript
<b>OnResumeUIAfter</b>	InstallScript MSI
<b>OnResumeUIBefore</b>	InstallScript MSI
<b>OnRMFilesInUse</b>	InstallScript MSI
<b>OnSelfRegistrationError</b>	InstallScript
<b>OnSetTARGETDIR</b>	InstallScript
<b>OnSetUpdateMode</b>	InstallScript
<b>OnShowUI</b>	InstallScript
<b>OnSuiteInstallAfter</b>	InstallScript
<b>OnSuiteInstallBefore</b>	InstallScript
<b>OnSuiteMaintAfter</b>	InstallScript
<b>OnSuiteMaintBefore</b>	InstallScript
<b>OnSuiteUpdateAfter</b>	InstallScript



**Table 1 •** Event Handlers Index (cont.)

Event Handler	Project Type
<b>OnSuiteUpdateBefore</b>	InstallScript
<b>OnSuiteShowUI</b>	InstallScript
<b>OnSQLBatchScripts</b>	InstallScript
<b>OnSQLComponentInstalled</b>	InstallScript
<b>OnSQLComponentUninstalled</b>	InstallScript
<b>OnSQLLogin</b>	InstallScript
<b>OnSQLServerInitialize</b>	InstallScript
<b>OnSQLServerInitializeMaint</b>	InstallScript
<b>OnUninstall</b>	InstallScript, InstallScript MSI
<b>OnUnInstalled</b>	InstallScript, InstallScript MSI
<b>OnUninstalledFile</b>	InstallScript
<b>OnUnInstalling</b>	InstallScript, InstallScript MSI
<b>OnUninstallingDIFxDriverFile</b>	InstallScript
<b>OnUninstallingFile</b>	InstallScript
<b>OnUninstallingFontFile</b>	InstallScript
<b>OnUpdateUIAfter</b>	InstallScript
<b>OnUpdateUIBefore</b>	InstallScript
<b>OnWarning</b>	InstallScript MSI
<b>OnXMLComponentInstalled</b>	InstallScript
<b>OnXMLComponentUninstalling</b>	InstallScript
<b>OnXMLInitialize</b>	InstallScript
<b>OnXMLUninitialize</b>	InstallScript

## Component Event Handlers

Windows Installer installations use features—not components—as the highest level of organization in your project. Therefore, component event handlers have been replaced with feature event handlers. In addition, all supported component functions now have feature equivalents that should be used. For more information, see [Feature Functions](#) and [Feature Event Handlers](#).

## Global Event Handlers

Global event handlers carry out processes required before and after feature installation and uninstallation. They include event handlers in the following categories:

**Table 2 • Global Event Handlers Categories**

Category	Description
<b>Initialization Handlers</b>	Event handlers that are called directly by the installation engine.
<b>Before Move Data Handlers</b>	Event handlers that are triggered before features are installed or uninstalled on the target computer.
<b>Move Data Handlers</b>	Event handlers that are triggered immediately before and immediately after features are installed or uninstalled on the target computer.
<b>After Data Move Handlers</b>	Event handlers that are triggered after features are installed or uninstalled on the target computer.

## Initialization Handlers

The following event handlers are called directly by the installation engine in InstallScript projects:

**Table 3 • Initialization Handlers**

Event Handler	Project Type	Description
<b>OnCheckMediaPassword</b>	InstallScript	Called directly by the framework during initiation to query the end user for the installation media's password if the password is not already in the installation log file (as it is if the installation is running as a maintenance installation or uninstallation) and you selected the "Show Password dialog box during setup initialization" check box in the Release Wizard's Password panel or set the Show Password Dialog property to Yes in the Releases view.
<b>OnFilterComponents</b>	InstallScript, InstallScript MSI	Called directly by the framework to filter out components in each feature by language and platform. Override this event to perform custom filtering.
<b>OnIISCheckRequirements</b>	InstallScript	This event handler is obsolete.

**Table 3 • Initialization Handlers**

Event Handler	Project Type	Description
<b>OnSetTARGETDIR</b>	InstallScript	Called directly by the framework during initialization to initialize TARGETDIR to its default value.
<b>OnSetUpdateMode</b>	InstallScript	Called directly by the framework during initialization to set the UPDATEMODE system variable appropriately to control which UI events are called by OnShowUI.

## OnCheckMediaPassword



**Project** • This information applies to InstallScript projects.

The OnCheckMediaPassword event handler function is called directly by the setup engine. The handler's default code queries the end user for the setup's password if the password is not already in the setup log file (as it is if the setup is running as a maintenance setup or uninstallation) and you checked the "Show Password dialog box during setup initialization" check box in the Release Wizard's Password panel or set Show Password Dialog to Yes in the Release property sheet.

This event handler is called (when appropriate) in any setup, including a setup that uses a procedural script (a script with a program...endprogram block).

### Syntax

```
OnCheckMediaPassword ( );
```

### Parameters

None.

### Return Values

None.

## OnFilterComponents



**Project** • This information applies to the following project types:

- InstallScript
- InstallScript MSI

The OnFilterComponents event handler is called directly by the InstallScript engine to handle feature filtering—that is, the including and excluding of features' components in the file transfer based on their Language and Operating System settings.

## Syntax

```
OnFilterComponents ( );
```

## Parameters

None.

## Return Values

None.

## Additional Information

By default, OnFilterComponents calls [FeatureFilterLanguage](#) to exclude all components with languages other than the one specified by the [SELECTED\\_LANGUAGE](#) system variable, and calls [FeatureFilterOS](#) to exclude all components with operating systems other than the one specified by the [SYSINFO](#) variable's nISOSL member.

This event handler is not called in an installation that uses a procedural script (a script with a program...endprogram block).

## OnSetTARGETDIR



**Project** • This information applies to InstallScript projects.

The OnSetTARGETDIR event handler function is called directly by the setup engine to set the value of the system variable TARGETDIR.

This event handler is called in any setup, including a setup that uses a procedural script (a script with a program...endprogram block).

## Syntax

```
OnSetTARGETDIR ( );
```

## Parameters

None.

## Return Values

None.

## Additional Information

- By default, in a first installation OnSetTARGETDIR sets TARGETDIR to the value that you specified in the TARGETDIR setting in the General Information view, or—if you did not specify a value in InstallShield—to **<FOLDER\_APPLICATIONS>\<IFX\_COMPANY\_NAME>\<IFX\_PRODUCT\_NAME>**. <FOLDER\_APPLICATIONS>, <IFX\_COMPANY\_NAME>, and <IFX\_PRODUCT\_NAME> are texts substitutions whose values are resolved when TARGETDIR is referenced; that is, if you change the value of the system variable IFX\_COMPANY\_NAME or IFX\_PRODUCT\_NAME after OnSetTARGETDIR has been called, that change is reflected in subsequent references to TARGETDIR.

- A maintenance installation or uninstallation initializes TARGETDIR to the value that is stored in the log file, and by default OnSetTARGETDIR does not modify the value of TARGETDIR.

## OnSetUpdateMode



**Project** • This information applies to InstallScript projects.

The OnSetUpdateMode event handler function is called directly by the setup engine to determine whether the setup is an update to an existing installation and set the value of the system variable UPDATEMODE accordingly.

This event handler is called (when appropriate) in any setup, including a setup that uses a procedural script (a script with a program...endprogram block).

### Syntax

```
OnSetUpdateMode ( );
```

### Parameters

None.

### Return Values

None.



## Before Move Data Handlers

The following event handlers are triggered before files are transferred to the target computer. Most of these events are also triggered during a maintenance setup.





**Table 4 • Before Move Data Handlers**

Event Handler	Project Type	Description
<b>OnBegin</b>	InstallScript	In InstallScript projects: Called directly by the framework after Initialization events.
	InstallScript MSI	In InstallScript MSI projects: Responds to the Begin event, the first redefinable event in a setup.
<b>OnAppSearch</b>	InstallScript	In InstallScript projects: Called directly by the framework after OnBegin.
	InstallScript MSI	In InstallScript MSI projects: Responds to the Application Search event. Code this handler in installations that must search for a specific application on the target computer.



**Table 4 •** Before Move Data Handlers

Event Handler	Project Type	Description
<b>OnCCPSearch</b>	InstallScript InstallScript MSI	<p>In InstallScript projects: Called directly by the framework after AppSearch.</p> <p>In InstallScript MSI projects: Responds to the Upgrade Compliance event. Code this handler in installations that must search for an application whose presence qualifies the end user to install your application.</p>
<b>OnFirstUIBefore</b>	InstallScript InstallScript MSI	<div>  <p><b>Note •</b> <i>If the InstallScript installation is included in an Advanced UI or Suite/Advanced UI project as an InstallScript package, the Advanced UI or Suite/Advanced UI installation does not call this event handler. For more information, see <a href="#">Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project</a>.</i></p> </div> <p>In InstallScript projects: Called by the framework when the setup is running in first install mode. By default, this event displays UI allowing the end user to specify installation parameters.</p> <p>In InstallScript MSI projects: Responds to the First UI Before event by displaying dialogs that gather information from the end user for the first installation of an application.</p>
<b>OnMaintUIBefore</b>	InstallScript InstallScript MSI	<div>  <p><b>Note •</b> <i>If the InstallScript installation is included in an Advanced UI or Suite/Advanced UI project as an InstallScript package, the Advanced UI or Suite/Advanced UI installation does not call this event handler. For more information, see <a href="#">Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project</a>.</i></p> </div> <p>In InstallScript projects: Called by OnShowUI when the installation is running in maintenance mode. OnShowUI can be customized to control whether this event is called.</p> <p>In InstallScript MSI projects: Responds to the MaintenanceUIBefore event by displaying dialogs that gather information from the end user for a maintenance installation of an application.</p>

**Table 4 •** Before Move Data Handlers

Event Handler	Project Type	Description
<b>OnUpdateUIBefore</b>	InstallScript	 <p><b>Note •</b> If the InstallScript installation is included in an Advanced UI or Suite/Advanced UI project as an InstallScript package, the Advanced UI or Suite/Advanced UI installation does not call this event handler. For more information, see <i>Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project</i>.</p> <p>Called by OnShowUI when the setup is running in update mode. By default this event displays UI that allows the end user to update the application to the current version.</p>
<b>OnSuiteInstallBefore</b>	InstallScript	<p>The OnSuiteShowUI event calls the OnSuiteInstallBefore event. By default, OnSuiteInstallBefore is called to initialize information, such as the selection of features, that is necessary to perform file transfer.</p>  <p><b>Note •</b> This event is not called automatically in a program...endprogram style installation.</p>
<b>OnSuiteMaintBefore</b>	InstallScript	<p>The OnSuiteShowUI event calls the OnSuiteMaintBefore event. By default, OnSuiteMaintBefore is called to initialize information, such as the selection of features, that is necessary to perform file transfer.</p>  <p><b>Note •</b> This event is not called automatically in a program...endprogram style installation.</p>
<b>OnSuiteUpdateBefore</b>	InstallScript	<p>The OnSuiteShowUI event calls the OnSuiteUpdateBefore event. By default, OnSuiteUpdateBefore is called to initialize information, such as the selection of features, that is necessary to perform file transfer.</p>  <p><b>Note •</b> This event is not called automatically in a program...endprogram style installation.</p>
<b>OnSQLLogin</b>	InstallScript MSI	<p>In InstallScript MSI projects: Responds to the First UI Before event. This event handler function creates a dialog which is used by the script to specify SQL login credentials. These credentials include the login ID and password.</p>

**Table 4 •** Before Move Data Handlers

Event Handler	Project Type	Description
<b>OnSQLServerInitialize</b>	InstallScript	Called by OnFirstUIBefore to establish any connections necessary for SQL Server support. This function will initialize the SQL Server runtime and attempt to make any necessary SQL Server connections displaying a login dialog for each one.
<b>OnSQLServerInitializeMain</b> <b>t</b>	InstallScript	Called by OnMaintUIBefore to establish any connections necessary for SQL Server support.
<b>OnIISInitialize</b>	InstallScript	The OnIISInitialize event is called by OnFirstUIBefore to initialize the IIS run time.   <b>Note •</b> This event is not called automatically in a program...endprogram style installation.
<b>OnXMLInitialize</b>	InstallScript	The OnXMLInitialize event is called by OnFirstUIBefore to initialize the XML runtime.   <b>Note •</b> This event is not called automatically in a program...endprogram style installation.

## OnAppSearch



**Project •** This information applies to the following project types:

- InstallScript
- InstallScript MSI

The OnAppSearch event handler responds to the Application Search event. Code this handler in setups that must search for a specific application on the target computer. Your code might, for example, call FindFile to locate a key file, or RegDBKeyExist to search for a registry entry.

The following OnAppSearch function aborts the installation if a file called Notepad.exe is not available in the user's Windows or WinNT folder, or in a subdirectory of it.

```
function OnAppSearch( )
    NUMBER nResult;
    STRING svIgnore;
begin
    nResult =
        FindAllFiles(WindowsFolder, "Notepad.exe",
                    svIgnore, RESET);

    if (nResult < 0) then
        MessageBox("Unable to find a qualifying program. " +
```



```

        "Setup will now exit.", SEVERE);
    abort;
endif;
end;

```



**Note** • This event handler is not executed during a maintenance setup or uninstallation.

## OnBegin



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The OnBegin event handler responds to the Begin event, the first redefinable event in a setup. Code that must be executed before anything else in the script should be included in this handler. For example, you can verify that the user's machine meets your product's system requirements here.

OnBegin is prototyped for you when you include iswi.h or ifx.h in your script. Define OnBegin in your script as in the following example:

```
#include "iswi.h"
```

```

function OnBegin( )
    // local variables
begin
    // beginning code
end;

```

For example, an OnBegin function that verifies a particular registry key exists before continuing the installation might appear as follows:

```

// abort installation if HKLM\Software\InstallShield does not exist
function OnBegin( )
    NUMBER nReturn;
begin

    // set the root key
    RegDBSetDefaultRoot(HKEY_LOCAL_MACHINE);

    // check for the existence of the subkey
    nReturn = RegDBKeyExist("Software\\InstallShield");

    if (nReturn < 0) then
        MessageBox("Your system does not meet the system requirements. " +
            "Setup will now exit.", SEVERE);
        abort;
    endif;

end;

```

Code in this event handler is always executed, even during a maintenance setup or uninstallation, unless you place it inside the following if-then structure:

```

if (!MAINTENANCE) then
    // non-maintenance code
endif;

```

The setup sets the system variable [MAINTENANCE](#) equal to FALSE the first time the setup is run, and TRUE every time the setup is run thereafter.

## OnCCPSearch



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The OnCCPSearch event handler responds to the Upgrade Compliance event. Code this handler in setups that must search for an application whose presence qualifies the end user to install your application. Your code might, for example, call FindFile to locate a specific file. Note that this event handler is not executed during a maintenance setup or uninstallation.

## OnFirstUIBefore



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

*If the InstallScript installation is included in an Advanced UI or Suite/Advanced UI project as an InstallScript package, the Advanced UI or Suite/Advanced UI installation does not call this event handler. For more information, see [Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project](#).*

The OnFirstUIBefore event handler responds to the First UI Before event. It performs tasks that must take place before the installation of features for a first-time installation of an application.

Typically, this handler calls InstallScript functions for the following tasks:

- To set up the screen.
- To display dialogs that welcome the end user, display the software license, and show other information about the software to be installed.
- To get information from the end user—including user registration, the destination path (TARGETDIR in InstallScript projects and INSTALLDIR in InstallScript MSI projects), and the setup type.

## OnIISInitialize



**Project** • This information applies to InstallScript projects.

The OnIISInitialize is called before OnMoving. You can override this for similar purposes as OnIISVRootUninstalling, as well as add code that checks the version of IIS or makes custom IIS modifications before installing the web sites and virtual roots.

## OnMaintUIBefore



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

If the InstallScript installation is included in an Advanced UI or Suite/Advanced UI project as an InstallScript package, the Advanced UI or Suite/Advanced UI installation does not call this event handler. For more information, see [Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project](#).

The OnMaintUIBefore event handler responds to the Maintenance UI Before event. It performs tasks that must take place before the reinstallation of features for a maintenance installation of an application.

## OnSQLLogin



**Project** • This information applies to InstallScript MSI projects.

The OnSQLLogin event responds to the First UI Before event. This event handler function creates a dialog which is used by the script to specify SQL login credentials. These credentials include the login ID and password.



**Note** • If you want to call a SQL built-in function before the OnSQLLogin event gets called, you need to first call the [SQLRTInitialize2](#) function. This is applicable to all SQL-related functions. To learn more, see [Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects](#).

## OnSQLServerInitialize



**Project** • This information applies to InstallScript projects.

The OnSQLServerInitialize event is called by OnFirstUIBefore to establish any connections necessary for SQL Server support. This function will initialize the SQL Server runtime and attempt to make any necessary SQL Server connections, displaying a login dialog for each one. Parameter nBtn indicates whether NEXT or BACK was the result of the previously displayed dialog. It is for information purposes only.

If you are working on a script that had overridden OnFirstUIBefore when you upgraded to a newer version of InstallShield and your script is not calling OnSQLServerInitialize, then you should add the OnFirstUIBefore code to your script file.



---

**Note** • If you want to call a SQL built-in function before the `OnSQLServerInitialize` event gets called, you need to first call the `SQLRTInitialize2` function. This is applicable to all SQL-related functions. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

## OnSQLServerInitializeMaint



---

**Project** • This information applies to InstallScript projects.

The `OnSQLServerInitializeMaint` event is called by `OnMaintUIBefore` to establish any connections necessary for SQL Server support. This function will initialize the SQL Server runtime and attempt to make any necessary SQL Server connections using login credentials stored in the log file.



---

**Note** • This event is not called automatically in a `program...endprogram` style installation.

## OnSuiteInstallBefore



---

**Project** • This information applies to InstallScript projects.

The `OnSuiteShowUI` event calls the `OnSuiteInstallBefore` event. By default, `OnSuiteInstallBefore` is called to initialize information, such as the selection of features, that is necessary to perform file transfer.



---

**Note** • This event is not called automatically in a `program...endprogram` style installation.

## OnSuiteMaintBefore



---

**Project** • This information applies to InstallScript projects.

The `OnSuiteShowUI` event calls the `OnSuiteMaintBefore` event. By default, `OnSuiteMaintBefore` is called to initialize information, such as the selection of features, that is necessary to perform file transfer.



---

**Note** • This event is not called automatically in a `program...endprogram` style installation.

## OnSuiteUpdateBefore



---

**Project** • This information applies to InstallScript projects.

The OnSuiteShowUI event calls the OnSuiteUpdateBefore event. By default, OnSuiteUpdateBefore is called to initialize information, such as the selection of features, that is necessary to perform file transfer.




---

**Note** • This event is not called automatically in a *program...endprogram* style installation.

## OnUpdateUIBefore




---

**Project** • This information applies to InstallScript projects.

If the InstallScript installation is included in an Advanced UI or Suite/Advanced UI project as an InstallScript package, the Advanced UI or Suite/Advanced UI installation does not call this event handler. For more information, see *Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project*.

The OnUpdateUIBefore event handler function is called by the OnShowUI event handler to display the pre-file-transfer user interface for an update setup.

This event handler is not called in a setup that uses a procedural script (a script with a *program...endprogram* block).

### Syntax

```
OnUpdateUIBefore ( );
```

### Parameters

None.

### Return Values

None.

## OnXMLInitialize




---

**Project** • This information applies to InstallScript projects.

The OnXMLInitialize event is called by OnFirstUIBefore to initialize the XML run time.




---

**Note** • This event is not called automatically in a *program...endprogram* style installation.

## Move Data Handlers

The following event handlers are triggered immediately before, during, or immediately after the installation or uninstallation of all features on the target computer:

**Table 5 • Move Data Handlers**

Event Handler	Project Type	Description
<b>OnMoveData</b>	InstallScript	Called by OnShowUI to transfer files. The event handler's default code calls FeatureTransferData to transfer the files.
<b>OnCustomizeUninstInfo</b>	InstallScript	Called by OnMoveData to customize the uninstall information after calling MaintenanceStart.
<b>OnMoving</b>	InstallScript InstallScript MSI	In InstallScript projects: Called as a result of the installation calling FeatureTransferData or Feature MoveData. The event is called before any file transfer operations occur.  In InstallScript MSI projects: Called just before the action GenerateMSIScript is executed.
<b>OnMoved</b>	InstallScript InstallScript MSI	In InstallScript projects: Called directly by the framework after FeatureTransferData is called in the script and after data transfer.  In InstallScript MSI projects: Responds to the Moved event that is generated just after all features are installed or uninstalled on the target computer.
<b>OnInstallingFile</b>	InstallScript	Called when a file is about to be installed as a result of FeatureTransferData or FeatureMoveData.
<b>OnUninstallingFile</b>	InstallScript	Called when a file is about to be uninstalled as a result of FeatureTransferData or FeatureMoveData.
<b>OnUninstallingFontFile</b>	InstallScript	Called when a font file logged by RegisterFontResource is uninstalled.
<b>OnUninstallingDIFxDriverFile</b>	InstallScript	Called when a driver installed or preinstalled by the DIFxDriverPackageInstall or DIFxDriverPackagePreinstall functions is uninstalled with uninstall logging enabled.
<b>OnInstalledFile</b>	InstallScript	Called after a file is installed as a result of FeatureTransferData or FeatureMoveData.
<b>OnInstalledFontFile</b>	InstallScript	Called after a file is installed that is listed in the media as a font file.
<b>OnUninstalledFile</b>	InstallScript	Called after a file is uninstalled as a result of FeatureTransferData or FeatureMoveData.

**Table 5 • Move Data Handlers (cont.)**

Event Handler	Project Type	Description
<b>OnSQLComponentInstalled</b>	InstallScript	Called after each component is installed so any SQL scripts attached to that component can be run. SQLComponentInstalled is called after each component is installed so any SQL scripts attached to that component can be run.
<b>OnSQLComponentUninstalled</b>	InstallScript	Called after each component is uninstalled so any SQL scripts attached to that component can be run. The SQLComponentUninstalled event is called after each component is uninstalled so any SQL scripts attached to that component can be run.
<b>OnSQLBatchScripts</b>	InstallScript	Called automatically by the framework after file transfer.
<b>OnIISComponentInstalled</b>	InstallScript	IISComponentInstalled is called after each component is installed so that any IIS information attached to that component can be installed.
<b>OnIISVRootUninstalling</b>	InstallScript	Called before each IISVRoot is removed.
<b>OnXMLComponentInstalled</b>	InstallScript	ISXMLComponentInstall is called after each component is installed so that any XML information attached to that component can be installed.
<b>OnXMLComponentUninstalling</b>	InstallScript	XMLComponentUninstalling is called before each .xml file is removed.
<b>OnNetApiCreateUserAccount</b>	InstallScript	Called before OnMoving event. This event handler function does nothing unless <b>NetApiRT.obl</b> is added before <b>Isrt.obl</b> in the list of linked libraries.
<b>OnGeneratedMSIScript</b>	InstallScript MSI	Responds to the MSI standard action LaunchConditions after it is executed.
<b>OnGeneratingMSIScript</b>	InstallScript MSI	Responds to the MSI standard action LaunchConditions before it is executed.
<b>OnInstallFilesActionBefore</b>	InstallScript MSI	Responds to the InstallFiles event handler function before it is called.
<b>OnInstallFilesActionAfter</b>	InstallScript MSI	Responds to the InstallFiles event handler function after it is called.

## OnCustomizeUninstInfo



---

**Project** • This information applies to InstallScript projects.

The OnCustomizeUninstInfo event handler function is called by OnMoveData to customize the uninstall information after calling MaintenanceStart.

### Syntax

```
OnCustomizeUninstInfo ( );
```

### Parameters

None.

### Return Values

None.

## OnGeneratedMSIScript



---

**Project** • This information applies to InstallScript MSI projects.

The OnGeneratedMSIScript event handler is called after the MSI standard action LaunchConditions is executed. This event handler does not contain any code by default; it is included for legacy installations.

## OnGeneratingMSIScript



---

**Project** • This information applies to InstallScript MSI projects.

The OnGeneratingMSIScript event handler responds to the MSI standard action LaunchConditions before it is executed. This event handler does not contain any code by default; it is included for legacy installations.

## OnIISComponentInstalled

The OnIISComponentInstalled event handler function is called after each component is installed so any IIS information attached to that component can be installed.

### Syntax

```
OnIISComponentInstalled ( szComponent );
```



## Parameters

**Table 6** • OnIISComponentInstalled Parameters

Parameter	Description
<b>szComponent</b>	This parameter will have the name of the component that has been installed.

## Return Values

This event handler function should return ISERR\_SUCCESS currently in all cases.

## OnIISVRootUninstalling

The OnIISVRootUninstalling event is called before each Virtual Directory is removed. You can check conditions related to IIS in your script, and then call abort() to stop the setup.

## OnInstalledFile



**Project** • This information applies to InstallScript projects.

The OnInstalledFile event handler function is called after a file is installed as a result of FeatureTransferData or FeatureMoveData.

## Syntax

```
OnInstalledFile ( szFilename );
```

## Parameters

**Table 7** • OnInstalledFile Parameters

Parameter	Description
<b>szFilename</b>	Specifies the fully qualified file name of the file that was transferred.

## Return Values

None.

## OnInstalledFontFile



**Project** • This information applies to InstallScript projects.

The OnInstalledFontFile event handler function is called after a file is installed that is listed in the media as a font file.

## Syntax

```
OnInstalledFontFile ( pFontFileInfo );
```

## Parameters

**Table 8 • OnInstalledFontFile Parameters**

Parameter	Description
<b>pFontFileInfo</b>	Pointer to a <a href="#">_FONTFILEINFO structure</a> that gives information about the font file that is being installed.

## Return Values

None.

## OnInstallFilesActionAfter

The OnInstallFilesActionAfter event handler is called just after the standard Windows Installer action InstallFiles is performed.

## OnInstallFilesActionBefore

The OnInstallFilesActionBefore event handler is called just before the standard Windows Installer action InstallFiles is performed.

## OnInstallingFile



**Project •** This information applies to InstallScript projects.

The OnInstallingFile event handler function is called when a file is about to be installed as a result of FeatureTransferData or FeatureMoveData.

Code in this event handler is always executed, even during a maintenance setup, unless you place it inside the following if-then structure:

```
if !MAINTENANCE then
    \\ non-maintenance code
endif;
```

## Syntax

```
OnInstallingFile ( szFilename );
```

## Parameters

**Table 9 • OnInstallingFile Parameters**

Parameter	Description
<b>szFilename</b>	Specifies the fully qualified file name of the file that is about to be transferred.

## Return Values

None.

## OnMoved

In InstallScript projects, OnMoved is called as a result of the installation calling FeatureTransferData or FeatureMoveData. The event is called when all file transfer operations are completed except for batch self-registration. In InstallScript MSI projects, OnMoved is called just before the action GenerateMSIScript is executed.

Code in this event handler is always executed, even during a maintenance setup or uninstallation, unless you place it inside the following if-then structure:

```
if (!MAINTENANCE) then
    // non-maintenance code
endif;
```

## OnMoveData



**Project •** This information applies to InstallScript projects.

The OnMoveData event handler function is called by the OnShowUI event handler to handle the file transfer. By default, OnMoveData calls FeatureTransferData to transfer the files.

This event handler is not called in a setup that uses a procedural script (a script with a program...endprogram block).

## Syntax

```
OnMoveData ( );
```

## Parameters

None.

## Return Values

None.

## OnMoving

In InstallScript projects, OnMoving is called as a result of the installation calling FeatureTransferData or FeatureMoveData. The event is called before any file transfer operations occur. In InstallScript MSI projects, OnMoving is called just before the action GenerateMSIScript is executed.

Code in this event handler is always executed, even during a maintenance setup, unless you place it inside the following if-then structure:

```
if !MAINTENANCE then
    \\ non-maintenance code
endif;
```

## OnNetApiCreateUserAccount

The OnNetApiCreateUserAccount event handler function is called before the OnMoving event. This event handler function does nothing unless you add **NetApiRT.obl** before **Isrt.obl** in the list of linked libraries.

### Syntax

OnNetApiCreateUserAccount()

### Parameters

None

### Return Values

This event handler function should always return ISERR\_SUCCESS.

### Additional Information



#### Task

#### To add NetApiRT.obl to the list of linked libraries:

1. On the **Build** menu, click **Settings**. The **Settings** dialog box opens.
2. In the **Libraries (.obl)** box, enter **NetApiRT.obl**. Note that it must be listed before **Isrt.obl**.

The path to **NetApiRT.obl** is as follows:

```
<ISProductFolder>\Script\ISRT\Lib\NetApiRT.obl
```

## OnSQLBatchScripts

The OnSQLBatchScripts event is called automatically by the framework after file transfer.

## OnSQLComponentInstalled



**Project** • This information applies to InstallScript projects.

The OnSQLComponentInstalled event is called after each component is installed so any SQL scripts attached to that component can be run. SQLComponentInstalled is called after each component is installed so any SQL scripts attached to that component can be run.

## OnSQLComponentUninstalled



**Project** • This information applies to InstallScript projects.

The SQLComponentUninstalled event is called after each component is uninstalled so any SQL scripts attached to that component can be run. The SQLComponentUninstalled event is called after each component is uninstalled so any SQL scripts attached to that component can be run. szComponent will have the name of the component that has been installed.

## OnUninstalledFile



**Project** • This information applies to InstallScript projects.

The OnUninstalledFile event handler function is called after a file is uninstalled as a result of FeatureTransferData or FeatureMoveData.

### Syntax

```
OnUninstalledFile ( szFilename );
```

### Parameters

**Table 10** • OnUninstalledFile Parameters

Parameter	Description
<b>szFilename</b>	Specifies the fully qualified file name of the file that was uninstalled.

### Return Values

None.

## OnUninstallingFile



**Project** • This information applies to InstallScript projects.

The OnUninstallingFile event handler function is called when a file is about to be uninstalled as a result of FeatureTransferData or FeatureMoveData.

Code in this event handler is always executed, even during a maintenance setup, unless you place it inside the following if-then structure:

```
if !MAINTENANCE then
    \\ non-maintenance code
endif;
```

## Syntax

```
OnUninstallingFile ( szFilename );
```

## Parameters

**Table 11** • OnUninstallingFile

Parameter	Description
<b>szFilename</b>	Specifies the fully qualified file name of the file that is about to be uninstalled.

## Return Values

None.

## OnUninstallingDIFxDriverFile



**Project** • This information applies to InstallScript projects.

The OnUninstallingDIFxDriverFile event handler function is called when a driver installed or preinstalled by the DIFxDriverPackageInstall or DIFxDriverPackagePreinstall functions is uninstalled with uninstall logging enabled.

By default, the event uninstalls the driver using the DIFxDriverPackageUninstall function.

## Syntax

```
OnUninstallingDIFxDriverFile ( byval string szDriver );
```

## Parameters

**Table 12** • OnUninstallingDIFxDriverFile

Parameter	Description
<b>szDriver</b>	The complete path and file of the installed driver file.

## Return Values

None.

## OnUninstallingFontFile



**Project** • This information applies to InstallScript projects.

The OnUninstallingFontFile event handler function is called when a font file logged by RegisterFontResource is uninstalled.

## Syntax

```
OnUninstallingFontFile ( pFontFileInfo );
```

## Parameters

**Table 13** • OnUninstallingFontFile

Parameter	Description
<b>pFontFileInfo</b>	Pointer to a <a href="#">_FONTFILEINFO</a> structure that gives information about the font file that is being uninstalled.

## Return Values

None.

## OnXMLComponentInstalled

OnXMLComponentInstalled is the event handler function associated with the ISXMLComponentInstall event. The ISXMLComponentInstall event is called after each component is installed so that any XML information attached to that component can be installed.

## Syntax

```
OnXMLComponentInstalled ( szComponent );
```

## Parameters

**Table 14** • OnXMLComponentInstalled

Parameter	Description
<b>szComponent</b>	This parameter will have the name of the component that has been installed.

## Return Values

This event handler function should return ISERR\_SUCCESS currently in all cases.

## OnXMLComponentUninstalling

OnXMLComponentUninstalling is the event handler function associated with the XMLRTComponentUninstall event. The XMLRTComponentUninstall event is called before each .xml file is removed.

## Syntax

```
OnXMLComponentUninstalling ( szXmlComponent )
```

## Parameters

**Table 15 • OnXMLComponentUninstalling**

Parameter	Description
<b>szXmlComponent</b>	This parameter will have the name of the component that has been installed.



## Return Values

This event handler function should return ISERR\_SUCCESS currently in all cases.

## After Data Move Handlers





The following event handlers are triggered after files and other data are transferred to the target computer:

**Table 16 • After Data Move Handlers**


Event Handler	Project Type	Description
OnFirstUIAfter	InstallScript	In InstallScript projects: The OnFirstUIAfter event called by OnShowUI after the file transfer of the setup when the setup is running in first install mode. By default this event displays UI that informs the end user that the setup has been completed successfully.
	InstallScript MSI	
		
<b>Note</b> • <i>This event is not called automatically in a program...endprogram style installation.</i>		
		In InstallScript MSI projects: Responds to the First UI After event.
OnMaintUIAfter	InstallScript	In InstallScript projects: The OnMaintUIAfter event called by OnShowUI after the file transfer of the setup when the setup is running in maintenance mode. By default this event displays UI that informs the end user that the maintenance setup has been completed successfully.
	InstallScript MSI	
		
<b>Note</b> • <i>This event is not called automatically in a program...endprogram style installation.</i>		
		In InstallScript MSI projects: Responds to the Maintenance UI After event.



**Table 16 •** After Data Move Handlers (cont.)

Event Handler	Project Type	Description
<b>OnUpdateUIAfter</b>	InstallScript	The OnUpdateUIAfter event called by OnShowUI after the file transfer of the setup when the setup is running in update mode. By default this event displays UI that informs the end user that the maintenance setup has been completed successfully. This event is not automatically called in a program.
<b>OnEnd</b>	InstallScript InstallScript MSI	In InstallScript projects: The OnEnd event is called at the end of the setup. This event is not called if the setup is aborted.  In InstallScript MSI projects: Responds to the End event, the last redefinable event in a setup.
<b>OnSuiteInstallAfter</b>	InstallScript	The OnSuiteShowUI event calls the OnSuiteInstallAfter event after file transfer has completed during an installation.   <b>Note •</b> This event is not called automatically in a program...endprogram style installation.
<b>OnSuiteMaintAfter</b>	InstallScript	The OnSuiteShowUI event calls the OnSuiteMaintAfter event after file transfer has completed during maintenance.   <b>Note •</b> This event is not called automatically in a program...endprogram style installation.
<b>OnSuiteUpdateAfter</b>	InstallScript	The OnSuiteShowUI event calls the OnSuiteUpdateAfter event after file transfer has completed during an upgrade.   <b>Note •</b> This event is not called automatically in a program...endprogram style installation.
<b>OnIISUninitialize</b>	InstallScript	The OnIISUninitialize event is called by OnMoveDataAfter to initialize the IIS runtime.   <b>Note •</b> This event is not called automatically in a program...endprogram style installation.

**Table 16 •** After Data Move Handlers (cont.)

Event Handler	Project Type	Description
<b>OnXMLUninitialize</b>	InstallScript	The OnXMLUninitialize event is called by OnMoveDataAfter to initialize the XML runtime.
		 <p><b>Note •</b> This event is not called automatically in a program...endprogram style installation.</p>

## OnEnd

The OnEnd event handler responds to the End event, the last redefinable event in a setup. You can place any required cleanup code in the OnEnd function.

OnEnd is prototyped for you when you include ifx.h or iswi.h in your script. Define OnEnd in your script as in the following example:

```
#include "iswi.h"

function OnEnd( )
    // local variables
begin
    // cleanup code
end;
```

Code in this event handler is always executed, even during a maintenance setup, unless you place it inside the following if-then structure:

```
if !MAINTENANCE then
    \\ non-maintenance code
endif;
```

## OnFirstUIAfter



**Project •** If the InstallScript installation is included in an Advanced UI or Suite/Advanced UI project as an InstallScript package, the Advanced UI or Suite/Advanced UI installation does not call this event handler. For more information, see *Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project*.

The OnFirstUIAfter event handler responds to the First UI After event. It performs tasks that must take place after the installation of features for a first installation of an application.

## OnIISUninitialize

The OnIISUninitialize event is called by OnMoveDataAfter to initialize the IIS runtime. This event will not be called automatically in an endprogram style installation.

## OnMaintUIAfter




---

**Project** • If the InstallScript installation is included in an Advanced UI or Suite/Advanced UI project as an InstallScript package, the Advanced UI or Suite/Advanced UI installation does not call this event handler. For more information, see *Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project*.

The OnMaintUIAfter event handler responds to the Maintenance UI After event. It performs tasks that must take place after the reinstallation of features for a maintenance installation of an application.

## OnSuiteInstallAfter




---

**Project** • This information applies to InstallScript projects.

The OnSuiteShowUI event calls the OnSuiteInstallAfter event after file transfer has completed during an installation.




---

**Note** • This event is not called automatically in a program...endprogram style installation.

## OnSuiteMaintAfter




---

**Project** • This information applies to InstallScript projects.

The OnSuiteShowUI event calls the OnSuiteMaintAfter event after file transfer has completed during maintenance.




---

**Note** • This event is not called automatically in a program...endprogram style installation.

## OnSuiteUpdateAfter




---

**Project** • This information applies to InstallScript projects.

The OnSuiteShowUI event calls the OnSuiteUpdateAfter event after file transfer has completed during an upgrade.




---

**Note** • This event is not called automatically in a program...endprogram style installation.

## OnUpdateUIAfter



**Project** • If the InstallScript installation is included in an Advanced UI or Suite/Advanced UI project as an InstallScript package, the Advanced UI or Suite/Advanced UI installation does not call this event handler. For more information, see *Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project*.

The OnUpdateUIAfter event handler function is called by the OnShowUI event handler to display the post-file-transfer user interface for an update setup.

This event handler is not called in a setup that uses a procedural script (a script with a program...endprogram block).

### Syntax

```
OnUpdateUIAfter ( );
```

### Parameters

None.

### Return Values

None.

## OnXMLUninitialize

The OnXMLUninitialize event is called by OnMoveDataAfter to initialize the XML runtime. This event will not be called automatically in an endprogram style installation.

## Feature Event Handlers

Feature event handlers carry out processes required just before and just after the installation or uninstallation of a single feature. The number of feature events and handlers depends on the number of features in your project.

To create an event handler function for a feature, select the feature name from the left event-category list, and select the event you want from the right event list. InstallShield creates a second InstallScript file, called **FeatureEvents.rul**, in the InstallScript view.

Note that if you change the default feature event handler code in **FeatureEvents.rul**, you must put the following statement in **Setup.rul** to include your changes in the installation:

```
#include "FeatureEvents.rul"
```

Following is a list of the feature event handlers.

**Table 17 • Feature Event Handlers**

Event Handler	Project Type	Description
<b>OnInstalled</b>	InstallScript InstallScript MSI	In InstallScript projects: Runs in response to the Installed event that is generated just after the applicable feature is installed on the target system.  In InstallScript MSI projects: Runs just after Windows Installer transfers the files to the target system.
<b>OnInstalling</b>	InstallScript InstallScript MSI	In InstallScript projects: Runs in response to the Installing event that is generated just before the feature is installed on the target system.  In InstallScript MSI projects: Runs just before Windows Installer transfers the files to the target system.
<b>OnUnInstalled</b>	InstallScript InstallScript MSI	In InstallScript projects: Runs in response to the UnInstalled event that is generated just after the feature is removed from the target system.  In InstallScript MSI projects: Runs just after Windows Installer removes the files from the target system.
<b>OnUnInstalling</b>	InstallScript InstallScript MSI	In InstallScript projects: Runs in response to the UnInstalling event that is generated just before the feature is removed from the target system.  In InstallScript MSI projects: Runs just before Windows Installer removes the files from the target system.



**Project •** Because Windows Installer controls the installation of features in InstallScript MSI installations, the order in which feature event-handler functions are called cannot be specified. In addition, feature events are not launched until all features are copied to the target system.

Also note that feature-uninstallation event handlers (OnUnInstalling and OnUnInstalled) are not called during rollback.

## OnInstalled



**Project •** This information applies to the following project types:

- InstallScript
- InstallScript MSI

In InstallScript projects, the OnInstalled feature event handler runs in response to the Installed event that is generated just after the applicable feature is installed on the target system.

In InstallScript MSI projects, the OnInstalled feature event handler runs just after Windows Installer transfers the files to the target system.

The OnInstalled feature event handler is listed in the InstallScript code in the following format:

```
MyFeatureName_Installed()
```




---

**Project** • Because Windows Installer controls the installation of features in InstallScript MSI installations, the order in which feature event-handler functions are called cannot be specified. In addition, feature events are not launched until all features are copied to the target system.

Also note that feature-uninstallation event handlers (OnUnInstalling and OnUnInstalled) are not called during rollback.

## OnInstalling




---

**Project** • This information applies to the following project types:

- InstallScript
- InstallScript MSI

In InstallScript projects, the OnInstalling feature event handler runs in response to the Installing event that is generated just before the feature is installed on the target system.

In InstallScript MSI projects, the OnInstalling feature event handler runs just before Windows Installer transfers the files to the target system.

The OnInstalling feature event handler is listed in the InstallScript code in the following format:

```
MyFeatureName_Installing()
```




---

**Project** • Because Windows Installer controls the installation of features in InstallScript MSI installations, the order in which feature event-handler functions are called cannot be specified. In addition, feature events are not launched until all features are copied to the target system.

Also note that feature-uninstallation event handlers (OnUnInstalling and OnUnInstalled) are not called during rollback.

## OnUnInstalled




---

**Project** • This information applies to the following project types:

- InstallScript
- InstallScript MSI

In InstallScript projects, the OnUnInstalled feature event handler runs in response to the UnInstalling event that is generated just before the feature is removed from the target system.

In InstallScript MSI projects, the OnUnInstalled feature event handler runs just before Windows Installer removes the files from the target system.

The OnUnInstalled feature event handler is listed in the InstallScript code in the following format:

```
MyFeatureName_UnInstalled()
```




---

**Project** • Because Windows Installer controls the installation of features in InstallScript MSI installations, the order in which feature event-handler functions are called cannot be specified. In addition, feature events are not launched until all features are copied to the target system.

Also note that feature-uninstallation event handlers (OnUnInstalling and OnUnInstalled) are not called during rollback.

## OnUnInstalling




---

**Project** • This information applies to the following project types:

- InstallScript
- InstallScript MSI

In InstallScript projects, the OnUnInstalling feature event handler runs in response to the UnInstalling event that is generated just before the feature is removed from the target system.

In InstallScript MSI projects, the OnUnInstalling feature event handler runs just before Windows Installer removes the files from the target system.

The OnUnInstalling feature event handler is listed in the InstallScript code in the following format:

```
MyFeatureName_UnInstalling()
```




---

**Project** • Because Windows Installer controls the installation of features in InstallScript MSI installations, the order in which feature event-handler functions are called cannot be specified. In addition, feature events are not launched until all features are copied to the target system.

Also note that feature-uninstallation event handlers (OnUnInstalling and OnUnInstalled) are not called during rollback.

# Miscellaneous Event Handlers

Miscellaneous event handlers are triggered by unscheduled events during an installation, such as the end user exiting the installation.

**Table 18 • Miscellaneous Event Handlers**


Event Handler	Project Type	Description
<b>OnAbort</b>	InstallScript, InstallScript MSI	In InstallScript projects: The OnAbort event handler is called when a setup is aborted via the abort keyword.  In InstallScript MSI projects: Responds to the Abort event generated by InstallScript's abort command.
<b>OnAdminInstallUIAfter</b>	InstallScript MSI	Responds to the Admin Install UI After event.
<b>OnAdminInstallUIBefore</b>	InstallScript MSI	Responds to the Admin Install UI Before event by displaying dialogs that gather information from the end user for an administrative installation of an application.
<b>OnAdminPatchUIAfter</b>	InstallScript MSI	Called after the file transfer of an administrative patch.
<b>OnAdminPatchUIBefore</b>	InstallScript MSI	Called before the file transfer of an administrative patch.
<b>OnAdvertisementAfter</b>	InstallScript MSI	Responds to the Advertisement After event.
<b>OnAdvertisementBefore</b>	InstallScript MSI	Responds to the Advertisement After Before event.
<b>OnCanceling</b>	InstallScript, InstallScript MSI	InstallScript projects: The OnCanceling event is sent when the installation is cancelled, usually as result of the end user clicking the Cancel button of a dialog or pressing the Esc key. Calling Do(EXIT) will also trigger this event.  In InstallScript MSI projects: Responds to the Cancel event generated when the end user clicks the Cancel button in a built-in dialog.
<b>OnComponentError</b>	InstallScript, InstallScript MSI	In InstallScript projects: The OnComponentError event is called by the framework when the call to FeatureTransferData or FeatureMoveData returns an error.  In InstallScript MSI projects: Responds to general file-transfer errors.
<b>OnDIFxLogCallback</b>	InstallScript	Called when a DIFx-related event occurs that is logged by the built-in DIFx callback functionality.
<b>OnError</b>	InstallScript MSI	This event is called when MSI sends an INSTALLMESSAGE_ERROR message.
<b>OnException</b>	InstallScript MSI	Responds to exceptions generated by a procedural script.



**Table 18 •** Miscellaneous Event Handlers (cont.)

Event Handler	Project Type	Description
<b>OnFileError</b>	InstallScript	The OnFileError event is sent when an unknown error occurs during the installation or uninstallation of a file.
<b>OnFileLocked</b>	InstallScript	The OnFileLocked event is called when a file that is in use by another application needs to be installed or uninstalled unless the files are in a file group which is potentially marked as locked or shared. In that case, the file will be installed or uninstalled after reboot.
<b>OnFileReadOnly</b>	InstallScript	The OnFileReadOnly event is called when a read-only file needs to be installed or uninstalled.
<b>OnFilesInUse</b>	InstallScript MSI	<p>The OnFilesInUse event handler is called in when the Windows Installer sends an INSTALLMESSAGE_FILESINUSE message to the installation.</p> <p>By default, the OnFilesInUse event handler displays the <b>SdFilesInUse</b> dialog.</p>
<b>OnHelp</b>	InstallScript, InstallScript MSI	<p>In InstallScript projects: The OnHelp event is called when the end user presses F1 or Do(HELP) is called.</p> <p>In InstallScript MSI projects: Responds to the Help event generated when the end user presses the F1 key.</p>
<b>OnInternetError</b>	InstallScript	The OnInternetError is called when an error occurs while installing a file from the Internet.
<b>OnLaunchAppAndWaitCallback</b>	InstallScript, InstallScript MSI	The OnLaunchAppAndWaitCallback is called while the installation is waiting for an application to launch if LAAW_OPTION_USE_CALLBACK was specified when calling the LaunchAppAndWait function.
<b>OnLogonUserSetMsiProperties</b>	InstallScript MSI	The OnLogonUserSetMsiProperties event handler sets the Windows Installer properties for the logon user support that is described for InstallScript MSI projects in Adding the Ability to Create or Set an Existing User Account.
<b>OnMD5Error</b>	InstallScript	The OnMD5Error event is called when the MD5 signature of a file being installed does not match the MD5 value stored in the InstallShield CAB file. (The MD5 is calculated when the media is built.)
<b>OnMsiSilentInstall</b>	InstallScript MSI	Responds to the MSI Silent Install event generated when a user runs an InstallScript MSI project's .msi package in silent mode.

**Table 18 •** Miscellaneous Event Handlers (cont.)

Event Handler	Project Type	Description
<b>OnNextDisk</b>	InstallScript	The OnNextDisk event is called during file transfer when the setup cannot find a data file that is needed. An example of this occurrence is during a multiple floppy or CD install when the next disk is needed.
<b>OnOutOfDiskSpace</b>	InstallScript MSI	Responds to the target system running out of free disk space.
<b>OnPatchUIAfter</b>	InstallScript MSI	Responds to the Patch UI After event.
<b>OnPatchUIBefore</b>	InstallScript MSI	Responds to the Patch UI Before event by displaying dialogs for a patch installation.
<b>OnRebooted</b>	InstallScript, InstallScript MSI (if the InstallScript user interface (UI) style is the traditional style, which uses the InstallScript engine as an external UI handler)	<p>In InstallScript projects: The OnRebooted event is called when the setup is run automatically after rebooting the system. This is the only event that will be called in this case.</p> <p>In InstallScript MSI projects: Responds to the Rebooted event that the installation generates when it restarts after the target system is rebooted.</p>  <p><b>Important •</b> This event is not called in an InstallScript MSI installation in which the InstallScript UI style is the new style (which uses the InstallScript engine as an embedded UI handler). To learn more, see <i>Using the InstallScript Engine as an External vs. Embedded UI Handler for InstallScript MSI Installations</i>.</p>
<b>OnRemovingSharedFile</b>	InstallScript	The OnRemovingSharedFile event is called during file transfer when a shared file is being uninstalled and the reference count for the file has reached zero.
<b>OnResumeUIAfter</b>	InstallScript MSI	Responds to the Resume UI After event.
<b>OnResumeUIBefore</b>	InstallScript MSI	Responds to the Resume UI Before event.
<b>OnRMFilesInUse</b>	InstallScript MSI	<p>The OnRMFilesInUse event handler is called when the Restart Manager is enabled and Windows Installer 4.0 sends an INSTALLMESSAGE_RMFILESINUSE message to the installation.</p> <p>By default, the OnRMFilesInUse event handler displays the SdRMFilesInUse dialog.</p>

**Table 18 •** Miscellaneous Event Handlers (cont.)

Event Handler	Project Type	Description
<b>OnSelfRegistrationError</b>	InstallScript	The OnSelfRegistrationError event handler is called directly by the framework when a call to Do(SELFREGISTRATIONPROCESS) fails to register files successfully.
<b>OnWarning</b>	InstallScript MSI	Responds to the Warning event generated when the Windows Installer service sends an INSTALLMESSAGE_WARNING message to the installation's user interface.

## OnAbort



**Project •** This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The OnAbort event handler responds to the Abort event that is generated by the InstallScript abort statement, which uninstalls any changes made to the target system and then exits. In the event handler, you can place any additional cleanup code that you require, such as deleting temporary files created by your installation.

## OnAdminInstallUIAfter



**Project •** This information applies to *InstallScript MSI* projects.

The OnAdminInstallUIAfter event handler responds to the Admin Install UI After event. It performs tasks that must take place after data transfer for an administrative installation of an application. To perform an administrative installation, the user launches **Setup.exe** with the /a switch.

Typically, this handler calls [SdFinishEx](#) to inform the user that the administrative installation is complete.

## OnAdminInstallUIBefore



**Project •** This information applies to *InstallScript MSI* projects.

The OnAdminInstallUIBefore event handler responds to the Admin Install UI Before event. It performs tasks that must take place before data transfer for an administrative installation of an application. To perform an administrative installation, the user launches **Setup.exe** with the /a switch.

Typically, this handler calls the [SdWelcome](#) and [AdminAskPath](#) functions to welcome the administrative user and prompt for a destination directory.

## OnAdminPatchUIAfter



**Project** • This information applies to InstallScript MSI projects.

The AdminPatchUIAfter event is called after the file transfer of an admin patch setup. By default this event displays UI that informs the end user that the installation has been completed successfully.

## OnAdminPatchUIBefore



**Project** • This information applies to InstallScript MSI projects.

The OnAdminInstallUIBefore event is called before the file transfer in an admin patch setup. By default this event displays UI allowing the end user to specify installation parameters.

## OnAdvertisementAfter



**Project** • This information applies to InstallScript MSI projects.

The OnAdvertisementAfter event handler responds to the Advertisement After event. It performs tasks that must take place after an advertised installation, which occurs when the user runs **Setup.exe** with the /j argument.

## OnAdvertisementBefore



**Project** • This information applies to InstallScript MSI projects.

The OnAdvertisementBefore event handler responds to the Advertisement Before event. It performs tasks that must take place before an advertised installation, which occurs when the user runs **Setup.exe** with the /j argument.

## OnCanceling



**Project** • This information applies to the following project types:

- InstallScript
- InstallScript MSI

The OnCanceling event handler responds to the Cancel event that is generated when the end user clicks the Cancel button in one of the built-in InstallScript dialogs.

```
function OnCanceling( )
begin
```

```
if (YES = AskYesNo(  
    "Are you sure you want to cancel the setup?",  
    YES))  
    then abort;  
endif;  
  
end;
```

## OnComponentError



---

**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The OnComponentError event handler responds to the ComponentError event that is generated when the installation encounters a general file-transfer error.

The default OnComponentError implementation uses properties of an ErrorInfo object that is declared in OnComponentError and assigned a value by the following statement:

```
set ErrorInfo = ComponentErrorInfo( );
```

**Table 19 • OnComponentError Parameters**

Property	Description
<b>ErrorInfo.Feature</b>	An object whose properties provide information about the feature that was being transferred when the error occurred. If the error is not associated with a particular feature, or the feature cannot be identified, this property is not set; test for this case by checking the value of <code>IsObject(ErrorInfo.Feature)</code> .
<b>ErrorInfo.Feature.Description</b>	A string describing the file transfer error. This string may be null ("").
<b>ErrorInfo.Feature.DisplayName</b>	The display name of the feature that was being transferred when the error occurred. If you did not specify a display name for this feature, this string is null ("").
<b>ErrorInfo.Feature.Name</b>	The name of the feature that was being transferred when the error occurred.
<b>ErrorInfo.FileError.Description</b>	A string-formatted description of <code>ErrorInfo.LastError</code> .
<b>ErrorInfo.FileError.File</b>	For a file-related error: The path and name of the file that encountered an error while the installation attempted to transfer it.
<b>ErrorInfo.FileGroup</b>	The name of the component that is being transferred when the error occurs.
<b>ErrorInfo.LastError</b>	The numeric code for the file-transfer error.

## OnDIFxLogCallback



**Project** • This information applies to *InstallScript* projects.

The `OnDIFxLogCallback` event handler is called when a DIFx-related event occurs that is logged by the built-in DIFx callback functionality. See the Windows documentation on `DIFXAPISetLogCallback` for more information.



**Note** • This event is not supported for 64-bit drivers and is not called as a result of installing a 64-bit driver.

### Syntax

```
OnDIFxLogCallback ( byval number nEventType, byval number nError, byval string szDescription );
```

## Parameters

**Table 20 • OnDIFxLogCallback Parameters**

Parameter	Description
<b>nEventType</b>	The event type as documented in the DIFxAPI. The following values are available: <ul style="list-style-type: none"> <li>• <b>DIFXAPI_SUCCESS</b>—A success event that logs a message that indicates an operation succeeded.</li> <li>• <b>DIFXAPI_INFO</b>—An information event that logs a message that describes the context or progress of an operation.</li> <li>• <b>DIFXAPI_WARNING</b>—A warning event that logs a message about a possible problem that is not a fatal error.</li> <li>• <b>DIFXAPI_ERROR</b>—An error event that logs a message about a fatal error.</li> </ul>
<b>nError</b>	Specifies a Win32 error code that is associated with an event if one exists; otherwise, zero.
<b>szDescription</b>	A string that describes the event.

## Return Values

None.

## OnError



**Project •** This information applies to InstallScript MSI projects.

This event is called when Windows Installer sends an `INSTALLMESSAGE_ERROR` message.

## OnException



**Project •** This information applies to InstallScript MSI projects.

The `OnException` event handler responds to exceptions that are generated by a procedural script (one that uses an `explicit program...endprogram` block). The default implementation displays the error number, source, and description that are stored in the [Err Object](#).

Note that `OnException` is not called in an event-based script. For an event-based script, you must implement your own `try...catch...endcatch` blocks to catch exceptions.

## OnFileError



**Project** • This information applies to InstallScript projects.

The OnFileError event handler responds to the FileError event that is generated when the setup encounters a file error that does not generate any other file error event (for example, FileLocked or FileReadOnly). When creating an InstallShield object, note that this event is not triggered in an object.

This event handler is called (when appropriate) in any setup, including a setup that uses a procedural script (a script with a program...endprogram block).

### Syntax

```
OnFileError ( szFilename, nError );
```

### Parameters

**Table 21** • OnFileError Parameters

Parameter	Description
<b>szFilename</b>	Specifies the fully qualified file name of the file for which the error occurred.
<b>nError</b>	Specifies the value returned by the Windows API function GetLastError when the error occurred.

### Return Values

**Table 22** • OnFileError Return Values

Return Value	Description
<b>ERR_IGNORE</b>	Returned to the setup by the OnFileError event handler to request that the setup ignore the failure to install or uninstall the file specified in OnFileError's first argument and continue without performing the operation.
<b>ERR_RETRY</b>	Returned to the setup by the OnFileError event handler to request that the setup attempt again to install or uninstall the file specified in OnFileError's first argument.
<b>ERR_ABORT</b>	Returned to the setup by the OnFileError event handler to request that the setup be aborted.

## OnFileLocked



**Project** • This information applies to InstallScript projects.



The OnFileLocked event handler responds to the FileLocked event that is generated when the setup encounters a locked (in use) file that must be deleted or overwritten. When creating an InstallShield object, note that this event is not triggered in an object.

This event handler is not called for files that are in components whose Potentially Locked property is set to Yes. In this case the file operation is automatically performed after reboot.

This event handler is called (when appropriate) in any setup, including a setup that uses a procedural script (a script with a program...endprogram block).

## Syntax

```
OnFileLocked ( szFilename );
```

## Parameters

**Table 23 • OnFileLocked Parameters**

Parameter	Description
<b>szFilename</b>	Specifies the fully qualified file name of the locked file.

## Return Values

**Table 24 • OnFileLocked Return Values**

Return Value	Description
<b>ERR_IGNORE</b>	Returned to the setup by the OnFileLocked event handler to request that the setup ignore the failure to install or uninstall the file specified in OnFileLocked's argument and continue without performing the operation.
<b>ERR_RETRY</b>	Returned to the setup by the OnFileLocked event handler to request that the setup attempt again to install or uninstall the file specified in OnFileLocked's argument.
<b>ERR_ABORT</b>	Returned to the setup by the OnFileLocked event handler to request that the setup be aborted.
<b>ERR_PERFORM_AFTER_REBOOT</b>	Returned to the setup by the OnFileLocked event handler to request that the setup perform the operation on the file after the target system is rebooted.

## OnFileReadOnly



**Project •** This information applies to InstallScript projects.

The OnFileReadOnly event handler responds to the ReadOnly event that is generated when a file that must be deleted or overwritten is set to Read Only. When creating an InstallShield object, note that this event is not triggered in an object.

This event handler is called (when appropriate) in any setup, including a setup that uses a procedural script (a script with a program...endprogram block).

## Syntax

```
OnFileReadOnly ( szFilename );
```

## Parameters

**Table 25** • OnFileReadOnly Parameters

Parameter	Description
<b>szFilename</b>	Specifies the fully qualified file name of the read-only file.

## Return Values

**Table 26** • OnFileReadyOnly Return Values

Return Value	Description
<b>ERR_YES</b>	Returned to the setup by the OnRemovingSharedFile event handler to request that the setup perform the operation on the file.
<b>ERR_NO</b>	Returned to the setup by the OnRemovingSharedFile event handler to request that the setup not perform the operation on the file.

# OnFilesInUse



**Project** • This information applies to InstallScript MSI projects.

The OnFilesInUse event handler is called in an InstallScript MSI installation when the Windows Installer sends an INSTALLMESSAGE\_FILESINUSE message to the installation.

The szMessage parameter contains the string that the Windows Installer provides. This parameter indicates the files that are in use. The **SdFilesInUse** function parses this string appropriately.

By default, the OnFilesInUse event handler displays the **SdFilesInUse** dialog. The event handler returns the value that the dialog returns, and the value is then passed back to the Windows Installer to indicate how the message was handled and what action the Windows Installer should take.

## Syntax

```
OnFilesInUse (szMessage);
```

## Parameters

**Table 27** • OnFilesInUse Parameters

Parameter	Description
<b>szMessage</b>	String provided by the Windows Installer indicating the files that are in use. The <b>SdFilesInUse</b> function parses this string.

## Return Values

**Table 28** • OnFilesInUse Return Values

Return Value	Description
<b>IDCANCEL</b>	Indicates that the installation should be cancelled.
<b>IDRETRY</b>	Indicates that the Windows Installer should recheck for in-use files and send the <b>INSTALLMESSAGE_FILESINUSE</b> message if it still needed.
<b>IDIGNORE</b>	Indicates that the Windows Installer should ignore the fact that the files are in use and should continue the installation.

## OnHelp



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The OnHelp event handler responds to the Help event generated when the end user presses the F1 key.

```
function OnHelp( )
begin
    /* Assumes that MySetupHelp.chm is located
       in the Support Files/Billboards view. */
    LaunchAppAndWait( WINDIR ^ "Hh.exe",
                     SUPPORTDIR ^ "MySetupHelp.chm",
                     NOWAIT );
end;
```

## OnInternetError



**Project** • This information applies to *InstallScript* projects.

The OnInternetError event handler responds to the FileError event that is generated when the setup encounters a file error that does not generate any other file error event (for example, FileLocked or FileReadOnly). When creating an InstallShield object, note that this event is not triggered in an object.

This event handler is called (when appropriate) in any setup, including a setup that uses a procedural script (a script with a program...endprogram block).

## Syntax

```
OnInternetError ( hInternet, szFilename, nError );
```

## Parameters

**Table 29** • OnInternetError Parameters

Parameter	Description
<b>hInternet</b>	Internal handle—ignore.
<b>szFilename</b>	Specifies the fully qualified file name of the file for which the error occurred.
<b>nError</b>	Specifies the value returned by the Windows API function GetLastError when the error occurred.

## Return Values

**Table 30** • OnInternetError Return Values

Return Value	Description
<b>ERR_IGNORE</b>	Returned to the setup by the OnInternetError event handler to request that the setup ignore the failure to install or uninstall the file specified in OnInternetError's second argument and continue without performing the operation.
<b>ERR_RETRY</b>	Returned to the setup by the OnInternetError event handler to request that the setup attempt again to install or uninstall the file specified in OnInternetError's second argument.
<b>ERR_ABORT</b>	Returned to the setup by the OnInternetError event handler to request that the setup be aborted.

## OnLaunchAppAndWaitCallback



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The OnLaunchAppAndWaitCallback event handler is called while the installation is waiting for an application to launch if LAAW\_OPTION\_USE\_CALLBACK was specified when calling the **LaunchApplication** function. The event is called in the intervals determined by the amount of time specified in the LAAW\_PARAMETERS.nCallbackInterval parameter.

If the installation contains multiple **LaunchApplication** calls in which LAAW\_OPTION\_USE\_CALLBACK was specified, the same event is called during each wait. In this case, use the LAAW\_PARAMETERS.szCommandLineResult parameter to determine which call is currently being executed. You can return LAAW\_CALLBACK\_RETURN\_CONTINUE\_TO\_WAIT to continue waiting or LAAW\_CALLBACK\_RETURN\_END\_WAIT to end the wait immediately.

```
function number OnLaunchAppAndWaitCallback( )

begin
    return LAAW_CALLBACK_RETURN_CONTINUE_TO_WAIT;
end;
```



**Note** • This event handler appears in installations and object projects. Any specified override for this event applies on to the script in which it is overwritten. An override in the main installation script does not affect contained objects and an override in an object project has no effect on the main installation script.

## OnLogonUserSetMsiProperties



**Project** • This information applies to InstallScript MSI projects.

The OnLogonUserSetMsiProperties event handler sets the Windows Installer properties for the logon user support that is described for InstallScript MSI projects in Adding the Ability to Create or Set an Existing User Account. Specifically, OnLogonUserSetMsiProperties sets the following

- The Windows Installer property **IS\_NET\_API\_LOGON\_USERNAME** is set to the InstallScript variable IFX\_NETAPI\_USER\_ACCOUNT
- The Windows Installer property **IS\_NET\_API\_LOGON\_PASSWORD** is set to the InstallScript variable IFX\_NETAPI\_PASSWORD
- The Windows Installer property **IS\_NET\_API\_LOGON\_GROUP** is set to the InstallScript variable IFX\_NETAPI\_GROUP.

The IFX\_NETAPI\_\* variables are set to the values of the input fields in the **SdLogonUserInformation** and related dialogs.

## OnMD5Error



**Project** • This information applies to InstallScript projects.

The OnMD5Error event handler responds to the MD5Error event that is generated during MD5 checking when the setup extracts a file whose MD5 hash value does not correspond to the value stored in the setup header file. When creating an InstallShield object, note that this event is not triggered in an object.

This event handler is called (when appropriate) in any setup, including a setup that uses a procedural script (a script with a program...endprogram block).



**Tip** • MD5 checking can detect corrupted files, which is useful during Internet setups; not doing MD5 checking can make file transfer proceed faster. You can enable or disable MD5 checking with the *Advanced* button in the Release Wizard's General Options panel, or with the *Setup.ini* file's [Startup] section's CheckMD5 key.

## Syntax

```
OnMD5Error ( szFilename );
```

## Parameters

**Table 31** • OnMD5Error Parameters

Parameter	Description
<b>szFilename</b>	Specifies the fully qualified file name of the file that generated the MD5 error.

## Return Values

**Table 32** • OnMD5Error Return Values

Return Value	Description
<b>ERR_IGNORE</b>	Returned to the setup by the OnMD5Error event handler to request that the setup ignore the failure to install or uninstall the file specified in OnMD5Error's argument and continue without performing the operation.
<b>ERR_RETRY</b>	Returned to the setup by the OnMD5Error event handler to request that the setup attempt again to install or uninstall the file specified in OnMD5Error's argument.
<b>ERR_ABORT</b>	Returned to the setup by the OnMD5Error event handler to request that the setup be aborted.

## Additional Information

An MD5Error event will be generated at run time if, after running the media builder, you replace an uncompressed file in a disk image folder. You can modify the default OnMD5Error code to automatically handle this MD5Error event rather than display a dialog. For example, to automatically ignore the MD5Error event for the file ReplacedAfterBuild.txt, which is installed to TARGETDIR, add the following lines at the beginning of the OnMD5Error code:

```
if szFilename = TARGETDIR ^ "ReplacedAfterBuild.txt" then
    return ERR_IGNORE;
endif;
```

## OnMsiSilentInstall



**Project** • This information applies to InstallScript MSI projects.

An installation program created by an InstallScript MSI project requires the end user to run **Setup.exe**. The OnMsiSilentInstall event handler responds to the end user attempting to run an InstallScript MSI project's .msi database in silent mode with the /q option to **MsiExec.exe**. The default implementation is to display an error message and then abort the installation.

## OnNextDisk



**Project** • This information applies to InstallScript projects.

*Because the Windows Installer handles disk prompts, this event is not supported for InstallScript MSI projects.*

The OnNextDisk event handler responds to the NextDisk event that occurs in a multi-disk setup when the next disk in the sequence is required to continue the setup.

### Return Values

**Table 33** • OnNextDisk Return Values

Return Value	Description
<b>ERR_RETRY</b>	Returned to the setup by the OnNextDisk event handler to request that the setup search again for the file specified in OnNextDisk's first argument.
<b>ERR_ABORT</b>	Returned to the setup by the OnNextDisk event handler to request that the setup be aborted.

## OnOutOfDiskSpace



**Project** • This information applies to InstallScript MSI projects.

The OnOutOfDiskSpace event handler responds to the Out Of Disk Space event. The default implementation of OnOutOfDiskSpace displays the [SdDiskSpace2](#) dialog, and then aborts the installation.

## OnPatchUIAfter



**Project** • This information applies to InstallScript MSI projects.

The OnPatchUIAfter event handler is called after data transfer for a patch installation. The default implementation of OnPatchUIAfter calls the [SdFinishEx](#) function.

## OnPatchUIBefore



**Project** • This information applies to InstallScript MSI projects.

The OnPatchUIBefore event handler is called when the user launches a patch installation. The default implementation of OnPatchUIBefore calls the [SdPatchWelcome](#) dialog function.

## OnRebooted



**Project** • This information applies to the following project types:

- InstallScript
- InstallScript MSI—if the InstallScript user interface (UI) style is the traditional style (which uses the InstallScript engine as an external UI handler)

*This information does not apply to InstallScript MSI projects in which the InstallScript UI style is the new style (which uses the InstallScript engine as an embedded UI handler). To learn more, see [Using the InstallScript Engine as an External vs. Embedded UI Handler for InstallScript MSI Installations](#).*

The OnRebooted event handler responds to the Rebooted event that the installation generates when it restarts after the target system is rebooted. When the installation restarts after reboot, OnRebooted is the only event handler it calls.

## OnRemovingSharedFile



**Project** • This information applies to InstallScript projects.

The OnRemovingSharedFile event handler responds to the RemovingSharedFile event that occurs during uninstallation when a file that might be shared with other applications is about to be removed. When creating an InstallShield object, note that this event is not triggered in an object.

This event handler is called (when appropriate) in any setup, including a setup that uses a procedural script (a script with a program...endprogram block).

### Syntax

```
OnRemovingSharedFile ( szFilename );
```



## Parameters

**Table 34 • OnRemovingSharedFile Parameters**

Parameter	Description
<b>szFilename</b>	Specifies the fully qualified file name of the shared file.

## Return Values

**Table 35 • OnRemovingSharedFile Return Values**

Return Value	Description
<b>ERR_YES</b>	Returned to the setup by the OnRemovingSharedFile event handler to request that the setup remove the file.
<b>ERR_NO</b>	Returned to the setup by the OnRemovingSharedFile event handler to request that the setup not remove the file.

## OnResumeUIAfter



**Project •** This information applies to InstallScript MSI projects.

The OnResumeUIAfter event handler responds to the Resume UI After event. It performs tasks that must take place after an application's reinstallation or minor upgrade.

The OnResumeUIAfter and OnResumeUIBefore event handlers are called only when all of the following conditions apply:

- The application is already installed on the target machine.
- A patch is *not* being run.
- One of the following properties is set at the command line or in the CmdLine property of Setup.ini:
  - ADDDEFAULT
  - ADDLOCAL
  - ADDSOURCE
  - ADVERTISE
  - COMPADDLOCAL
  - COMPADDSOURCE
  - FILEADDDEFAULT
  - FILEADDLOCAL
  - FILEADDSOURCE

- REINSTALL
- REMOVE

## OnResumeUIBefore



**Project** • This information applies to InstallScript MSI projects.

The OnResumeUIBefore event handler responds to the Resume UI Before event. It performs tasks that must take place before an application's reinstallation or minor upgrade.

The OnResumeUIBefore and OnResumeUIAfter event handlers are called only when all of the following conditions apply:

- The application is already installed on the target machine.
- A patch is *not* being run.
- One of the following properties is set at the command line or in the CmdLine property of Setup.ini:
  - ADDDEFAULT
  - ADDLOCAL
  - ADDSOURCE
  - ADVERTISE
  - COMPADDLOCAL
  - COMPADDSOURCE
  - FILEADDDEFAULT
  - FILEADDLOCAL
  - FILEADDSOURCE
  - REINSTALL
  - REMOVE

## OnRMFilesInUse



**Project** • This information applies to InstallScript MSI projects.

The OnRMFilesInUse event handler is called in an InstallScript MSI installation when the Restart Manager is enabled and Windows Installer 4.0 sends an INSTALLMESSAGE\_RMFILESINUSE message to the installation.

Note that the INSTALLMESSAGE\_RMFILESINUSE message is not sent if the Restart Manager is unavailable or disabled. For more information, see MSIRESTARTMANAGERCONTROL Property in the Windows Installer Help Library.

If the Restart Manager is unavailable or disabled, or if the target system has Windows Installer 3.x or earlier, Windows Installer sends an `INSTALLMESSAGE_FILESI` message to the installation.

By default, the `OnRMFilesInUse` event handler displays the **SdRMFilesInUse** dialog. The event handler returns the value that the dialog returns, and the value is then passed back to the Windows Installer to indicate how the message was handled and what action the Windows Installer should take.

## Syntax

```
OnRMFilesInUse (szMessage);
```


## Parameters

**Table 36 • OnRMFilesInUse Parameters**

Parameter	Description
<b>szMessage</b>	String provided by the Windows Installer indicating the files that are in use. The <b>SdRMFilesInUse</b> function parses this string.

## Return Values

**Table 37 • OnRMFilesInUse Return Values**

Return Value	Description
<b>IDCANCEL</b>	Indicates that the installation should be cancelled.
<b>IDRETRY</b>	Indicates that the Windows Installer should recheck for in-use files and send the <code>INSTALLMESSAGE_RMFILESI</code> message if it still needed.   <b>Note</b> • Unlike the <i>SdFilesInUse</i> dialog, the <i>SdRMFilesInUse</i> dialog does not return this value.
<b>IDIGNORE</b>	Indicates that the Windows Installer should ignore the fact that the files are in use and should continue the installation.
<b>IDOK</b>	Indicates that the Windows Installer should use the Restart Manager to attempt to shut down running applications that are locking files. For more information, see <code>INSTALLMESSAGE_RMFILESI</code> in the Windows Installer Help Library.

## OnSelfRegistrationError



**Project** • This information applies to InstallScript projects.

The `OnSelfRegistrationError` event handler is called directly by the framework when a call to `Do(SELFREGISTRATIONPROCESS)` fails to register files successfully.

## Syntax

```
OnSelfRegistrationError ( );
```

## Parameters

None.

## Return Values

None.

## Additional Information

The default OnSelfRegistrationError code uses the following properties of the global FileRegistrar object:

**Table 38** • Global FileRegistrar Object Properties

Property	Description
<b>FileRegistrar.Errors.Count</b>	The number of self-registering files that were not registered.
<b>FileRegistrar.Errors(i)</b>	An object whose properties provide information about the <i>i</i> -th unregistered file. ( <i>i</i> runs from 1 to the value of FileRegistrar.Errors.Count.)
<b>FileRegistrar.Errors(i).File</b>	The name of the <i>i</i> -th unregistered file.
<b>FileRegistrar.Errors(i).Description</b>	A string describing the error that occurred when the setup attempted to register the <i>i</i> -th unregistered file. This string may be null ("").
<b>FileRegistrar.Errors(i).LastError</b>	The numeric code for the error that occurred when the setup attempted to register the <i>i</i> -th unregistered file.

## OnWarning



**Project** • This information applies to InstallScript MSI projects.

The OnWarning event handler responds to the Warning event, which is raised when the Windows Installer service sends a INSTALLMESSAGE\_WARNING message.

# Advanced Event Handlers

Advanced event handlers are triggered under special circumstances.

**Table 39 • Advanced Event Handlers**

Event Handler	Project Type	Description
<b>OnShowUI</b>	InstallScript	Called directly by the installation engine to initiate the user interface and file transfer.
<b>OnSuiteShowUI</b>	InstallScript	When an Advanced UI or Suite/Advanced UI installation launches an InstallScript package, the OnSuiteShowUI event is called instead of the OnShowUI event. By default, the OnSuiteShowUI event initializes any feature states that are passed from an Advanced UI or Suite/Advanced UI installation to the InstallScript package, and it starts file transfer.
<b>OnUninstall</b>	InstallScript, InstallScript MSI	The OnUninstall event handler is called when the installation is run with the /uninst parameter. This is the only event that is called when the /uninst parameter is used. The default code for the OnUninstall event uninstalls a previously installed product.

## OnShowUI

The OnShowUI event drives the UI sequence and file transfer of the installation.

The OnShowUI event is called directly by the framework to initiate the UI sequence and file transfer of the installation. By default, this event displays UI that informs the end user that maintenance has been completed successfully.



**Note** • This event is not called automatically in a program...endprogram style installation.

### Syntax

```
OnShowUI ( );
```

### Parameters

None.

### Return Values

None.

## Additional Information

You can easily convert a procedural script to an event-based script by performing the following steps. If you do this, neither the user interface event handlers (OnFirstUIBefore, OnMaintUIBefore, OnUpdateUIBefore, OnFirstUIAfter, OnMaintUIAfter, and OnUpdateUIAfter) nor OnMoveData are called; but all other feature event handlers are called as appropriate.



### Task

#### **To convert a procedure script to an event-based script:**

1. Open your project in InstallShield; when you are prompted to convert it, click Yes.
2. Open the script and change the following line:

```
program
```

to this:

```
function OnShowUI  
begin
```

3. Also change the following line:

```
endprogram
```

to this:

```
end
```

4. If you are planning to display any custom objects that include a user interface (objects provided with InstallShield do not), add a call to [ShowObjWizardPages](#) to your dialog sequence at the location where you want any object UI to be shown:

```
ShowObjWizardPages( nResult );
```

This function can be called before or after file transfer; the appropriate object event is called based on whether or not file transfer has occurred and on the value of the MAINTENANCE system variable.

5. Make the changes needed to compile the script in InstallShield.

You can also customize the OnBegin and OnEnd events if you wish.

## Why This Works

In InstallScript, most installation events are driven by the main UI event, OnShowUI. When you replace the default code in this event with the code in your program...endprogram block, you are basically providing a custom UI sequence. Note that since OnShowUI is called when the installation is run, your installation behaves just like it did when it was a program...endprogram script.

## Launching an InstallScript Package in an Advanced UI or Suite/Advanced UI Installation

Note that when an Advanced UI or Suite/Advanced UI installation launches an InstallScript package, the [OnSuiteShowUI event](#) is called instead of the OnShowUI event.

## OnSuiteShowUI



**Project** • This information applies to *InstallScript* projects.

When an Advanced UI or Suite/Advanced UI installation launches an *InstallScript* package, the *OnSuiteShowUI* event is called instead of the *OnShowUI* event. By default, the *OnSuiteShowUI* event initializes any feature states that are passed from an Advanced UI or Suite/Advanced UI installation to the *InstallScript* package, and it starts file transfer. The *OnSuiteShowUI* event does not typically show any user interface, since the Advanced UI or Suite/Advanced UI installation's setup launcher typically handles the user interface for the entire Advanced UI or Suite/Advanced UI installation.

Depending on the installation state (first-time installation, maintenance, or update), *OnSuiteShowUI* ignores the UI events such as *OnFirstUIBefore* and *OnFirstUIAfter* and instead calls the following events:

- **First-time installation**—*OnSuiteInstallBefore*, *OnSuiteInstallAfter*
- **Maintenance**—*OnSuiteMaintBefore*, *OnSuiteMaintAfter*
- **Update**—*OnSuiteUpdateBefore*, *OnSuiteUpdateAfter*

All other events and event call sequencing in an *InstallScript* package that is launched from an Advanced UI or Suite/Advanced UI installation remain the same as in an *InstallScript* installation that is launched separate from an Advanced UI or Suite/Advanced UI installation, or that is launched as an executable package from an Advanced UI or Suite/Advanced UI installation.



**Note** • This event is not called automatically in a *program...endprogram* style installation.

## OnUninstall



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The *OnUninstall* event is called when the installation is run with the `/uninst` parameter. This is the only event that is called when the `/uninst` parameter is used. The default code for the *OnUninstall* event uninstalls a previously installed product.





# Functions

A function is a named set of instructions that operate together to perform a specific task.

## Function Characteristics

Every function has the following characteristics:

- A function is named. Each function has a unique name. When you call the function by name, you know which set of instructions will run, and you can be sure of consistent results. You can also call a function from within another function.
- A function is independent. In most cases, any function can perform its instructions without interfering with other parts of the program.
- A function performs a certain task. A task is any single job that the script must perform, such as displaying a bitmap, compressing a file, or creating a folder.
- A function can return a value to the script. When the script executes, it performs the instructions of the function. Based on the result of the instructions, a function can return information to the script.

## Function Types

InstallShield allows you to use three types of functions in your setup script:

**Table 1 • Function Types**

Function Type	Description
<b>Built-in functions</b>	Functions supplied by InstallShield or included for Sd dialogs.
<b>User-defined functions</b>	Functions that you create.
<b>DLL-called functions</b>	Functions that you can call in a DLL.



**Note** • Like the C programming language, InstallScript does not support nested function blocks.

## Using Built-In Functions

InstallShield has several hundred built-in functions that you can use in your setup scripts to create program groups and items, manipulate folders, work with lists, monitor the status of the setup, create dialogs, manipulate files, and much more. Because the InstallShield Script Compiler already recognizes these function names, you do not have to declare them before you can use them.

## Learning About Function Names and Formats

In order call a built-in function, you must know both its name and its format.

To find a function that will fit your requirements, review [Built-In Functions by Category](#), which describes each of the major categories of available functions. Click a category link to view a list of functions and descriptions that pertain to that category.

All of the built-in functions are listed alphabetically in the Built-in Functions sections. To display a complete description of any function in that list, click its name. The help topic for that function provides the function's format.

For example, `AskYesNo` is a built-in function that displays a query in a dialog and then waits for the end user to respond by clicking a button, either Yes or No. `AskYesNo` has the following format:

```
AskYesNo (szQuestion, nDefault);
```

The format shows the correct spelling of the function name, which is followed by the function's parameter list, enclosed in parentheses. In the help topic for a built-in function, each parameter is expressed in Hungarian notation, which indicates the type of data that must be passed in that position. `AskYesNo` requires two parameters: the first parameter is a string; the second is a number.




---

**Note** • Like C, `InstallScript` is case sensitive. Be sure to pay close attention to the capitalization of letters in the built-in function names.

## Using Built-In Functions in Your Script

To use a built-in function in your script, pass the required number of parameters, and make sure that the data you pass in each parameter is the type indicated for that position. If you pass the incorrect number of parameters, or if you pass the wrong type of data in any parameter positions, the script will not compile.

The specific documentation for each built-in function provides a description of its parameters. For `AskYesNo`, `szQuestion` is the question to be displayed in the dialog and `nDefault` indicates which button to preselect—Yes or No. One of two predefined constants can be passed in `nDefault`: YES or NO.

Consider a dialog in which the Yes button is preselected. To display this dialog, call `AskYesNo` as shown below:

```
AskYesNo ("Installation Complete. Would you like to view the ReadMe file now?", YES);
```




---

**Note** • String literals passed as parameters must be enclosed in single or double quotation marks, for example: "Please wait while files are transferred", or 'This is a string', or "C:\\Myfolder\\Myfile.ext".

## Troubleshooting

Note the following:

- `InstallScript` functions do not allow you to pass an assignment statement as a parameter. In addition, you cannot use the `&&` or `||` operators within an argument to a function.
- An autosized string variable that is passed by reference to a function will not be autosized within the called function. If the function attempts to assign a value whose length is greater than the current size of that parameter, run-time error 401 occurs. To avoid this error, declare strings with a specific size when they are to be passed by reference to a function.

# Built-In Functions by Category

The following categories of functions are available in the InstallScript language.

**Table 2 • Built-In Functions by Category**

Function Category	Description
<b>Batch File Functions</b>	Work with batch files.
<b>Component Functions</b>	In InstallShield, feature functions replace the component functions that were used in InstallShield Professional.
<b>Configuration File Functions</b>	Modify the default system configuration file.
<b>Device Driver Functions</b>	Install and uninstall device drivers using DIFx.
<b>Dialog Functions</b>	Create InstallScript dialogs and message boxes.
<b>Dialog Customization Functions</b>	Customize InstallScript dialogs, and modify the text, controls, and behavior of InstallScript dialogs.
<b>Extensibility Functions</b>	Call functions in dynamic-link libraries, call Windows APIs, launch another application or setup script, and load or unload a .dll file into memory.
<b>Feature Functions</b>	Control file media. Also create and process script-created feature sets.
<b>File and Folder Functions</b>	Work with text files, binary files, and folders.
<b>FlexNet Connect Functions</b>	Enables you to use FlexNet Connect to notify end users about updates that are available for your product.
<b>Information Functions</b>	Provide data about resources that are available in the operating environment: disk space, memory, and operating mode.
<b>Initialization File Functions</b>	Obtain information from and copy information to the initialization and profile files.
<b>List Processing Functions</b>	Implement lists in a setup script.
<b>Log File Functions</b>	Obtain information from and copy information to the log file's custom logging section.
<b>Long File Name Functions</b>	Create long file names from short file names, convert short file names to long file names, and place double quotation marks around long file names so that operating systems that handle long file names can recognize them.
<b>Miscellaneous Functions</b>	Serve various purposes, such as low-level hardware interface, feature creation and manipulation, and user output.

**Table 2 • Built-In Functions by Category (cont.)**

Function Category	Description
<b>Object Functions</b>	Initialize objects, as well as obtain and set object status information.
<b>Path Buffer Functions</b>	Work with strings that contain search paths. The path string functions work on a unique temporary string variable known as the path buffer.
<b>Registry Functions</b>	Access the registry, read, create and delete registry keys, and establish registry-related parameters for uninstallation.
<b>Shared and Locked File Functions</b>	Handle shared or locked files.
<b>Shell Functions</b>	Create shortcuts, delete existing shortcuts, and configure shortcuts.
<b>Special Registry-Related Functions</b>	Set up the minimum required registry keys and values. The special registry-related functions work only with the per application paths key, the application uninstallation key, or the application information key.
<b>SQL Functions</b>	Perform SQL tasks such as connect to a catalog, create SQL-related dialogs, and obtain SQL run-time errors.
<b>String Functions</b>	Manipulate string variables and literals. String functions behave similarly to the standard C language functions. The return values also follow the C language convention.
<b>Suite/Advanced UI and Advanced UI Interaction Functions</b>	Interact with an Advanced UI or Suite/Advanced UI installation that contains an InstallScript package, or with a Suite/Advanced UI installation that includes InstallScript actions.
<b>Text Substitutions</b>	Associate a string with another string—for example, associating "<MYTEXTSUB>" with "My Text Sub Value"—and of replacing the former string with the latter in other strings—for example, changing "This string demonstrates text substitution <MYTEXTSUB>" to "This string demonstrates text substitution My Text Sub Value".
<b>Uninstallation Functions</b>	Perform services required for the uninstallation and/or maintenance setup of an installed application.
<b>User Interface Functions</b>	Customize certain error messages and titles of error boxes. Note that several internal error messages that may be encountered during the development of the setup cannot be modified using user interface functions.
<b>Version-Checking Functions</b>	Obtain a specific file's version, find a file and get its version, or search for an existing file and try to install a newer version of the file. The functions work with either compressed or uncompressed files.
<b>Windows Installer Functions</b>	Exported by the Windows Installer engine. These functions enable you to query and manipulate properties of a running installation.

# Batch File Functions

When you work with batch and configuration files, you can treat the files as normal text files, or you can use InstallScript functions designed specifically to modify batch and configuration files.

## Ez and Advanced Batch Functions

There are two classes of InstallScript batch file functions—Ez and advanced. Ez functions are quick and easy to use because they have many preprogrammed features. If you need greater flexibility and control over the changes you need to make to configuration and batch files, use the advanced functions.

## Return Value

InstallShield does not display a message if a batch function fails, but returns a value that is less than zero (< 0). The return value indicates whether the function performed successfully. You can check the return value and display a message based on the result.

## Ez Batch File Functions

Ez batch file functions make modifications to the default batch file. Unless changed by a call to [BatchSetFileName](#), the default batch file is the Autoexec.bat file that was executed by the system during the boot sequence. It is important to note that each Ez batch file function opens the default batch file and then saves it automatically after making the specified modification. You do not make calls to open or save when using Ez batch file functions.

**Table 3 • Ez Batch File Functions**

Function	Description
<a href="#">EzBatchAddPath</a>	Modifies the default batch file by adding a directory to the search path in a PATH command or to the value assigned to an environment variable.
<a href="#">EzBatchAddString</a>	Adds a line of text to the default batch file.
<a href="#">EzBatchReplace</a>	Replaces a statement in the default batch file.

## Advanced Batch File Functions

Advanced batch file functions differ from Ez batch file functions in that they provide greater flexibility and more control over batch files. Use these functions when you need to make more extensive or complex changes to a batch file.

To edit a batch file with these advanced functions, you must first load the file into memory by calling [BatchFileLoad](#). When modifications to the batch file are complete, you must then save the file by calling [BatchFileSave](#).

When an InstallScript custom action initializes, InstallShield selects the target system's startup batch file (Autoexec.bat) as the default batch file; unless changed by a call to [BatchSetFileName](#) this is the file that BatchFileLoad reads into memory if no other file name is specified. To determine the fully qualified name of the default batch file, call [BatchGetFileName](#).

**Table 4 • Advanced Batch File Functions**

Function	Description
<a href="#">BatchAdd</a>	Adds an environment variable to a batch file.
<a href="#">BatchDeleteEx</a>	Deletes a line in the batch file.
<a href="#">BatchFileLoad</a>	Loads a batch file into memory for editing with advanced batch functions.
<a href="#">BatchFileSave</a>	Saves a batch file that has been loaded with BatchFileLoad.
<a href="#">BatchFind</a>	Finds items in a batch file.
<a href="#">BatchGetFileName</a>	Retrieves the fully qualified file name of the default batch file.
<a href="#">BatchMoveEx</a>	Moves an item within a batch file.
<a href="#">BatchSetFileName</a>	Specifies a batch file to be the default batch file.

## Related Function

**Table 5 • Related Function**

Function	Description
<a href="#">SdShowFileMods</a>	Creates a dialog that displays proposed file changes and offers options on how to proceed.

## Component Functions

Because a Windows Installer-based installation uses features as the highest level of organization in installation projects, and not components, component-related functions are deprecated in InstallShield. All supported component functions now have corresponding feature functions that should be used in your script.

For example, the ComponentAddItem function is now [FeatureAddItem](#). For more information and a list of available feature-related functions, see [Feature Functions](#)

## Configuration File Functions

Configuration file functions modify the default system configuration file. There are two types of configuration file functions:

- [Ez Config.sys File Functions](#)

- Advanced Configuration File Functions

## Ez Config.sys File Functions

Ez configuration file functions modify the default system configuration file. Unless changed by a call to [ConfigSetFileName](#), that file is the Config.sys file that was executed by the system during the boot sequence.

**Table 6 • Ez Configuration File Functions**

Function	Description
<b>EzConfigAddDriver</b>	Adds a device driver statement to the default system configuration file.
<b>EzConfigAddString</b>	Adds a statement or line of text to the default system configuration file.
<b>EzConfigGetValue</b>	Retrieves the value of a system configuration file parameter, such as FILES or BUFFERS.
<b>EzConfigSetValue</b>	Sets the value of a system configuration file parameter, such as FILES or BUFFERS.



**Note •** Each of these functions opens the default system configuration file, performs its assigned task, and then saves the file back to disk. It is not necessary to load and save the configuration file as with the advanced configuration file functions.

## Advanced Configuration File Functions

The advanced configuration file functions provide the advanced developer greater flexibility and more control over system configuration files than do the Ez configuration file functions. To access and edit a system configuration file with these advanced functions, start by calling [ConfigFileLoad](#). When you are finished editing the system configuration file, call [ConfigFileSave](#) to save your changes. Note that the functions [ConfigGetFileName](#) and [ConfigSetFileName](#) can be used with both advanced and Ez configuration file functions.

**Table 7 • Advanced Configuration File Functions**

Function	Description
<b>ConfigAdd</b>	Adds a statement to a system configuration file that has been loaded in memory.
<b>ConfigDelete</b>	Deletes an item from a system configuration file.
<b>ConfigFileLoad</b>	Loads a system configuration file into memory for editing.
<b>ConfigFileSave</b>	Saves a system configuration file that has been loaded into memory with <a href="#">ConfigFileLoad</a> .

**Table 7 •** Advanced Configuration File Functions (cont.)

Function	Description
<b>ConfigFind</b>	Searches for an item in a system configuration file.
<b>ConfigGetFileName</b>	Retrieves the fully qualified name of default system configuration file.
<b>ConfigGetInt</b>	Retrieves a value from a system configuration file.
<b>ConfigMove</b>	Moves an item within a system configuration file.
<b>ConfigSetFileName</b>	Specifies the fully qualified file name of a system configuration file.
<b>ConfigSetInt</b>	Sets a value in a system configuration file.

## Related Function

**Table 8 •** Related Function

Function	Description
<b>SdShowFileMods</b>	Creates a dialog displaying proposed file changes and offering options on how to proceed.

# Device Driver Functions

The functions below handle installing and uninstalling device drivers using Windows Driver Install Frameworks (DIFx). For more information on DIFx and DIFxAPI, see the MSDN Library.

**Table 9 •** Device Driver Functions

Function	Description
<b>DIFxDriverPackageGetPath</b>	Retrieves the path to the .inf file for driver package after a driver package is preinstalled in the driver store.
<b>DIFxDriverPackageInstall</b>	Preinstalls a driver package in the driver store and then installs the driver in the system.
<b>DIFxDriverPackagePreinstall</b>	Preinstalls a driver package for a Plug and Play (PnP) function driver in the driver store and installs the .inf file for the driver package in the system .inf file directory.
<b>DIFxDriverPackageUninstall</b>	Uninstalls the specified driver package from the system and removes the driver package from the driver store.



# Dialog Functions

Some of the following functions create simple dialogs, such as Yes/No dialogs and message boxes. Several functions let you easily display various types of common dialogs. Other functions are script dialog (Sd) functions. Sd dialogs are created using special InstallScript definition functions that create a dialog with custom input. The dialogs then automatically return values to the script based on the selected action.



**Note** • Dialogs that have a Cancel button do not return a CANCEL value when that button is clicked. Instead, the [OnCanceling](#) event handler is called.

**Table 10** • Dialog Functions

Function	Project Type	Description
<a href="#">AdminAskPath</a>	InstallScript MSI	Displays a dialog that prompts the end user to enter the path to a destination location for an administrative installation (when the end user runs an InstallScript MSI project's Setup.exe file with the /a argument).
<a href="#">AskDestPath</a>	InstallScript, InstallScript MSI	Presents a dialog that requests destination path information.
<a href="#">AskOptions</a>	InstallScript, InstallScript MSI	Presents a dialog that prompts the end user to select options by using check boxes or radio buttons.
<a href="#">AskPath</a>	InstallScript, InstallScript MSI	Presents a dialog that prompts the end user to enter a path.
<a href="#">AskText</a>	InstallScript, InstallScript MSI	Presents a dialog that prompts the end user to enter text.
<a href="#">AskYesNo</a>	Basic MSI, InstallScript, InstallScript MSI	Presents a message box that prompts the end user to respond to a question by clicking on a Yes or No button.
<a href="#">EnterDisk</a>	Basic MSI, InstallScript, InstallScript MSI	Presents a message box that prompts the end user for a specific disk.
<a href="#">EnterDiskError</a>	Basic MSI, InstallScript, InstallScript MSI	Checks whether a specified path and file exists. The function displays an appropriate error message box if the file does not exist in the specified path; then it returns success or failure, depending on whether the specified file exists.



**Table 10 •** Dialog Functions (cont.)

Function	Project Type	Description
<b>EnterLoginInfo</b>	InstallScript, InstallScript MSI	Displays a dialog that enables the end user to specify a user name and password. Note that the dialog does not validate or use the specified information. In addition, the dialog does not perform any error checking for the specified information.
<b>EnterPassword</b>	InstallScript, InstallScript MSI	Displays a dialog that queries the end user for a password; the characters that the end user types in the edit box are displayed as asterisks (*).
<b>FeatureDialog</b>	InstallScript, InstallScript MSI	Presents a dialog that enables end users to select features and specify a destination location.
<b>MessageBox</b>	Basic MSI, InstallScript, InstallScript MSI	Presents a message in a message box.
<b>MessageBoxEx</b>	Basic MSI, InstallScript, InstallScript MSI	Presents a message in a message box.
<b>RebootDialog</b>	InstallScript, InstallScript MSI	Presents a message box that enables end users to choose to restart the computer.
<b>SdAskDestPath</b>	InstallScript, InstallScript MSI	Creates a dialog that enables end users to select an alternate destination path.
<b>SdAskDestPath2</b>	InstallScript, InstallScript MSI	Creates a dialog that enables end users to select an alternate destination path.
<b>SdAskOptions</b>	InstallScript, InstallScript MSI	Creates a dialog that has greater flexibility than the standard <a href="#">AskOptions</a> function.
<b>SdAskOptionsList</b>	InstallScript, InstallScript MSI	Presents a dialog that enables end users to select and deselect items from a list.
<b>SdBitmap</b>	InstallScript, InstallScript MSI	Displays a bitmap on a dialog.
<b>SdConfirmNewDir</b>	InstallScript, InstallScript MSI	Displays a message box that prompts the user to confirm the folder selection.
<b>SdConfirmRegistration</b>	InstallScript, InstallScript MSI	Displays a message box that prompts the end user to confirm the information that was entered in dialogs presented by <b>SdRegisterUser</b> or <b>SdRegisterUserEx</b> .


**Table 10 •** Dialog Functions (cont.)

Function	Project Type	Description
<b>SdCustomerInformation</b>	InstallScript, InstallScript MSI	Displays a dialog that enables the end user to specify the user name and company name for the product being installed. The dialog may also include radio buttons that let the end user specify whether the product should be installed for all users or only the current user.
<b>SdCustomerInformationEx</b>	InstallScript, InstallScript MSI	Displays a dialog that enables the end user to specify the user name, company name, and serial number for the product being installed. The dialog may also include radio buttons that let the end user specify whether the product should be installed for all users or only the current user.
<b>SdDiskSpace2</b>	InstallScript, InstallScript MSI	Displays a dialog that shows either of the following: <ul style="list-style-type: none"> <li>• A list view of volumes, required space, available space, and the difference between available space and required space.</li> <li>• A warning message indicating that the target system does not have enough available space for the installation to take place. The dialog also displays a list view of volumes, required space, available space, and the difference between available space and required space.</li> </ul>
<b>SdDiskSpaceRequirements</b>	InstallScript, InstallScript MSI	Displays a list of volumes, required space, available space, and the difference between available and required space. <b>SdDiskSpace2</b> supersedes this function.
<b>SdDisplayTopics</b>	InstallScript, InstallScript MSI	Displays a list of topics.
<b>SdExceptions</b>	InstallScript, InstallScript MSI	Displays a message box informing the end user that a shared, locked (in use), or read-only file has been encountered.
<b>SdFeatureDialog</b>	InstallScript, InstallScript MSI	Displays a dialog that enables end users to select features to install and a destination folder.
<b>SdFeatureDialog2</b>	InstallScript, InstallScript MSI	Displays a dialog that enables end users to select folders, features, and subfeatures to install.
<b>SdFeatureDialogAdv</b>	InstallScript, InstallScript MSI	Displays a dialog that enables end users to select features to install and a destination folder.

**Table 10 •** Dialog Functions (cont.)

Function	Project Type	Description
<b>SdFeatureMult</b>	InstallScript, InstallScript MSI	Displays a dialog that enables the end user to select the features and subfeatures to install. Additional information about disk space is also provided to help determine the best location for the installation.
<b>SdFeatureTree</b>	InstallScript, InstallScript MSI	Displays a dialog with a tree control that enables end users to select the features and subfeatures to install. Additional information about disk space is also provided to help determine the best location for the installation.
<b>SdFilesInUse</b>	InstallScript MSI	Displays a dialog that includes a list box containing a list of the applications that are open and are locking files.
<b>SdFinish</b>	InstallScript, InstallScript MSI	Displays a dialog that informs the end user that the setup is complete and offers a choice of options, such as whether to view an information file or launch an application.
<b>SdFinishEx</b>	InstallScript, InstallScript MSI	Displays a dialog informing the end user that the installation is complete.
<b>SdFinishReboot</b>	InstallScript, InstallScript MSI	Displays a dialog that informs the user that the setup is complete and offers a choice of options for restarting Windows and the computer.
<b>SdFinishUpdate</b>	InstallScript, InstallScript MSI	<p>Displays a dialog that indicates that the installation is complete. The dialog includes the option to check for application updates.</p>  <p><b>Note • SdFinishUpdate</b> does not check for updates; to check for updates, add FlexNet Connect API calls in your InstallScript code. For more information, see the FlexNet Connect SDK documentation.</p>
<b>SdFinishUpdateEx</b>	InstallScript, InstallScript MSI	<p>Displays a dialog that indicates that the installation is complete. The dialog includes the option to check for application updates.</p>  <p><b>Note • SdFinishUpdateEx</b> does not check for updates; to check for updates, add FlexNet Connect API calls in your InstallScript code. For more information, see the FlexNet Connect SDK documentation.</p>

**Table 10 •** Dialog Functions (cont.)

Function	Project Type	Description
<b>SdFinishUpdateReboot</b>	InstallScript, InstallScript MSI	<p>Displays a dialog that indicates that the installation is complete. The dialog gives the end user the option to restart the system and to check for application updates.</p>  <p><b>Note • SdFinishUpdateReboot</b> does not check for updates; to check for updates, add FlexNet Connect API calls in your InstallScript code. For more information, see the FlexNet Connect SDK documentation.</p>
<b>SdLicense</b>	InstallScript, InstallScript MSI	<p>Displays a dialog that contains a license agreement in a multi-line edit field. The license agreement is stored in a text file identified in the parameter szLicenseFile.</p> <p>The dialog shows a question in a static text field. The end user responds by clicking the Yes or No button.</p> <p>The <a href="#">SdLicenseEx</a> function supersedes this function.</p>
<b>SdLicense2</b>	InstallScript, InstallScript MSI	<p>Displays a dialog that contains a license agreement in a multi-line edit field. The license agreement is stored in a text file identified in the parameter szLicenseFile.</p> <p>The dialog displays two radio buttons (one for accepting the terms of the license agreement, and one for not accepting them). The Next button becomes enabled when the end user clicks the appropriate button to accept the terms of the license agreement.</p> <p>The <a href="#">SdLicense2Ex</a> function supersedes this function.</p>
<b>SdLicense2Ex</b>	InstallScript, InstallScript MSI	<p>Displays a dialog that contains a license agreement in a multi-line edit field. The license agreement is stored in a text file (.txt) or a rich text file (.rtf) identified in the parameter szLicenseFile.</p> <p>The dialog displays two radio buttons (one for accepting the terms of the license agreement, and one for not accepting them). The Next button becomes enabled when the end user clicks the appropriate button to accept the terms of the license agreement.</p>

**Table 10 •** Dialog Functions (cont.)

Function	Project Type	Description
<b>SdLicense2Rtf</b>	InstallScript, InstallScript MSI	<p>Displays a dialog that contains a license agreement in a multi-line edit field. The license agreement is stored in a text or rich-text format (RTF) file identified in the parameter <code>szLicenseFile</code>.</p> <p>The dialog displays two radio buttons (one for accepting the terms of the license agreement, and one for not accepting them). The Next button becomes enabled when the end user clicks the appropriate button to accept the terms of the license agreement.</p> <p>The <a href="#">SdLicense2Ex</a> function supersedes this function.</p>
<b>SdLicenseEx</b>	InstallScript, InstallScript MSI	<p>Displays a dialog that contains a license agreement in a multi-line edit field. The license agreement is stored in a text file (.txt) or a rich text file (.rtf).</p> <p>The dialog shows a question in a static text field. The end user responds by clicking the Yes or No button.</p>
<b>SdLicenseRtf</b>	InstallScript, InstallScript MSI	<p>Displays a dialog that contains a license agreement in a multi-line edit field. The license agreement is stored in a text or rich-text format (RTF) file identified in the parameter <code>szLicenseFile</code>.</p> <p>The dialog shows a question in a static text field. The end user responds by clicking the Yes or No button.</p> <p>The <a href="#">SdLicenseEx</a> function supersedes this function.</p>
<b>SdLogonUserBrowse</b>	InstallScript, InstallScript MSI	Displays a message box that enables end users to select a domain or server and a user name.
<b>SdLogonUserCreateUser</b>	InstallScript, InstallScript MSI	Displays a dialog that enables end users to enter new user information after the end user clicks the New User Information button on the <b>SdLogonUserInformation</b> dialog.
<b>SdLogonUserInformation</b>	InstallScript, InstallScript MSI	Displays a dialog that prompts the end user for existing user account information or new user information if an account is to be created during the installation.
<b>SdLogonUserListGroup</b>	InstallScript, InstallScript MSI	Displays a dialog that enables end users to select a group from a specified server and populate the Group field in the <b>SdLogonUserCreateUser</b> dialog.
<b>SdLogonUserListServers</b>	InstallScript, InstallScript MSI	Displays a dialog that enables end users to browse for the domain or server with which the user account is associated.

**Table 10 •** Dialog Functions (cont.)

Function	Project Type	Description
<b>SdLogonUserListUsers</b>	InstallScript, InstallScript MSI	Displays a dialog that enables end users to browse and select an existing user for a specified domain or server.
<b>SdOptionsButtons</b>	InstallScript, InstallScript MSI	Displays a dialog with user-defined buttons that provide an end user with various options.
<b>SdOutOfDiskSpace</b>	InstallScript MSI	When triggered by the Windows Installer's <code>INSTALLMESSAGE_OUTOFDISKSPACE</code> message, this dialog displays a message indicating that the target system is out of disk space.  <b>SdDiskSpace2</b> supersedes this function.
<b>SdPatchWelcome</b>	InstallScript MSI	Creates a dialog that displays a welcome message to end users during a patch installation.
<b>SdRegisterUser</b>	InstallScript, InstallScript MSI	Displays a dialog that enables the end user to specify the user name and company name for the product being installed.
<b>SdRegisterUserEx</b>	InstallScript, InstallScript MSI	Displays a dialog that enables the end user to specify the user name, company name, and serial number for the product being installed.
<b>SdRMFilesInUse</b>	InstallScript MSI	Displays a dialog that includes a list box containing a list of the applications that are open and are locking files. The dialog also includes two radio buttons that allow end users to specify whether the installation should attempt to use the Restart Manager to shut down the applications that are locking files or attempting to overwrite the locked files (which most likely results in the need for a reboot to complete the installation).
<b>SdSelectFolder</b>	InstallScript, InstallScript MSI	Presents a dialog that enables end users to select a folder from a list of program folders.
<b>SdSetupCompleteError</b>	InstallScript, InstallScript MSI	Displays a dialog to inform the end user that the installation was interrupted before it could be completed.
<b>SdSetupType</b>	InstallScript, InstallScript MSI	Displays a dialog that enables the end user to select one of the three standard setup types: Typical, Compact, or Custom.
<b>SdSetupType2</b>	InstallScript, InstallScript MSI	Displays a dialog that enables end users to select one of the two standard setup types: Typical or Custom.

**Table 10 •** Dialog Functions (cont.)

Function	Project Type	Description
<b>SdSetupTypeEx</b>	InstallScript MSI	Displays a dialog that enables end users to select standard or custom setup types.
<b>SdShowAnyDialog</b>	InstallScript, InstallScript MSI	Displays a general-purpose dialog from a resource DLL. You cannot receive any input from the end user when showing a dialog with <b>SdShowAnyDialog</b> .
<b>SdShowDlgEdit1</b>	InstallScript, InstallScript MSI	Displays a dialog that has one single-line edit field and other static controls.
<b>SdShowDlgEdit2</b>	InstallScript, InstallScript MSI	Displays a dialog that has two single-line edit fields and other static controls.
<b>SdShowDlgEdit3</b>	InstallScript, InstallScript MSI	Displays a dialog that has three single-line edit fields and other static controls.
<b>SdShowFileMods</b>	InstallScript, InstallScript MSI	Presents a dialog that previews the changes that may be made to a file and enables end users to approve the changes, reject the changes, or request that the changes be written to a file.
<b>SdShowInfoList</b>	InstallScript, InstallScript MSI	Displays a scrollable list of messages in a dialog.
<b>SdShowMsg</b>	InstallScript, InstallScript MSI	Displays a message in a small window.
<b>SdStartCopy</b>	InstallScript, InstallScript MSI	Presents a dialog that displays the options and settings that have been specified by the end user.
<b>SdStartCopy2</b>	InstallScript, InstallScript MSI	Presents a dialog that informs the end user that the file transfer process is about to begin. The user can click the Back button to return to previous dialogs in order to change settings as required.
<b>SdWelcome</b>	InstallScript, InstallScript MSI	Displays a general-purpose greeting.
<b>SdWelcomeMaint</b>	InstallScript, InstallScript MSI	Displays a dialog for use at the beginning of a maintenance setup.
<b>SelectDir</b>	InstallScript, InstallScript MSI	Presents a dialog that enables end users to select a folder. SelectDir creates the folder if it does not exist.
<b>SelectDirEx</b>	InstallScript, InstallScript MSI	Presents a dialog that enables end users to select a folder.



**Table 10 •** Dialog Functions (cont.)

Function	Project Type	Description
<b>SelectFolder</b>	InstallScript, InstallScript MSI	Presents a dialog that enables end users to select a folder from a list of program folders.
<b>SetupType</b>	InstallScript, InstallScript MSI	Presents a dialog that enables the end user to select a typical, compact, or custom setup.
<b>SetupType2</b>	InstallScript, InstallScript MSI	Presents a dialog that enables the end user to select one of the two standard setup types: Complete or Custom.
<b>SprintfBox</b>	Basic MSI, InstallScript, InstallScript MSI	Returns a formatted string composed of one or more character, numeric, or string values.
<b>SQLBrowse</b>	InstallScript, InstallScript MSI	Creates a dialog that enables the end user to bring up a list of all SQL Servers available on the network.  <b>SQLBrowse2</b> supersedes this function.
<b>SQLBrowse2</b>	InstallScript, InstallScript MSI	Creates a dialog that lets an end user display a list of all database servers that are available on the network for the database technologies specified for a connection.
<b>SQLServerLogin</b>	InstallScript, InstallScript MSI	Creates a dialog that is used by the script to specify SQL login credentials. These credentials include the login ID and password.
<b>SQLServerSelect</b>	InstallScript, InstallScript MSI	Creates a dialog to specify a server to target.
<b>SQLServerSelectLogin</b>	InstallScript, InstallScript MSI	Creates a login dialog that enables the targeted end user to specify which SQL Server should to be used for the current connection, as well as which login credential should be used. The dialog displays a combo box that contains a list of SQL Servers accessed through DSNs. The end user can type a server name in the combo box or click the Browse button next to the combo box; clicking this button displays a list of all SQL Servers that are available on the network.  <b>SQLServerSelectLogin2</b> supersedes this function.

**Table 10 •** Dialog Functions (cont.)

Function	Project Type	Description
<b>SQLServerSelectLogin2</b>	InstallScript, InstallScript MSI	<p>Creates a login dialog that is used by the default script. It lets the targeted end user specify which SQL Server should be used for the current connection, as well as which login credential should be used. The dialog displays a combo box that contains a list of SQL Servers accessed through DSNs. The end user can type a server name in the combo box or click the Browse button next to the Server Name combo box; clicking this button displays a list of all SQL Servers that are available on the network.</p> <p>This function also optionally shows the connection name that is associated with the connection information. In addition, it optionally enables end users to specify which database catalog should be used for the current connection.</p>
<b>SQLServerSelectLoginEx</b>	InstallScript, InstallScript MSI	<p>Creates a login dialog that is used by the default script. It lets the targeted end user specify which SQL Server should be used for the current connection, as well as which login credential should be used. The dialog displays a combo box that contains a list of SQL Servers accessed through DSNs. The end user can type a server name in the combo box or click the Browse button next to the Server Name combo box; clicking this button displays a list of all SQL Servers that are available on the network.</p> <p>This function also shows the connection name that is associated with the connection information.</p> <p><b>SQLServerSelectLogin2</b> supersedes this function.</p>
<b>Welcome</b>	InstallScript, InstallScript MSI	Presents a dialog that displays welcome information.

## Dialog Customization Functions

Use the following functions to create and customize new InstallScript dialogs, and to modify the text, controls, and behavior of InstallScript dialogs. Some of the functions handle custom dialog processing.

Any Windows dialog that you can create can be used in a setup script. The dialogs can have single and multiline edit boxes, single and multiselection list boxes, combo boxes, radio buttons, check boxes, and push buttons as standard controls. For more complex controls, advanced functions such as **CmdGetHwndDlg**, **LOWORD**, and **HIWORD** are provided.

**Table 11 •** Dialog Customization Functions

Function	Description
<b>CmdGetHwndDlg</b>	Retrieves the handle of a dialog.
<b>CtrlClear</b>	Deletes the contents of an edit, static, list box, or combo box control.
<b>CtrlDir</b>	Fills a list box or combo box with either a directory listing or a file listing.
<b>CtrlGetCurSel</b>	Returns the selected item from a list box or combo box.
<b>CtrlGetDlgItem</b>	Retrieves the window handle of a control in a custom dialog.
<b>CtrlGetMLEText</b>	Retrieves the text from a multi-line edit or static field.
<b>CtrlGetMultCurSel</b>	Returns the selected items from a multi-selection list box.
<b>CtrlGetState</b>	Retrieves the state of a radio button, check box, or push button control from a dialog.
<b>CtrlGetSubCommand</b>	Retrieves the operation performed on the control after a <b>WaitOnDialog</b> function call.
<b>CtrlGetText</b>	Retrieves the text from an edit field, a static field, or the edit field of a combo box.
<b>CtrlGetUrlForLinkClicked</b>	Retrieves the URL for a link that an end user clicked.
<b>CtrlPGroups</b>	Retrieves a list of program group names that exist on the target system.
<b>CtrlSelectText</b>	Selects the text displayed in an edit field.
<b>CtrlSetCurSel</b>	Finds and sets the current selection in a list box or combo box.
<b>CtrlSetFont</b>	Specifies a font for a control in the dialog.
<b>CtrlSetList</b>	Places the contents of a list into a list box or combo box.
<b>CtrlSetMLEText</b>	Sets the text in a multi-line edit field.
<b>CtrlSetMultCurSel</b>	Sets the current selection in a multi-selection list box.
<b>CtrlSetState</b>	Sets the current state of a check box, radio button, or push button control.
<b>CtrlSetText</b>	Sets the text in an edit field, a static text field, or the edit field of a combo box.

**Table 11 •** Dialog Customization Functions (cont.)

Function	Description
<b>DefineDialog</b>	Registers a custom dialog with InstallShield.
<b>DialogSetFont</b>	Sets the font for dialogs displayed during the setup.
<b>DialogSetInfo</b>	Changes display elements in the dialogs presented by some built-in dialog functions.
<b>EndCurrentDialog</b>	Closes the currently displayed dialog by calling <b>EndDialog</b> .
<b>EndDialog</b>	Closes a custom dialog.
<b>EzDefineDialog</b>	Registers a custom dialog with InstallShield.
<b>GetCurrentDialogName</b>	Retrieves the name of the currently displayed dialog as it was specified in the call to <b>EzDefineDialog</b> when the dialog was defined.
<b>GetFont</b>	Retrieves the handle of a font.
<b>HIWORD</b>	Retrieves the high-order word from a 32-bit integer.
<b>LOWORD</b>	Retrieves the low-order word from a 32-bit integer.
<b>ReleaseDialog</b>	Frees the memory associated with a dialog.
<b>SdGeneralInit</b>	Provides standard dialog initialization, including setting the enable or disable state of the Next, Back, and Cancel buttons. This function also replaces all %P, %VS, and %VI instances on static controls with control IDs 700 through 724, and 202.
<b>SdInit</b>	Prepares a setup for calls to Sd dialog functions.
<b>SdLoadString</b>	Returns the string value associated with a specified resource ID.
<b>SdMakeName</b>	Creates a section name for a custom dialog. This section name is used in writing to and reading from an .iss file, which is used by InstallShield Silent.
<b>SdProductName</b>	Inserts your product name in certain static fields of the script dialogs.
<b>SdSubstituteProductInfo</b>	Replaces any occurrences of the %P, %VS, and %VI placeholders with the values of the system variables IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and IFX_INSTALLED_DISPLAY_VERSION. You can use this function before calling a function, such as MessageBox, that does not automatically perform this replacement before displaying strings.
<b>SilentReadData</b>	Instructs InstallShield Silent to read the .iss file dialog data for a custom dialog.

**Table 11** • Dialog Customization Functions (cont.)

Function	Description
<b>SilentWriteData</b>	Instructs InstallShield Silent to write to the .iss file dialog data for a custom dialog.
<b>WaitOnDialog</b>	Presents a custom dialog.

## Extensibility Functions

Extensibility functions allow you to call functions in dynamic-link libraries, call Windows APIs, or launch another application or setup script. The UseDLL and UnUseDLL functions allow you to load or unload a DLL into memory and make use of the DLL. The LaunchApp and LaunchAppAndWait functions allow you to launch another Windows or DOS application while the script is still executing.


**Table 12** • Extensibility Functions

Function	Description
<b>CallDLLFx</b>	Calls a function from an external DLL.
<b>Delay</b>	Delays the execution of the setup script.
<b>LaunchApp</b>	Launches another program. <b>LaunchApplication</b> supersedes this function.
<b>LaunchAppAndWait</b>	Launches another program and waits for that program to terminate. <b>LaunchApplication</b> supersedes this function.
<b>LaunchAppAndWaitInitStartupInfo</b>	Initializes the LAAW_STARTUPINFO and LAAW_PARAMETERS system variables to the appropriate default values. <b>LaunchApplicationInit</b> supersedes this function.
<b>LaunchApplication</b>	Uses either the Windows API function <b>CreateProcess</b> or the Windows API function <b>ShellExecuteEx</b> to launch the specified application. After the application is launched, the installation can optionally call <a href="#">WaitForApplication</a> to wait for the application to terminate.
<b>LaunchApplicationInit</b>	Initializes the LAAW_STARTUPINFO and LAAW_PARAMETERS system variables to the appropriate default values. This function is called automatically during installation initialization.
<b>UnUseDLL</b>	Unloads a DLL from memory.
<b>UseDLL</b>	Loads a DLL into memory.
<b>WaitForApplication</b>	Waits for a running application to terminate before returning.

# Feature Functions

The functions below allow you to control file media and to create and process script-created feature sets.

**Table 13 • Feature Functions**

Function	Project Type	Description
<b>FeatureAddCost</b>	InstallScript	Specifies that the feature includes additional installation operations that should be accounted for when updating the progress bar during the installation.
		 <p><b>Note</b> • This function is supported only for file media. Use <b>FeatureGetData</b> or <b>FeatureSetData</b> to set the size of script-created features.</p>
<b>FeatureAddItem</b>	InstallScript, InstallScript MSI	Adds a new feature to a script-created feature set.
<b>FeatureAddUninstallCost</b>	InstallScript	Specifies that the feature includes additional uninstallation operations that must be accounted for when updating the progress bar during uninstallation.
<b>FeatureCompareSizeRequired</b>	InstallScript, InstallScript MSI	Determines if enough free disk space exists for the selected features.
<b>FeatureDialog</b>	InstallScript, InstallScript MSI	Presents a dialog that enables end users to select features and specify a destination location.
<b>FeatureError</b>	InstallScript, InstallScript MSI	Returns additional error information when a feature function fails.
<b>FeatureErrorInfo</b>	InstallScript, InstallScript MSI	Returns additional error information when a feature function fails.
<b>FeatureFileEnum</b>	InstallScript, InstallScript MSI	Builds a list of the files in a component associated with the specified feature.
<b>FeatureFileInfo</b>	InstallScript, InstallScript MSI	Retrieves information on file in the file media that is referenced within the function.
<b>FeatureFilterLanguage</b>	InstallScript, InstallScript MSI	Enables and disables filtering based on language.
<b>FeatureFilterOS</b>	InstallScript, InstallScript MSI	Enables and disables filtering based on operating system (OS).

**Table 13 • Feature Functions (cont.)**


Function	Project Type	Description
<b>FeatureGetCost</b>	InstallScript, InstallScript MSI	Retrieves the total space, in kilobytes (KB), required on the target drive for the specified feature.  <b>FeatureGetCostEx</b> supersedes this function.
<b>FeatureGetCostEx</b>	InstallScript, InstallScript MSI	Retrieves the cost of the specified feature (in bytes) using the <b>nvCostHigh</b> and <b>nvCostLow</b> parameters.
<b>FeatureGetData</b>	InstallScript, InstallScript MSI	Retrieves information about a feature.
<b>FeatureGetItemSize</b>	InstallScript, InstallScript MSI	Determines the size of a specified feature.
<b>FeatureGetTotalCost</b>	InstallScript, InstallScript MSI	Determines the total space required for the feature installations and uninstallations that have been specified.  <b>FeatureGetCostEx</b> supersedes this function.
<b>FeatureInitialize</b>	InstallScript	Supported only for compatibility with scripts created in earlier versions of InstallShield. It is recommended that you avoid using multiple file media libraries in InstallShield.
<b>FeatureIsItemSelected</b>	InstallScript, InstallScript MSI	Determines if the specified feature has been selected by the end user.
<b>FeatureListItems</b>	InstallScript, InstallScript MSI	Creates a list of the features in a file media library or a script-created feature set.
<b>FeatureLoadTarget</b>	InstallScript	Called automatically during the initialization of any installation for which a valid log file exists.
<b>FeatureMoveData</b>	InstallScript	Transfers and decompresses files associated with selected features in the file media.
<b>FeaturePatch</b>	InstallScript	Called only in installations that use differential media. This function causes the next call to <b>FeatureTransferData</b> or <b>FeatureMoveData</b> to reinstall all features that are already installed when <b>FeatureTransferData</b> is called, including all of the maintenance/uninstallation feature's files except for <b>Data1.hdr</b> , <b>Data1.cab</b> , and <b>Layout.bin</b> .
<b>FeatureReinstall</b>	InstallScript, InstallScript MSI	Configures the setup so that the next call to <b>FeatureTransferData</b> performs the file transfer that was specified the last time the setup was run.
<b>FeatureRemoveAll</b>	InstallScript, InstallScript MSI	Configures the setup so that the next call to <b>FeatureTransferData</b> uninstalls the setup.

**Table 13 • Feature Functions (cont.)**

Function	Project Type	Description
<b>FeatureRemoveAllInLogOnly</b>	InstallScript	Called during an update installation to force the removal of all features that are not in the current media but were installed previously, as recorded in the setup log file.
<b>FeatureRemoveAllInMedia</b>	InstallScript	Used during a maintenance installation to force the removal of all features that are in the current media and were installed previously. This function is generally called when the user selects the Remove option on the <b>SdWelcomeMaint</b> dialog.
<b>FeatureRemoveAllInMediaAndLog</b>	InstallScript	Called during an update installation to force the removal of all features that were installed previously—both those that are in the current media, and those that are not in the current media but are recorded in the setup log file.
<b>FeatureSaveTarget</b>	InstallScript, InstallScript MSI	Retrieves the current values of all text substitutions used by the installation project and stores them in the installation log file.
<b>FeatureSelectItem</b>	InstallScript, InstallScript MSI	Selects or deselects features.
<b>FeatureSelectNew</b>	InstallScript	Sets the selection status of all new features to either selected or unselected.
<b>FeatureSetData</b>	InstallScript, InstallScript MSI	Sets properties and data for the specified feature.
<b>FeatureSetTarget</b>	InstallScript, InstallScript MSI	Specifies a user-defined variable for a file media library.
<b>FeatureSetupTypeEnum</b>	InstallScript, InstallScript MSI	Enumerates the setup types associated with the specified file media library.
<b>FeatureSetupTypeGetData</b>	InstallScript, InstallScript MSI	Retrieves data associated with a specified setup type that has been created in the InstallShield interface.
<b>FeatureSetupTypeSet</b>	InstallScript, InstallScript MSI	Selects all features associated with the specified setup type.
<b>FeatureSpendCost</b>	InstallScript	Updates the progress bar for a certain amount of cost that has been spent by an event external to the installation.
<b>FeatureSpendUninstallCost</b>	InstallScript	Updates the progress bar for a certain amount of uninstallation cost that has been spent by an event external to the installation.



**Table 13** • Feature Functions (cont.)

Function	Project Type	Description
<b>FeatureStandardSetupTypeSet</b>	InstallScript, InstallScript MSI	Sets the current setup type to the standard setup type specified by nSetupType.
<b>FeatureTotalSize</b>	InstallScript, InstallScript MSI	Calculates the total size, in bytes, of selected features and subfeatures.
<b>FeatureTransferData</b>	InstallScript, InstallScript MSI (if the InstallScript user interface (UI) style is the traditional style, which uses the InstallScript engine as an external UI handler)	 <p><b>Important</b> • This function does not apply to InstallScript MSI projects in which the InstallScript UI style is the new style (which uses the InstallScript engine as an embedded UI handler). To learn more, see <i>Using the InstallScript Engine as an External vs. Embedded UI Handler for InstallScript MSI Installations</i>.</p> <p>In an event-based script, installs or uninstalls features appropriately based on their selection state and whether they are currently installed.</p>
<b>FeatureUpdate</b>	InstallScript	Configures the installation so that the next call to <b>FeatureTransferData</b> or <b>FeatureMoveData</b> reinstalls all features that are already installed.
<b>FeatureValidate</b>	InstallScript	Validates the password of the file media library or of a specified feature.
<b>SdSetupType</b>	InstallScript, InstallScript MSI	Displays a dialog that enables end users to select one of the three standard setup types: Typical, Compact, or Custom.
<b>SdSetupType2</b>	InstallScript, InstallScript MSI	Displays a dialog that enables the end user to select one of the two standard setup types: Typical or Custom.
<b>SdSetupTypeEx</b>	InstallScript MSI	Displays a dialog that enables the end user to select the setup type when you specify setup types beyond Complete and Custom.

## Script-Created Feature Set vs. File Media Library

You can create features at run time by calling the [FeatureAddItem](#) function in your setup script. These script-created features reside only in memory and have no direct connection to a file media library. Unlike the information stored in a file media library, script-created features are not—and cannot be—directly associated with components or a setup type.

However, script-created features provide a convenient way of displaying feature-like options for the end user. After the end user makes feature selections in feature dialogs, you can test the script-created features' selection status and use the result as the basis for carrying out some action. For example, you might want to install files with XCopyFile or VerUpdateFile, select features in the file media library, or create or edit a file.

## Building a Script-Created Feature

To build a new script-created feature, call the [FeatureAddItem](#) function. Then, set and access properties for the script-created feature using the InstallScript feature functions much as you would for features in a file media library (exceptions noted below).

## Referring to Script-Created Feature Sets

Script-created features are often referred to collectively as a "script-created feature set." The reason is that they are often handled just like file media library features by feature functions. You must treat all features as a set when you pass their media name to the feature functions.



---

**Note** • You create the media name in the first parameter of *FeatureAddItem*. Use this value when you create features and subfeatures as part of the same "script-created feature set" and when you refer to existing script-created features in your script.

## Using Feature Functions

Since the two types of features are so different, you call some of the feature functions differently depending on whether you are working with a script-created feature or a feature in a file media library.

### File Media Library and Script-Created Feature Set Functions

These functions can be used on both a feature in a file media library or on a script-created feature:

- [FeatureGetData](#)
- [FeatureSetData](#)

### File Media Library Functions

These functions work only with features in a file media library, but not with script-created features:

- [FeatureCompareSizeRequired](#)
- [FeatureFilterLanguage](#)
- [FeatureFilterOS](#)
- [FeatureSetTarget](#)
- [FeatureSetupTypeEnum](#)
- [FeatureSetupTypeGetData](#)
- [FeatureSetupTypeSet](#)
- [FeatureTransferData](#)

## Script-Created Feature Set Function

The `FeatureAddItem` function works exclusively with script-created feature sets.

## File Media Library

The file media library contains your product's files and all the information that you entered in the InstallShield interface about your installation's component, feature, and setup type settings.

The file media library is created when you create a release for your project. It is defined by **Data1.hdr**, the InstallScript header file. It also has a media name, the default value of which is contained in the `MEDIA` system variable.

You can set and access information in your file media library using the InstallScript feature functions.



**Note** • Some InstallScript feature functions are specifically reserved for use with script-created features.

## File and Folder Functions

File and folder functions provide a comprehensive way to work with text files, binary files, and folders. Many of the functions use the variables `TARGETDIR` (in InstallScript projects), `INSTALLDIR` (in InstallScript MSI and Basic MSI projects) and `SRCDIR` as the paths and accept only file names as parameters. Wild-card characters are also accepted where appropriate.

**Table 14** • File and Folder Functions

Function	Description
<b>ChangeDirectory</b>	Makes the specified directory the current directory.
<b>CloseFile</b>	Closes an open file.
<b>CopyFile</b>	Copies a file from one folder to another.
<b>CreateDir</b>	Creates a new folder.
<b>CreateFile</b>	Creates a file with the specified file name.
<b>DeleteDir</b>	Deletes a folder.
<b>DeleteFile</b>	Deletes a file.
<b>ExistsDir</b>	Determines whether or not the specified directory exists.
<b>ExistsDisk</b>	Determines whether or not the specified disk exists.
<b>FileCompare</b>	Compares one file with another.
<b>FileDeleteLine</b>	Deletes a line in a text file.

**Table 14 • File and Folder Functions (cont.)**

Function	Description
<b>FileGrep</b>	Searches a text file for specified text.
<b>FileInsertLine</b>	Inserts a line in a text file.
<b>FindAllDirs</b>	Finds all subfolders under the specified folder.
<b>FindAllFiles</b>	Finds all files in the specified folder and its subfolders that match a file specification.
<b>FindFile</b>	Finds the first file in the specified folder that matches a file specification.
<b>GetFileInfo</b>	Retrieves a file's attributes, date, time, and size.
<b>GetLine</b>	Retrieves a line of text from an open file.
<b>GetTempFileNameIS</b>	Calls the Windows API <b>GetTempFileName</b> to create a temporary file and perform related actions. Note that unlike <b>GetTempFileName</b> , <b>GetTempFileNameIS</b> creates the folder specified by szPathName if it does not already exist.
<b>Is</b>	Provides file and path checking services, searches for a math coprocessor, tests for administrator privileges, determines whether a particular version of the .NET Framework or a language pack is present on the target system, determines whether Microsoft Windows is running from a shared copy on a network, and more.
<b>OpenFile</b>	Opens an existing file.
<b>OpenFileMode</b>	Sets the mode in which files will be opened with the OpenFile function.
<b>ReadBytes</b>	Reads the specified number of bytes from a binary file.
<b>RenameFile</b>	Renames a file.
<b>SeekBytes</b>	Positions the file pointer in a binary file.
<b>SetFileInfo</b>	Sets the attributes, date, and time of a file.
<b>SetObjectPermissions</b>	Sets permissions for a file, a folder, or a registry key. The file, folder, or registry key can be installed as part of your installation, or it can be already present on the target system.
<b>WriteBytes</b>	Writes a specified number of bytes to a binary file at the current file pointer location.
<b>WriteLine</b>	Writes a string to a text file.
<b>XCopyFile</b>	Copies one or more files from a source folder to a target folder, including subfolders if specified.

## Related Function



**Table 15** • Related Function

Function	Description
<a href="#">SelectDir</a>	Presents a dialog that enables end users to select a folder. <b>SelectDir</b> creates the folder if it does not exist.

## FlexNet Connect Functions

A number of built-in InstallScript functions support FlexNet Connect, which was previously called Update Service.

**Table 16** • FlexNet Connect Functions

Function	Project Type	Description
<a href="#">GetUpdateStatus</a>	Basic MSI, InstallScript MSI, InstallScript	This function is obsolete.
<a href="#">GetUpdateStatusReboot</a>	Basic MSI, InstallScript MSI, InstallScript	This function is obsolete.
<a href="#">SdFinishUpdate</a>	Basic MSI, InstallScript MSI, InstallScript	<p>Displays a dialog that indicates that the installation is complete. The dialog includes the option to check for application updates.</p> <p><a href="#">SdFinishUpdateEx</a> supersedes this function.</p> <p></p> <p><b>Note • SdFinishUpdate</b> does not check for updates; to check for updates, add FlexNet Connect API calls in your InstallScript code. For more information, see the FlexNet Connect SDK documentation.</p>
<a href="#">SdFinishUpdateEx</a>	Basic MSI, InstallScript MSI, InstallScript	<p>Displays a dialog that indicates that the installation is complete. The dialog includes the option to check for application updates.</p> <p></p> <p><b>Note • SdFinishUpdateEx</b> does not check for updates; to check for updates, add FlexNet Connect API calls in your InstallScript code. For more information, see the FlexNet Connect SDK documentation.</p>

**Table 16 •** FlexNet Connect Functions (cont.)

Function	Project Type	Description
<b>SdFinishUpdateReboot</b>	Basic MSI, InstallScript MSI, InstallScript	<p>Displays a dialog that indicates that the installation is complete. The dialog gives the end user the option to restart the system and to check for application updates.</p>  <p><b>Note • SdFinishUpdateReboot</b> does not check for updates; to check for updates, add FlexNet Connect API calls in your InstallScript code. For more information, see the FlexNet Connect SDK documentation.</p>
<b>SetUpdateStatus</b>	Basic MSI, InstallScript MSI, InstallScript	This function is obsolete.
<b>SetUpdateStatusReboot</b>	Basic MSI, InstallScript MSI, InstallScript	This function is obsolete.
<b>UpdateServiceCheckForUpdates</b>	InstallScript	<p>This function is obsolete.</p> <p>For information on adding FlexNet Connect support to an InstallScript project, consult the <a href="#">Knowledge Base</a>.</p>
<b>UpdateServiceCreateShortcut</b>	InstallScript	<p>This function is obsolete.</p> <p>For information on adding FlexNet Connect support to an InstallScript project, consult the <a href="#">Knowledge Base</a>.</p>
<b>UpdateServiceEnableUpdateManagerInstall</b>	InstallScript	<p>This function is obsolete.</p> <p>For information on adding FlexNet Connect support to an InstallScript project, consult the <a href="#">Knowledge Base</a>.</p>
<b>UpdateServiceGetAgentTarget</b>	InstallScript	<p>This function is obsolete.</p> <p>For information on adding FlexNet Connect support to an InstallScript project, consult the <a href="#">Knowledge Base</a>.</p>

**Table 16** • FlexNet Connect Functions (cont.)

Function	Project Type	Description
<b>UpdateServiceOnEnabledStateChange</b>	InstallScript	This function is obsolete.  For information on adding FlexNet Connect support to an InstallScript project, consult the <a href="#">Knowledge Base</a> .
<b>UpdateServiceRegisterProduct</b>	InstallScript	This function is obsolete.  For information on adding FlexNet Connect support to an InstallScript project, consult the <a href="#">Knowledge Base</a> .
<b>UpdateServiceRegisterProductEx</b>	InstallScript	This function is obsolete.  For information on adding FlexNet Connect support to an InstallScript project, consult the <a href="#">Knowledge Base</a> .
<b>UpdateServiceSetHost</b>	InstallScript	This function is obsolete.  For information on adding FlexNet Connect support to an InstallScript project, consult the <a href="#">Knowledge Base</a> .
<b>UpdateServiceSetLanguage</b>	InstallScript	This function is obsolete.  For information on adding FlexNet Connect support to an InstallScript project, consult the <a href="#">Knowledge Base</a> .

## Information Functions

The following information functions provide data about resources that are available in the operating environment: disk space, memory, and operating mode.

**Table 17** • Information Functions

Function	Description
<b>GetDiskInfo</b>	Gets information about a specified disk drive.
<b>GetDiskSpace</b>	Returns the number of bytes available (unused) on the specified disk (up to 2 gigabytes).
<b>GetDiskSpaceEx</b>	Returns the amount of free space on a disk in bytes, kilobytes, megabytes, or gigabytes.

**Table 17** • Information Functions (cont.)

Function	Description
<b>GetEnvVar</b>	Returns the current value of an environment variable.
<b>GetExtendedErrInfo</b>	Returns the error information that was set by <b>SetExtendedErrInfo</b> .
<b>GetExtents</b>	Returns the dimensions of the screen.
<b>GetMemFree</b>	This function is obsolete and should not be used.
<b>GetSystemInfo</b>	Retrieves system information.
<b>GetTrueTypeFontFileInfo</b>	Returns information about a particular TrueType font file.
<b>GetValidDrivesList</b>	Returns a listing of all available drives on the target system.
<b>GetWindowHandle</b>	Returns the handle of the main installation window.
<b>Is</b>	Provides file and path checking services, searches for a math coprocessor, tests for administrator privileges, determines whether a particular version of the .NET Framework or a language pack is present on the target system, determines whether Microsoft Windows is running from a shared copy on a network, and more.
<b>SetExtendedErrInfo</b>	Sets error information, which can be retrieved by <b>GetExtendedErrInfo</b> .
<b>SetInstallationInfo</b>	Sets the values of the system variables IFX_COMPANY_NAME, IFX_PRODUCT_NAME, IFX_PRODUCT_VERSION, and IFX_PRODUCT_KEY.

## Initialization File Functions

Initialization file functions obtain information from and copy information to the initialization and profile files. An initialization file is a special ASCII file that contains key name-value pairs. The key name-value pairs represent run-time options for applications. You can also access and update private initialization file and system initialization files. The following list briefly describes each initialization file function.

**Table 18** • Initialization File Functions

Function	Description
<b>AddProfString</b>	Adds a non-unique key to a section of the .ini file.
<b>GetProfInt</b>	Returns an integer from an .ini file.
<b>GetProfString</b>	Returns a string from an .ini file.
<b>GetProfStringList</b>	Retrieves lists of key names and string values from an .ini file.
<b>ReplaceProfString</b>	Replaces a string in a profile (.ini) file.



**Table 18** • Initialization File Functions (cont.)

Function	Description
<b>WriteProfInt</b>	Writes a string with an integer value to an .ini file.
<b>WriteProfString</b>	Writes a string to an .ini file.

## Related Function

**Table 19** • Related Function

Function	Description
<b>SdShowFileMods</b>	Creates a dialog displaying proposed file changes and offering options on how to proceed.

# List Processing Functions

Lists are used to store groups of related information. In InstallScript, there are two types of lists: string lists and number lists. Two sets of functions are provided to work with lists, one for each list type. List functions that end with "Item" operate with number lists. List functions that end with "String" operate with string lists. You cannot use number list functions on string lists and vice versa. Below are the functions to implement lists in a setup script.

**Table 20** • List Processing Functions

Function	Description
<b>ListAddItem</b>	Adds an item to a list.
<b>ListAddString</b>	Adds a string to a list.
<b>ListCount</b>	Returns the number of string or numeric elements in a specified list.
<b>ListCreate</b>	Creates a new string or number list.
<b>ListCurrentItem</b>	Returns the current item in a list.
<b>ListCurrentString</b>	Returns the current string in a list.
<b>ListDeleteItem</b>	Deletes the current item in a list.
<b>ListDeleteString</b>	Deletes the current string in a list.
<b>ListDestroy</b>	Destroys a list.
<b>ListFindItem</b>	Makes the specified item the current item in a numeric list.
<b>ListFindKeyValueString</b>	Searches a string or number list for a specified value. It returns a value from an additional list that corresponds with the position of the found string in the first list.

**Table 20** • List Processing Functions (cont.)

Function	Description
<b>ListFindString</b>	Makes the specified item the current item in a string list.
<b>ListGetFirstItem</b>	Retrieves the first element from a number list.
<b>ListGetFirstString</b>	Retrieves the first string from a string list.
<b>ListGetNextItem</b>	Retrieves the element after the current element from a number list.
<b>ListGetNextString</b>	Retrieves the element after the current element from a string list.
<b>ListReadFromFile</b>	Reads a text file into a list.
<b>ListSetCurrentItem</b>	Sets the current element of a number list.
<b>ListSetCurrentString</b>	Sets the current element of a string list.
<b>ListSetIndex</b>	Uses an index to set the current element of a list.
<b>ListValid</b>	Indicates whether the specified list is valid.
<b>ListValidType</b>	Indicates whether the specified list is valid and is of the specified type.
<b>ListWriteToFile</b>	Writes a string list to a file.
<b>ListWriteToFileEx</b>	Writes or appends a string list to a text file as Unicode or ANSI, depending on the constants that you provide for the <code>nOptions</code> parameter.

## Log File Functions



**Project** • This information applies to *InstallScript* projects.

Log file functions obtain information from and copy information to the custom logging section of the log file. Custom log file entries do not affect maintenance or uninstallation of the application unless you add code to the script to read custom values and perform actions based on those values. The log file functions cannot read data from or write data to the maintenance/uninstallation section of the log file (that is, the section where the setup automatically writes data, such as the files that are installed and the registry entries that are created, and from which it automatically reads data during maintenance or uninstallation). The following list briefly describes each log file function.

**Table 21** • Log File Functions

Function	Description
<b>LogReadCustomNumber</b>	Reads numeric data from the log file's custom logging section.

**Table 21** • Log File Functions (cont.)

Function	Description
<a href="#">LogReadCustomString</a>	Reads string data from the log file's custom logging section.
<a href="#">LogWriteCustomNumber</a>	Writes numeric data to the log file's custom logging section.
<a href="#">LogWriteCustomString</a>	Writes string data to the log file's custom logging section.

## Long File Name Functions

The following functions create long file names from short file names, convert short file names to long file names, and place double quotation marks around long file names so that operating systems that handle long file names can recognize them.

**Table 22** • Log File Name Functions

Function	Description
<a href="#">LongPathFromShortPath</a>	Creates a long file name from a short file name.
<a href="#">LongPathToQuote</a>	Inserts or removes double quotation marks around a long file name.
<a href="#">LongPathToShortPath</a>	Creates a short file name from a long file name.

## Miscellaneous Functions

The following functions serve various purposes, such as low-level hardware interface, feature creation and manipulation, and user output.

**Table 23** • Miscellaneous Functions

Function	Description
<a href="#">Do</a>	Executes the currently defined EXIT and HELP handlers.
<a href="#">DoInstall</a>	Launches another InstallShield installation.
<a href="#">FormatMessage</a>	Returns a string error message for a large negative error code.
<a href="#">Handler</a>	This function is obsolete. Use <a href="#">HandlerEx</a> instead.
<a href="#">HandlerEx</a>	Specifies a label to branch to in response to exit and help events.
<a href="#">ISCompareServicePack</a>	Compares the Service Pack number installed on the target OS to a specified Service Pack number.
<a href="#">IsEmpty</a>	Checks whether a variable of type VARIANT has been initialized.

**Table 23 • Miscellaneous Functions (cont.)**

Function	Description
<b>MessageBeep</b>	Produces a standard warning beep.
<b>Resize</b>	Resizes an InstallScript array.
<b>SendMessage</b>	Sends a Windows message to another window or application.
<b>SetObjectPermissions</b>	Sets permissions for a file, a folder, or a registry key.
<b>SizeOf</b>	Returns the size of an InstallScript array.
<b>Sprintf</b>	Returns a formatted string composed of one or more character, numeric, or string values.
<b>SprintfMsiLog</b>	Writes a message directly to the Windows Installer log file.
<b>StreamFileFromBinary</b>	Streams a binary key with a file.
<b>System</b>	Reboots the computer.
<b>VarInit</b>	Initializes or reinitializes internal lists used by the <b>VarSave</b> and <b>VarRestore</b> functions. Calling this function effectively clears any information stored by previous <b>VarSave</b> calls and not yet used by subsequent <b>VarRestore</b> functions.
<b>VarRestore</b>	Restores the values of the system variables SRCDIR, TARGETDIR (in InstallScript projects), and INSTALLDIR (in Basic MSI and InstallScript projects) that were saved by the last call to <b>VarSave</b> .
<b>VarSave</b>	Saves the current value of the system variables SRCDIR, TARGETDIR (in InstallScript projects), and INSTALLDIR (in Basic MSI and InstallScript projects).

## Object Functions

The object functions are available to assist you with initializing objects, as well as obtaining and setting status information for objects.

**Table 24 • Object Functions**

Function	Description
<b>CoCreateObject</b>	Initializes a COM object and returns a reference that can be assigned to a variable of type OBJECT by using the set keyword.
<b>CoCreateObjectDotNet</b>	<p>The <b>CoCreateObjectDotNet</b> function has been deprecated. Calling this function is the same as calling the <b>DotNetCoCreateObject</b> function with a null string ("") for the szAppDomain parameter.</p> <p>For more information, see <a href="#">DotNetCoCreateObject</a>.</p>

**Table 24 • Object Functions (cont.)**

Function	Description
<b>CoGetObject</b>	Returns a reference to the specified COM object (as Visual Basic's <code>GetObject</code> function does); that reference can be assigned to a variable of type <code>OBJECT</code> by using the <code>set</code> keyword.
<b>DotNetCoCreateObject</b>	Calls functions in .NET assemblies without the assembly being registered for COM interoperability. This function, unlike the <b>CoCreateObjectDotNet</b> function, lets you specify the .NET application domain in which the .NET assemblies should be loaded and run.
<b>DotNetUnloadAppDomain</b>	Unloads the specified .NET application domain and releases any assemblies that are currently loaded into the specified application domain.
<b>GetObject</b>	Initializes an object and returns a reference that can be assigned to a variable of type <code>OBJECT</code> by using the <code>set</code> keyword.
<b>GetObjectByIndex</b>	Finds the installation's or object's subobject that is specified by <code>nIndex</code> and returns a reference that can be assigned to a variable of type <code>OBJECT</code> by using the <code>set</code> keyword.
<b>GetObjectCount</b>	Returns the number of subobjects contained by the object or installation.
<b>GetStatus</b>	Retrieves the current status of the object; that is, the current value of <code>Status.Number</code> .
<b>IsObject</b>	Checks whether a variable of type <code>OBJECT</code> has been assigned a reference to a valid object, using the <code>CreateObject</code> or <code>GetObject</code> functions.
<b>SetStatus</b>	Called in an object script to set the object's <code>Status.Number</code> and <code>Status.Description</code> properties.
<b>SetStatusEx</b>	Called in an object script to set the object's status properties.

## Path Buffer Functions

The path buffer functions are available to assist you in working with strings that contain search paths. The path string functions work on a unique temporary string variable known as the path buffer. The path buffer is defined internally within InstallShield—all the path string functions act on the contents of the path buffer.

Path functions assist you in manipulating and building path strings. After you have created a path string, you can save it to the appropriate file.

**Table 25 • Path Buffer Functions**

Function	Description
<b>PathAdd</b>	Adds a path to the search path in the path buffer.
<b>PathDelete</b>	Deletes a directory from the path buffer.

**Table 25 • Path Buffer Functions (cont.)**

Function	Description
<b>PathFind</b>	Finds a specific path in the path buffer or any path that includes a specified name.
<b>PathGet</b>	Retrieves the current value of the path buffer.
<b>PathMove</b>	Rearranges the path buffer.
<b>PathSet</b>	Assigns a value to the path buffer.

## Registry Functions

The following functions allow you to access the registry, read, create and delete registry keys, and establish registry-related parameters for uninstallation.

**Table 26 • Registry Functions**

Function	Description
<b>CreateInstallationInfo</b>	Creates an application information key and a per application paths key for the program you are installing.
<b>CreateRegistrySet</b>	Creates one or all of the sets of registry entries specified in the Resources pane's Registry Entries folder.
<b>DeinstallSetReference</b>	This function is obsolete.  If you want to check whether a file is locked during uninstallation, write script code that calls the <a href="#">Is</a> function with the FILE_LOCKED constant for the nlsFlag parameter and that responds as needed.
<b>DeinstallStart</b>	This function is obsolete.
<b>InstallationInfo</b>	This function is obsolete. Use the <a href="#">CreateInstallationInfo</a> function instead.
<b>MaintenanceStart</b>	Enables uninstallation functionality by creating the <PRODUCT_GUID> registry keys.
<b>RegDBConnectRegistry</b>	Opens a connection to a remote registry.
<b>RegDBCopKeyValues</b>	Copies the registry keys and values under the key specified by szSourceKey to the key specified by szTargetKey.
<b>RegDBCopValues</b>	Copies the registry values under the key specified by szSourceKey to the key specified by szTargetKey.

**Table 26 • Registry Functions (cont.)**

Function	Description
<b>RegDBCreateKeyEx</b>	Creates a key in the registry. Also enables you to associate a class object with a registry key (advanced users only).
<b>RegDBDeleteItem</b>	Deletes values under the per application paths key or the application uninstallation key, depending on the value of <code>nltem</code> .
<b>RegDBDeleteKey</b>	Deletes the specified key from the registry.
<b>RegDBDeleteValue</b>	Deletes a value from a specified registry key.
<b>RegDBDisconnectRegistry</b>	Closes the connection to a remote registry.
<b>RegDBGetAppInfo</b>	Retrieves a value from under the application information key.
<b>RegDBGetDefaultRoot</b>	Returns the root key that is used by the general registry-related functions.
<b>RegDBGetItem</b>	Retrieves values under the per-application paths key or the application uninstallation key.
<b>RegDBGetKeyValueEx</b>	Retrieves a value from under a key in the registry.
<b>RegDBGetUninstCmdLine</b>	Gets the registered command line for the uninstallation that is specified by <code>szUninstallKey</code> and returns the command line in <code>svUninstCmdLine</code> .
<b>RegDBKeyExist</b>	Checks that a registry key exists.
<b>RegDBQueryKey</b>	Queries a key for its subkeys and value names.
<b>RegDBQueryKeyCount</b>	Returns the number of subkeys or values under a specified key.
<b>RegDBQueryStringMultiStringCount</b>	Returns the number of strings contained in the multistring value specified by a certain value under a specified key.
<b>RegDBSetAppInfo</b>	Sets a value under the application information key.
<b>RegDBSetDefaultRoot</b>	Sets the root key.
<b>RegDBSetItem</b>	Assign values under the per application paths key or the application uninstallation key.
<b>RegDBSetKeyValueEx</b>	Sets registry entries.
<b>SetInstallationInfo</b>	Specifies company and product information for use by <code>CreateInstallationInfo</code> .

**Table 26 • Registry Functions (cont.)**

Function	Description
<b>SetObjectPermissions</b>	Sets permissions for a file, a folder, or a registry key.

## Service Functions

The following functions allow you to access the registry, read, create and delete registry keys, and establish registry-related parameters for uninstallation.

**Table 27 • Service Functions**

Function	Description
<b>ServiceAddService</b>	Adds a service to the list of services that are registered on the target system.
<b>ServiceExistsService</b>	Determines whether the specified service is registered.
<b>ServiceGetServiceState</b>	Retrieves the state of the specified service.
<b>ServiceInitParams</b>	Initializes the SERVICE_IS_PARAMS system variable's members to default values. This function is called automatically during setup initialization.
<b>ServiceRemoveService</b>	Removes a service from the target system.
<b>ServiceStartService</b>	Starts a service. If the service is running when the function is called, the installation stops it and then restarts it.
<b>ServiceStopService</b>	Stops a service.

## Shared and Locked File Functions

A shared file is a file, such as a .dll, .vbx, or driver that can be used by more than one application. InstallShield protects shared files from being removed during uninstallation.

Functions using the SHAREDFILE option consider all files to be shared files, and therefore increment registry reference counters for all files involved. InstallShield increments the registry reference counter by one if the file exists in the target directory and it has a reference count greater than 0. If the shared file does not exist in the target directory and it has no reference counter, InstallShield creates the counter and sets it to 1. If the shared file already exists in the target directory but has no reference counter, InstallShield creates the counter and initializes it to 2 as a precaution against accidental removal during uninstallation.

Shared files should not be updated when they are locked. Some InstallShield file transfer functions use the SHAREDFILE option so that .dll and .exe files that are locked during file transfer can be recorded and updated when Windows or the system restarts.



InstallShield considers a file locked when it is in use by an application or the system. Locked files are not necessarily shared files.

The following functions handle shared or locked files:

**Table 28 • Shared and Locked File Functions**

Function	Description
<b>Is</b>	Provides file and path checking services, searches for a math coprocessor, tests for administrator privileges, determines whether a particular version of the .NET Framework or a language pack is present on the target system, determines whether Microsoft Windows is running from a shared copy on a network, and more.
<b>RebootDialog</b>	Presents a dialog with which the end user can choose to restart Windows or reboot the computer.
<b>SdFinishReboot</b>	Presents a dialog stating that the installation is complete and allowing the end user to choose to restart Windows or reboot the computer.
<b>SetObjectPermissions</b>	Sets permissions for a file, a folder, or a registry key.
<b>VerUpdateFile</b>	Updates files using version resource information. Allows updating of locked .dll and .exe files and incrementing of registry reference counters for all files involved.
<b>XCopyFile</b>	Copies files and subdirectories from the source directory to the target directory. Combines shared file and locked file handling by causing XCopyFile to treat all files as shared, and to record locked .dll and .exe files for update when Windows or the system restarts.

## Shell Functions

Shell functions create program folders, delete existing program folders, and add items to existing program folders. At the end of the setup, add the application to the appropriate program folder to allow the user to access your software immediately. The following functions also support various icon options.

**Table 29 • Shell Functions**

Function	Description
<b>AddFolderIcon</b>	Adds a shortcut or program folder to locations such as the Start menu, the Programs menu, or the desktop.  <b>CreateShortcut</b> supersedes this function.
<b>CreateProgramFolder</b>	Creates a program folder.  <b>CreateShortcutFolder</b> supersedes this function.

**Table 29** • Shell Functions (cont.)

Function	Description
<b>CreateShortcut</b>	<p>Adds a shortcut or program folder to locations such as the Start menu, the Programs menu, or the desktop.</p> <p>Optionally sets Windows Shell properties for the shortcut to configure behavior such as disabling the ability to pin the shortcut to the Start menu.</p>
<b>CreateShortcutFolder</b>	Creates a program folder.
<b>DeleteFolderIcon</b>	<p>Removes a shortcut from a folder.</p> <p><b>DeleteShortcut</b> supersedes this function.</p>
<b>DeleteProgramFolder</b>	<p>Removes a shortcut folder (that is, a subfolder of the Start menu's Programs folder) and its contents, including all shortcuts and all of the shortcut folder's subfolders and their contents.</p> <p><b>DeleteShortcutFolder</b> supersedes this function.</p>
<b>DeleteShortcut</b>	Removes a shortcut from a folder.
<b>DeleteShortcutFolder</b>	Removes a shortcut folder (that is, a subfolder of the Start menu's Programs folder) and its contents, including all shortcuts and all of the shortcut folder's subfolders and their contents.
<b>GetFolderNameList</b>	Retrieves all subfolder names and shortcuts in the specified folder.
<b>GetShortcutInfo</b>	Returns information about a specified shortcut or subfolder.
<b>ProgDefGroupType</b>	Sets the value of the ALLUSERS system variable.
<b>QueryProgItem</b>	<p>Returns information about a specified shortcut or subfolder.</p> <p><b>GetShortcutInfo</b> supersedes this function.</p>
<b>QueryShellMgr</b>	Returns the name of the current shell manager.
<b>ReplaceFolderIcon</b>	<p>Replaces a shortcut in a specified folder.</p> <p><b>ReplaceShortcut</b> supersedes this function.</p>
<b>ReplaceShortcut</b>	Replaces a shortcut in a specified folder.
<b>SelectFolder</b>	Presents a dialog that enables end users to select a folder from a list of program folders.
<b>SetShortcutProperty</b>	Sets one or more shortcut properties that you want the Windows Shell to set at installation run time.
<b>ShowProgramFolder</b>	Displays the specified program folder.

# Special Registry-Related Functions

The special registry-related functions are designed to make it easier for script writers to set up the minimum required registry keys and values. The special registry-related functions work only with the per application paths key, the application uninstallation key, or the application information key, as shown below. Refer to the individual function descriptions for more details.

## Per Application Paths Key

**<root key>\Software\Microsoft\Windows\CurrentVersion\App Paths\<per application paths key>**

This key is referred to as the per application paths key, or App Paths key. The per application paths key stores path information enabling Windows to find your application's executable files. The root key is HKEY\_CURRENT\_USER if the [ALLUSERS](#) system variable is FALSE or you have called ProgDefGroupType(PERSONAL), or HKEY\_LOCAL\_MACHINE otherwise.

**Table 30 • Per Application Paths Key**

Function	Description
<a href="#">CreateInstallationInfo</a>	Uses the name of the application executable file to prepare for the creation of the per application paths key. The key is not created until RegDBSetItem is called (see below).  If you use an event-based script, the CreateInstallationInfo function is called by the default OnMoveData event handler code.
<a href="#">RegDBDeleteItem</a>	Deletes the per application paths key and the value of [Path] or of [DefaultPath] under that key.
<a href="#">RegDBGetItem</a>	Retrieves the value of [Path] or of [DefaultPath] under the per applications path key.
<a href="#">RegDBSetItem</a>	Results in the creation of the per application paths key, and sets the value of [Path] or of [DefaultPath] under that key.

## Application Uninstallation Key

**<rootkey>\Software\Microsoft\Windows\CurrentVersion\Uninstall\<INSTANCE\_GUID>**

This key is referred to as the application uninstallation key. The application uninstallation key stores information enabling uninstallation functionality. The root key is HKEY\_CURRENT\_USER if the [ALLUSERS](#) system variable is FALSE or you have called ProgDefGroupType(PERSONAL), or HKEY\_LOCAL\_MACHINE otherwise.

**Table 31 • Application Uninstallation Key**

Function	Description
<b>MaintenanceStart</b>	Creates the application uninstallation key and sets the [UninstallString], [DisplayName] (the name displayed in Add or Remove Programs), and [LogFile] values under that key.  In an event-based script, MaintenanceStart is called in the default OnMoveData event handler code.
<b>RegDBDeleteItem</b>	Deletes the value of [DisplayName] (the name displayed in Add or Remove Programs) under the application uninstallation key.
<b>RegDBGetItem</b>	Retrieves the value of [DisplayName] under the application uninstallation key.
<b>RegDBSetItem</b>	Sets the value of [DisplayName] (the name displayed in Add or Remove Programs) under the application uninstallation key.  This value is also set by MaintenanceStart (which, if you use an event-based script, is called in the default OnMoveData event handler code).

## Application Information Key

**<root key>\Software\<company key>\<product key>\<version key>**

This key is referred to as the application information key. Your installation should create an application information key for each application it installs. The application information key stores information about the application. The root key is HKEY\_CURRENT\_USER if the [ALLUSERS](#) system variable is FALSE or you have called ProgDefGroupType(PERSONAL), or HKEY\_LOCAL\_MACHINE otherwise.

**Table 32 • Application Information Key**

Function	Description
<b>CreateInstallationInfo</b>	Uses the company name, product name, and product version number to create the application information key. No values are set under the key until you call the RegDBSetAppInfo function (see below).  If you use an event-based script, the CreateInstallationInfo function is called by the default OnMoveData event handler code.
<b>RegDBGetAppInfo</b>	Retrieves a value from under the application information key.
<b>RegDBSetAppInfo</b>	Sets a value under the application information key.

# SQL Functions

The SQL functions enable you to perform tasks such as connect to a catalog, create SQL-related dialogs, and obtain SQL run-time errors.



**Tip** • For information on SQL support and SQL-related InstallScript functions, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

**Table 33** • SQL Functions

Function	Project Type	Description
<b>SQLBrowse</b>	InstallScript, InstallScript MSI	Creates a dialog that enables the end user to bring up a list of all SQL Servers available on the network.  <b>SQLBrowse2</b> supersedes this function.
<b>SQLBrowse2</b>	InstallScript, InstallScript MSI	Creates a dialog that lets an end user display a list of all database servers that are available on the network for the database technologies specified for a connection.
<b>SQLDatabaseBrowse</b>	InstallScript, InstallScript MSI	Creates a dialog that lets the end user display a list of all database catalogs available on the specified database server. This function calls <b>SQLRTGetDatabases</b> , which uses <b>SQLRT.dll</b> for InstallScript projects and <b>ISSQLSRV.dll</b> for InstallScript MSI projects.
<b>SQLRTComponentInstall</b>	InstallScript	Executes the SQL script that is associated with the specified component if the script is scheduled to run during installation.
<b>SQLRTComponentUninstall</b>	InstallScript	Executes the SQL script that is associated with the specified component if the script is scheduled to run during uninstallation.
<b>SQLRTConnect</b>	InstallScript	Establishes a connection using the specified credential.  <b>SQLRTConnect2</b> supersedes this function.
<b>SQLRTConnect2</b>	InstallScript	Establishes a connection. This function must be called before file transfer if the connection is to be used to run scripts during installation. <b>SQLRTConnect2</b> returns the database server name when it fails to establish the connection. It uses <b>SQLRT.dll</b> so it can be called only after <b>SQLRTInitialize2</b> has already been called.

**Table 33 • SQL Functions (cont.)**

Function	Project Type	Description
<b>SQLRTConnectDB</b>	InstallScript	Establishes a connection to a specific catalog.
<b>SQLRTDoRollbackAll</b>	InstallScript	Executes all of SQL scripts scheduled to run during rollback.
<b>SQLRTGetBatchList</b>	InstallScript	<p>Returns the list of components that are associated with SQL scripts that need to be run when batch mode is enabled.</p> <p>For details about batch mode, see <a href="#">Specifying the Order for Running Multiple SQL Scripts That Are Associated with a Connection</a>.</p>
<b>SQLRTGetBatchMode</b>	InstallScript	Returns whether batch mode is enabled or disabled.
<b>SQLRTGetBrowseOption</b>	InstallScript	Returns the current value of the browse option for the SQL Server browse combo box and list box controls, which can display local servers, remote servers, server aliases, or a combination of these types.
<b>SQLRTGetComponentScriptError</b>	InstallScript	<p>Retrieves the last error while executing a SQL script that is associated with the component.</p> <p><b>SQLRTGetComponentScriptError2</b> supersedes this function.</p>
<b>SQLRTGetComponentScriptError2</b>	InstallScript	<p>Retrieves the last error while executing a SQL script that is associated with the component.</p> <p>This function takes several parameters (szScriptName, szTechnology, szServer, and szDB) that the <b>SQLRTGetComponentScriptError</b> function does not.</p>
<b>SQLRTGetConnectionAuthentication</b>	InstallScript, InstallScript MSI	Gets the default SQL Server connection authentication type.
<b>SQLRTGetConnectionInfo</b>	InstallScript, InstallScript MSI	Retrieves strings containing the connection information (the default server, database, default user name, and default password).
<b>SQLRTGetConnections</b>	InstallScript, InstallScript MSI	Retrieves a string list of connections that are present in the settings file.
<b>SQLRTGetDatabases</b>	InstallScript, InstallScript MSI	Returns a list of database catalogs that are available on the specified database server.

**Table 33 • SQL Functions (cont.)**

Function	Project Type	Description
<b>SQLRTGetErrorMessage</b>	InstallScript	Returns the descriptive message of the last error encountered by the SQL run time when a connection is being opened.
<b>SQLRTGetLastError</b>	InstallScript	Returns the text of the last error encountered by the SQL run time.  <b>SQLRTGetLastError2</b> supersedes this function.
<b>SQLRTGetLastError2</b>	InstallScript	Returns detailed information about the last error encountered by the SQL run time and loads the proper SQL error message.
<b>SQLRTGetScriptErrorMessage</b>	InstallScript	Returns the descriptive message of the last error encountered by the SQL run time when a SQL script is executing.
<b>SQLRTGetServers</b>	InstallScript, InstallScript MSI	Returns a list of database servers on the network for all database technologies included in the installation.  <b>SQLRTGetServers2</b> supersedes this function.
<b>SQLRTGetServers2</b>	InstallScript, InstallScript MSI	Returns a list of database servers for the database technologies that are specified for a connection. When szConnection is empty, this function behaves as <b>SQLRTGetServers</b> .
<b>SQLRTInitialize</b>	InstallScript	Loads the <b>SQLRT.dll</b> and initializes it using the settings file. This function must be the first function called in SQLRT.  <b>SQLRTInitialize2</b> supersedes this function.
<b>SQLRTInitialize2</b>	InstallScript, InstallScript MSI	Loads the <b>SQLRT.dll</b> file for InstallScript projects and the <b>ISSQLSRV.dll</b> file for InstallScript MSI projects, and it uses the settings file to initialize the .dll file.  This function must be the first function called in SQLRT or ISSQLSRV.
<b>SQLRTPutConnectionAuthentication</b>	InstallScript, InstallScript MSI	Sets the default SQL Server connection authentication type.

**Table 33 • SQL Functions (cont.)**

Function	Project Type	Description
<b>SQLRTPutConnectionInfo</b>	InstallScript, InstallScript MSI	Sets the connection information (the default server, default user name, and default password) so that it is available in the future. This is useful in situations when you need to recall what a user previously entered, like use of the Back button.  <b>SQLRTPutConnectionInfo2</b> supersedes this function.
<b>SQLRTPutConnectionInfo2</b>	InstallScript, InstallScript MSI	Sets the connection information (the default server, default database catalog, default user name, and default password) so that it is available in the future. This is useful in situations when you need to recall what an end user previously entered, like use of the Back button.
<b>SQLRTServerValidate</b>	InstallScript MSI	Tests connections specified in the installation.
<b>SQLRTSetBrowseOption</b>	InstallScript	Specifies whether the SQL Server browse combo box and list box controls should show local servers, remote servers, server aliases, or a combination of these types.
<b>SQLRTTestConnection</b>	InstallScript MSI	Tests all of the connections specified in the installation using the specified credential.  <b>SQLRTTestConnection2</b> supersedes this function.
<b>SQLRTTestConnection2</b>	InstallScript MSI	Establishes a connection.
<b>SQLServerLogin</b>	InstallScript, InstallScript MSI	Creates a dialog that is used by the script to specify SQL login credentials. These credentials include the login ID and password.
<b>SQLServerSelect</b>	InstallScript, InstallScript MSI	Creates a dialog to specify a server to target.



**Table 33 • SQL Functions (cont.)**

Function	Project Type	Description
<b>SQLServerSelectLogin</b>	InstallScript, InstallScript MSI	<p>Creates a login dialog that enables the targeted end user to specify which SQL Server should be used for the current connection, as well as which login credential should be used. The dialog displays a combo box that contains a list of SQL Servers accessed through DSNs. The end user can type a server name in the combo box or click the Browse button next to the combo box; clicking this button displays a list of all SQL Servers that are available on the network.</p> <p><b>SQLServerSelectLogin2</b> supersedes this function.</p>
<b>SQLServerSelectLogin2</b>	InstallScript, InstallScript MSI	<p>Creates a login dialog that is used by the default script. It lets the targeted end user specify which SQL Server should be used for the current connection, as well as which login credential should be used. The dialog displays a combo box that contains a list of SQL Servers accessed through DSNs. The end user can type a server name in the combo box or click the Browse button next to the Server Name combo box; clicking this button displays a list of all SQL Servers that are available on the network.</p> <p>This function also optionally shows the connection name that is associated with the connection information. In addition, it optionally enables end users to specify which database catalog should be used for the current connection.</p>
<b>SQLServerSelectLoginEx</b>	InstallScript, InstallScript MSI	<p>Creates a login dialog that is used by the default script. It lets the targeted end user specify which SQL Server should be used for the current connection, as well as which login credential should be used. The dialog displays a combo box that contains a list of SQL Servers accessed through DSNs. The end user can type a server name in the combo box or click the Browse button next to the Server Name combo box; clicking this button displays a list of all SQL Servers that are available on the network.</p> <p>This function also shows the connection name that is associated with the connection information.</p> <p><b>SQLServerSelectLogin2</b> supersedes this function.</p>

# String Functions

The string functions provide the ability to manipulate string variables and literals. String functions behave similarly to the standard C language functions. The return values also follow the C language convention.

**Table 34 • String Functions**

Function	Description
<b>CopyBytes</b>	Copies a specified number of bytes from one string to another.
<b>GetCArrayFromISArray</b>	Returns a pointer to an array of pointers that point to the actual data of the specified array. This function does not allocate any additional memory, but it returns a pointer to the data in the existing array.
<b>GetCHARArrayFromISStringArray</b>	Returns a pointer to an array of pointers to ANSI character strings that corresponds to the wide character strings that are contained in the specified array.
<b>GetDir</b>	Deletes the drive designation from a path or fully qualified file name.
<b>GetDisk</b>	Retrieves the disk drive designation from a path or fully qualified file name.
<b>NumToStr</b>	Converts a number to a string.
<b>ParsePath</b>	Retrieves the drive, path, file name, or extension from a path.
<b>StrAddLastSlash</b>	Adds a trailing backslash to a path specification if it does not already have one.
<b>StrCompare</b>	Compares one string to another.
<b>StrConvertSizeUnit</b>	Returns the appropriate display string for the InstallScript size unit constant that is specified.
<b>StrFind</b>	Finds a string in another string.
<b>StrFindEx</b>	Determines whether the string passed in the parameter szFindMe is found within the string passed in the parameter szString; the function begins its search at the location specified by nStart.
<b>StrGetTokens</b>	Gets a token from a string based on specified delimiters.
<b>StrLength</b>	Like <b>StrLengthChars</b> , returns the number of characters in a given string variable (that is, the number of code units in the UTF-16-encoded string) up to the first null character.

**Table 34** • String Functions (cont.)

Function	Description
<b>StrLengthChars</b>	Like <b>StrLength</b> , returns the number of characters in a given string variable (that is, the number of code units in the UTF-16-encoded string) up to the first null character.
<b>StrPutTokens</b>	Extracts list items from a specified string list and places them into the string that is specified by svString.
<b>StrRemoveLastSlash</b>	Removes the last backslash in a path string.
<b>StrReplace</b>	Searches svResult, beginning at the location specified by nStart, and replaces all found instances of szFind with szReplace.
<b>StrSub</b>	Returns a substring from a string.
<b>STRTOCHAR</b>	Returns the first character of a string as data of type CHAR.
<b>StrToLower</b>	Converts all alphabetic characters in string to lowercase.
<b>StrToNum</b>	Converts a string to a number.
<b>StrToNumHex</b>	Converts a string to a number—for example, 0x1A to 26.
<b>StrToUpper</b>	Converts all alphabetic characters in string to uppercase.
<b>StrTrim</b>	Removes the leading and trailing spaces and tabs from a string.

## Suite/Advanced UI and Advanced UI Interaction Functions



**Project** • This information applies to the following project types:

- InstallScript
- Suite/Advanced UI



**Note** • All of these functions are available for InstallScript installations that may be included as InstallScript packages in an Advanced UI or Suite/Advanced UI installation. For more information, see *Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project*.

Furthermore, all of these functions except for **FeatureConfigureFeaturesFromSuite** and **SuiteReportError** are available for InstallScript actions that are included in Suite/Advanced UI installations. For more information, see *Working with an Action that Runs InstallScript Code in a Suite/Advanced UI Installation*.

The **FeatureConfigureFeaturesFromSuite** is also available in the following scenarios; however, the other Advanced UI or Suite/Advanced UI functions return an error:

- The function is called in an *InstallScript* installation that is launched directly (that is, not from an Advanced UI or Suite/Advanced UI installation).
- The function is called in an *InstallScript* installation that is included in an Advanced UI or Suite/Advanced UI installation as an executable package.

*InstallScript* includes the following functions for interacting with an Advanced UI or Suite/Advanced UI installation that contains an *InstallScript* package. These functions (with the exception of **FeatureConfigureFeaturesFromSuite**) are also available for interacting with the running Suite/Advanced UI installation through an *InstallScript* action.

**Table 35 • Suite/Advanced UI and Advanced UI Interaction Functions**

Function	Description
<b>FeatureConfigureFeaturesFromSuite</b>	Sets feature states for the current <i>InstallScript</i> package that is running in an Advanced UI or Suite/Advanced UI installation based on the values of the Advanced UI or Suite/Advanced UI properties <b>ISFeatureInstall</b> and <b>ISFeatureRemove</b> . The function is called by the default code in the <i>OnSuiteInstallBefore</i> event (for an install operation) and <i>OnSuiteMaintBefore</i> event (for a modify operation).
<b>SuiteFormatString</b>	Resolves Advanced UI or Suite/Advanced UI properties in a string with values from the Advanced UI or Suite/Advanced UI installation.
<b>SuiteGetProperty</b>	Retrieves the value of an Advanced UI or Suite/Advanced UI property from the Advanced UI or Suite/Advanced UI installation.
<b>SuiteLogInfo</b>	Logs information about the <i>InstallScript</i> package or action that is running in an Advanced UI or Suite/Advanced UI installation to the Advanced UI or Suite/Advanced UI debug log.
<b>SuiteReportError</b>	Displays a message box in the Advanced UI or Suite/Advanced UI user interface to report an error that occurred as the Advanced UI or Suite/Advanced UI installation ran the <i>InstallScript</i> package.
<b>SuiteResolveString</b>	Replaces an Advanced UI or Suite/Advanced UI string identifier with its corresponding string value in the <i>InstallScript</i> package or action that is running in an Advanced UI or Suite/Advanced UI installation.
<b>SuiteSetProperty</b>	Sets the value of an Advanced UI or Suite/Advanced UI property in the Advanced UI or Suite/Advanced UI installation.

## Text Substitutions



**Project** • This information applies to *InstallScript* projects.

Text substitution consists of associating a string with another string—for example, associating "<MYTEXTSUB>" with "My Text Sub Value"—and of replacing the former string with the latter in other strings—for example, changing "This string demonstrates text substitution <MYTEXTSUB>" to "This string demonstrates text substitution My Text Sub Value". A text-substitution association can be either global—that is, applying to the scripts of the main installation and any included objects—or local—that is, applying only to the script file in which it occurs, any script files that are included in that script by using the #include preprocessor directive, and any script files in which that script is included. A local text-substitution association that is defined in an object script applies only to that object, not to the main installation or any other objects that are included in the installation; a local text-substitution association that is defined in the main installation script applies only to the main installation, not to any objects that are included in the installation.

A text-substitution association can be embedded in another text-substitution association; for example, "<MYTEXTSUB1>" can be associated with "My Text Sub 1 Value" and "<MYTEXTSUB2>" with "Text Sub <MYTEXTSUB1> Embedded".

InstallScript includes the following functions for using text substitutions:

**Table 36 • Text Substitutions Functions**

Function	Description
<b>TextSubGetValue</b>	Retrieves the text-substitution string that is associated with the specified string.
<b>TextSubParseTextSub</b>	Locates the first text substitution in the specified string.
<b>TextSubSetValue</b>	Creates a text-substitution association between the specified strings.
<b>TextSubSubstitute</b>	Performs text substitution on the specified string variable.

## Uninstallation Functions

The following functions perform services required for the uninstallation and/or maintenance setup of an installed application.

**Table 37 • Uninstallation Functions**

Function	Description
<b>FeatureGetTotalCost</b>	Determines the total space required for the feature installations and uninstallations that have been specified.
<b>FeatureTransferData</b>	Performs the feature installations and uninstallations that have been specified.
<b>InstallationInfo</b>	Creates registry keys based on a company name, product name, and product version number.
<b>RegDBGetItem</b>	Retrieves values under the per application paths key or the application uninstallation key.

**Table 37 •** Uninstallation Functions (cont.)

Function	Description
<b>RegDBSetItem</b>	Assign values under the per application paths key or the application uninstallation key.

## User Interface Functions

User interface functions allow you to customize certain error messages and titles of error boxes. However, several internal error messages that may be encountered during the development of the setup cannot be modified using user interface functions.

**Table 38 •** Visual Interface Functions

Function	Description
<b>Disable</b>	Disables the display of a user interface object.
<b>Enable</b>	Enables the display of a user interface object.
<b>FindWindow</b>	Retrieves the handle to a window.
<b>PlaceBitmap</b>	Inserts an image into the installation window.
<b>PlaceWindow</b>	Sets the position of user interface objects.
<b>PlayMMedia</b>	Plays an Adobe Flash application file (.swf), an AVI file, or a sound file (MIDI or WAVE).
<b>RGB</b>	Returns a custom color value based on the specified red, green and blue values.
<b>SetColor</b>	Changes the color of various user-interface elements.
<b>SetDialogTitle</b>	Creates custom dialog titles.
<b>SetDisplayEffect</b>	Sets the display effect for bitmap and metafile images.
<b>SetErrorMsg</b>	When a disk error occurs, this function sets the corresponding error message that is displayed by the <b>EnterDiskError</b> function.
<b>SetErrorTitle</b>	When a disk error occurs, this function sets the title bar for the error message that is displayed by the <b>EnterDiskError</b> function.
<b>SetFont</b>	Sets the font type and style.
<b>SetStatusWindow</b>	Sets the text for the status window and sets the initial percentage completed value that is displayed in the progress indicator.
<b>SetTitle</b>	Sets the text and color of the title in the main window.

**Table 38 •** Visual Interface Functions (cont.)

Function	Description
<b>SizeWindow</b>	Specifies the size of most user interface objects.
<b>StatusUpdate</b>	Sets the percent complete to display in the progress indicator after the next file transfer operation.

## Version-Checking Functions

The following functions enable you to access version information that is present on Windows-based systems. In order to use the functions, you need to know background information about version resources. Review Microsoft Windows documentation to gain a better understanding of the version resource.

**Table 39 •** Version-Checking Functions

Function	Description
<b>VerCompare</b>	Compares two strings containing version information.
<b>VerFindFileVersion</b>	Searches for the specified file and retrieves its version and location.
<b>VerGetFileLanguages</b>	Retrieves the list of languages that a specified file supports.
<b>VerGetFileVersion</b>	Retrieves the version of a specified file.
<b>VerProductCompareVersions</b>	Compares product versions.
<b>VerProductGetInstalledVersion</b>	Returns the string equivalent of the data in the Version value of the application uninstallation registry key if that data is a packed DWORD.
<b>VerProductIsVersionSupported</b>	Checks whether a version string is a supported version.
<b>VerProductNumToStr</b>	Returns a string for a specified packed DWORD version.
<b>VerProductStrToNum</b>	Returns a packed DWORD version for a specified string.
<b>VerProductVerFromVerParts</b>	Retrieves the packed DWORD that corresponds to the version parts that are specified by nVersionMajor, nVersionMinor, and nVersionBuild.
<b>VerProductVerPartsFromVer</b>	Retrieves as separate numeric values the version parts of a specified packed DWORD.
<b>VerSearchAndUpdateFile</b>	Replaces an existing file with a more recent version. If the specified file does not exist, the more recent version is installed.

**Table 39 •** Version-Checking Functions (cont.)

Function	Description
<b>VerUpdateFile</b>	Replaces an existing file with a more recent version. If the specified file does not exist, the more recent version is installed.

## Windows Installer Functions

Windows Installer functions, or APIs, are functions exported by the Windows Installer engine. They enable you to query and temporarily manipulate tables of a running installation.

One common use for adding temporary records to a running database is to populate user-interface elements with data not available until run time. For example, you can use Windows Installer APIs to populate a ListBox control in dialog with a list of mapped network drives, user accounts, directory names, or other data that can be discovered only while an installation is running on a particular target system.

For more information about any Windows Installer function, see the Windows Installer Help.

## Windows Installer API Functions

You can call Windows Installer API functions from the main script and from within an InstallScript custom action. InstallScript supports these Windows Installer API functions:



**Note •** Most Windows Installer API functions take a handle to the currently running database as an argument. For an InstallScript custom action, the database handle is the *HWND* argument passed to the custom action. In an event-handler function, you can use the global variable *ISMSI\_HANDLE*, which stores the handle to the running .msi database.



**Note •** For Windows APIs that require a buffer size and when the size of the value of the buffer size is greater than 1024 characters, a valid buffer size must be specified. For an example of how to write code for this scenario, refer to *Changes in Behavior for Some MSI APIs That Are Called in InstallScript Custom Actions* in the “Upgrading Projects from InstallShield 2011 or Earlier” topic.

**Table 40 •** Windows Installer API Functions

MsiApplyPatch	MsiGetLanguage	MsiRecordSetInteger
MsiCloseHandle	MsiGetLastErrorRecord	MsiRecordSetStream
MsiCreateTransformSummaryInfo	MsiGetMode	MsiRecordSetString
MsiDatabaseApplyTransform	MsiGetProperty	MsiSequence
MsiDatabaseExport	MsiGetSourcePath	MsiSetComponentState
MsiDatabaseGenerateTransform	MsiGetSummaryInformation	MsiSetFeatureAttributes



**Table 40 • Windows Installer API Functions (cont.)**

MsiDatabaseGetPrimaryKeys	MsiGetTargetPath	MsiSetFeatureState
MsiDatabaseImport	MsiInstallProduct	MsiSetInstallLevel
MsiDatabaseIsTablePersistent	MsiOpenDatabase	MsiSetMode
MsiDatabaseMerge	MsiOpenPackage	Msi SetProperty
MsiDatabaseOpenView	MsiPreviewBillboard	MsiSetTargetPath
MsiDoAction	MsiPreviewDialog	MsiSummaryInfoGetProperty
MsiEnumComponentCosts	MsiProcessMessage	MsiSummaryInfoSetProperty
MsiEvaluateCondition	MsiRecordClearData	MsiVerifyDiskSpace
MsiFormatRecord	MsiRecordDataSize	MsiViewClose
MsiGetActiveDatabase	MsiRecordGetFieldCount	MsiViewExecute
MsiGetComponentState	MsiRecordGetInteger	MsiViewFetch
MsiGetFeatureCost	MsiRecordGetString	MsiViewGetColumnInfo
MsiGetFeatureState	MsiRecordIsNull	MsiViewGetError
MsiGetFeatureValidStates	MsiRecordReadStream	

## Windows Installer API Functions by Category

This section includes the function signatures for the Windows Installer API functions available in InstallScript. See the Windows Installer Help for information about a function's usage, parameters, return values, and sequencing restrictions.

## MSI Property and Mode Functions

**Table 41** • MSI Property and Mode Functions

Function	Description
<b>prototype</b> INT <b>Msi SetProperty</b> (HWND, BYVAL STRING, BYVAL STRING);	Sets the value of a Windows Installer property. Creates the property if it does not exist. (For an example, see Getting or Setting Properties.)
<b>prototype</b> INT <b>Msi GetProductInfo</b> (BYVAL STRING, BYVAL STRING, BYVAL STRING, BYREF INT);	Returns product information for published and installed products.
<b>prototype</b> INT <b>MsiGetProperty</b> (HWND, BYVAL STRING, BYREF STRING, BYREF INT);	Gets the value of a Windows Installer property. Returns a null string ("") if the property does not exist.
<b>prototype</b> INT <b>MsiGetLanguage</b> (HWND);	Returns the numeric language ID for the running installation.
<b>prototype</b> BOOL <b>MsiGetMode</b> (HWND, INT);	Returns an internal boolean Installer state.
<b>prototype</b> INT <b>MsiSetMode</b> (HWND, INT, BOOL);	Sets an internal boolean Installer state.

## Feature and Component Functions

**Table 42** • Feature and Component Functions

Function	Description
<b>prototype</b> INT <b>MsiGetFeatureState</b> (HWND, BYVAL STRING, BYREF INT, BYREF INT);	Gets the installation state and action state of a feature.
<b>prototype</b> INT <b>MsiSetFeatureState</b> (HWND, BYVAL STRING, INT);	Sets the installation state of a feature.
<b>prototype</b> INT <b>MsiSetFeatureAttributes</b> (HWND, BYVAL STRING, INT);	Sets the attributes for a feature.
<b>prototype</b> INT <b>MsiGetFeatureValidStates</b> (HWND, BYVAL STRING, BYREF INT);	Returns a set of bit flags representing the valid installation states of a feature.

**Table 42** • Feature and Component Functions (cont.)

Function	Description
<b>prototype</b> INT <b>MsiGetComponentState</b> (HWND, BYVAL STRING, BYREF INT, BYREF INT);	Gets the installation state and action state of a component.
<b>prototype</b> INT <b>MsiSetComponentState</b> (HWND, BYVAL STRING, INT);	Sets the installation state of a component.
<b>prototype</b> INT <b>MsiGetFeatureCost</b> (HWND, BYVAL STRING, INT, INT, BYREF INT);	Returns the disk cost of a feature (in units of 512 bytes), and optionally its parent and child features.
<b>prototype</b> INT <b>MsiSetInstallLevel</b> (HWND, INT);	Sets the install level for the entire product.

## Directory Functions

**Table 43** • Directory Functions

Function	Description
<b>prototype</b> INT <b>MsiGetSourcePath</b> (HWND, BYVAL STRING, BYREF STRING, BYREF INT);	Returns the full source path for a directory listed in the Directory table. (The Directory table is exposed in the Direct Editor.)
<b>prototype</b> INT <b>MsiGetTargetPath</b> (HWND, BYVAL STRING, BYREF STRING, BYREF INT);	Returns the full target path for a directory listed in the Directory table.
<b>prototype</b> INT <b>MsiSetTargetPath</b> (HWND, BYVAL STRING, BYVAL STRING);	Sets the full target path for a directory listed in the Directory table.
<b>prototype</b> INT <b>MsiVerifyDiskSpace</b> (HWND);	Verifies if sufficient disk space exists for the current installation.

## Database Functions

With the exception of `MsiGetActiveDatabase`, the first `HWND` argument in most of these functions is a handle to a specific database view or record.

**Table 44** • Database Functions

Function	Description
<b>prototype</b> INT <b>MsiEvaluateCondition</b> (HWND);	Evaluates a conditional expression that contains property names and values.

**Table 44 • Database Functions (cont.)**

Function	Description
<b>prototype INT</b> <b>MsiGetActiveDatabase(HWND);</b>	Obtains a handle to the running .msi database, which you can use to open database views.
<b>prototype INT</b> <b>MsiDatabaseApplyTransform(HWND, BYVAL STRING, INT);</b>	Applies a transform to a database. A transform is a way of recording changes to a database without altering the original database.
<b>prototype INT</b> <b>MsiDatabaseExport(HWND, BYVAL STRING, BYVAL STRING, BYVAL STRING);</b>	Exports an installer table from an open database to a text archive file.
<b>prototype INT</b> <b>MsiDatabaseGenerateTransform(HWND, BYVAL STRING, INT, INT);</b>	Generates a transform file of differences between two databases. A transform is a way of recording changes to a database without altering the original database.
<b>prototype INT</b> <b>MsiDatabaseGetPrimaryKeys(HWND, BYVAL STRING, BYREF HWND);</b>	Returns a record containing the names of all the primary key columns for a specified table. This function returns a handle that should be closed using MsiCloseHandle.
<b>prototype INT</b> <b>MsiDatabaseImport(HWND, BYVAL STRING, BYVAL STRING);</b>	Imports an installer text archive table into an open database.
<b>prototype INT</b> <b>MsiDatabaseIsTablePersistent(HWND, BYVAL STRING);</b>	Returns an enumeration describing the state of a particular table.
<b>prototype INT</b> <b>MsiDatabaseMerge(HWND, HWND, BYVAL STRING);</b>	Merges two databases together, allowing duplicate rows.
<b>prototype INT</b> <b>MsiDatabaseOpenView(HWND, BYVAL STRING, BYREF INT);</b>	Prepares a database query, creating a view object.
<b>prototype INT</b> <b>MsiFormatRecord(HWND, HWND, BYREF STRING, BYREF INT);</b>	Formats record field data and properties using a format string.
<b>prototype INT</b> <b>MsiViewModify(HWND, INT, HWND);</b>	Modifies a database record. For a running installation, only temporary database changes are allowed.
<b>prototype INT</b> <b>MsiOpenDatabase(BYVAL STRING, BYVAL STRING, BYREF HWND);</b>	Opens a database file for data access. This function returns a handle that should be closed using MsiCloseHandle.
<b>prototype INT</b> <b>MsiViewClose(HWND);</b>	Closes an executed database view.

Table 44 • Database Functions (cont.)

Function	Description
<b>prototype</b> INT <b>MsiViewExecute</b> (HWND, HWND);	Executes a SQL query.
<b>prototype</b> INT <b>MsiViewFetch</b> (HWND, BYREF HWND);	Fetches a record for the current database view.
<b>prototype</b> INT <b>MsiRecordGetString</b> (HWND, INT, BYREF STRING, BYREF INT);	Returns the string stored in a specific field of the specified record.
<b>prototype</b> INT <b>MsiRecordSetString</b> (HWND, INT, BYVAL STRING);	Sets the string stored in a specific field of the specified record.
<b>prototype</b> INT <b>MsiRecordReadStream</b> (HWND, INT, CHAR, POINTER);	Returns the string value of a record field.
<b>prototype</b> INT <b>MsiRecordSetStream</b> (HWND, INT, BYVAL BINARY);	Sets a record stream field from a file. Stream data cannot be inserted into temporary fields.
<b>prototype</b> INT <b>MsiRecordGetInteger</b> (HWND, INT);	Returns the integer stored in a specific field of the specified record.
<b>prototype</b> INT <b>MsiRecordSetInteger</b> (HWND, INT, INT);	Sets the integer stored in a specific field of the specified record.
<b>prototype</b> INT <b>MsiViewGetColumnInfo</b> (HWND, INT, BYREF INT);	Returns a record containing database column names or definitions.
<b>prototype</b> INT <b>MsiRecordGetFieldCount</b> (HWND);	Returns the number of fields (columns) in a record.
<b>prototype</b> INT <b>MsiCloseHandle</b> (HWND);	Closes a database, view, or record handle.
<b>prototype</b> <b>MsiCloseAllHandles</b> ( );	Closes all open handles. Provided for diagnostic purposes, and should not be called for general cleanup.
<b>prototype</b> INT <b>MsiViewGetError</b> (HWND, BYREF STRING, BYREF INT);	Returns an error code for an error generated by MsiViewModify.

## Summary Information Stream Management Functions

**Table 45** • Summary Information Stream Management Functions

Function	Description
<b>prototype</b> <b>MsiGetSummaryInformation(HWND, BYVAL STRING, INT, BYREF HWND);</b>	Obtains a handle to summary information data for an installer database. This function returns a handle that should be closed using MsiCloseHandle.
<b>prototype</b> INT <b>MsiSummaryInfoGetProperty(HWND, INT, BYREF INT, BYREF INT, POINTER, BYREF STRING, BYREF INT);</b>	Gets a single property from the summary information.
<b>prototype</b> INT <b>MsiSummaryInfoSetProperty(HWND, INT, INT, INT, POINTER, BYREF STRING);</b>	Sets a single summary information property.

## Miscellaneous Functions

**Table 46** • Miscellaneous Functions

Function	Description
<b>prototype</b> INT <b>MsiApplyPatch(BYVAL STRING, BYVAL STRING, INT, BYVAL STRING);</b>	For each product listed by the patch package as eligible to receive the patch, the MsiApplyPatch function invokes an installation and sets the PATCH property to the path of the patch package.
<b>prototype</b> HWND <b>MsiCreateRecord(INT);</b>	Creates a new record object with the specified number of fields. This function returns a handle that should be closed using MsiCloseHandle.
<b>prototype</b> INT <b>MsiDoAction(HWND, BYVAL STRING);</b>	Executes a built-in action, custom action, or user-interface wizard action.
<b>prototype</b> INT <b>MsiEvaluateCondition(HWND, BYVAL STRING);</b>	Evaluates a conditional expression containing property names and values.
<b>prototype</b> INT <b>MsiInstallProduct(BYVAL STRING, BYVAL STRING);</b>	Installs or uninstalls a product.
<b>prototype</b> INT <b>MsiOpenPackage(BYVAL STRING, BYREF HWND);</b>	Opens a package for use with the functions that access the product database. The MsiCloseHandle function must be called with the handle when the handle is no longer needed.
<b>prototype</b> INT <b>MsiPreviewBillboard(HWND, BYVAL STRING, BYVAL STRING);</b>	Displays a billboard within a host control in the displayed dialog. Supplying a null billboard name removes any billboard displayed.

**Table 46 • Miscellaneous Functions (cont.)**

Function	Description
<b>prototype INT MsiPreviewDialog(HWND, BYVAL STRING);</b>	Displays a dialog as modeless and inactive.
<b>prototype int MsiProcessMessage(HWND, int, HWND);</b>	Sends an error record to the installer for processing.
<b>prototype INT MsiSequence(HWND, BYVAL STRING, INT);</b>	Executes another action sequence, as described in the specified table.

## Windows Installer API Functions Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates several Windows API functions that are used
 * to access the tables of the .msi database at run time and
 * add temporary records.
 *
 * Note: Before running this script, add a ListBox control to
 *       the ReadyToInstall dialog, and associate the control
 *       with the property LISTBOXPROP.
 *
 * This script populates the ListBox control in the ReadyToInstall
 * dialog with the current values of every property listed in
 * the Property table. The end user's selection for the ListBox
 * is stored in LISTBOXPROP.
 *
\*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "ifx.h"

// InstallScript custom action entry point
export prototype PropDisplay(HWND);

function PropDisplay(hInstall)
    HWND hDB, hViewlist, hRecordlist;
    HWND hViewprop, hRecordprop;
    NUMBER nBuffer, r;
    STRING svPropname, svPropvalue;
begin

    hDB = MsiGetActiveDatabase(hInstall);

    // open view into ListBox table
    MsiDatabaseOpenView(hDB,
        "SELECT * FROM `ListBox` WHERE `Property`='LISTBOXPROP'",
        hViewlist);

```

```

MsiViewExecute(hViewlist, NULL);

// open view into Property table
MsiDatabaseOpenView(hDB,
    "SELECT * FROM `Property`", hViewprop);
MsiViewExecute(hViewprop, NULL);

r = 0;

// for each Property record, add PROPNAME="value" record
// to ListBox table
while (MsiViewFetch(hViewprop, hRecordprop) != ERROR_NO_MORE_ITEMS)

    nBuffer = 256; // set size buffer
    MsiRecordGetString(hRecordprop, 1, svPropname, nBuffer);
    nBuffer = 256; // reset size buffer
    MsiGetProperty(hInstall, svPropname, svPropvalue, nBuffer);

    r = r + 1;

hRecordlist = MsiCreateRecord(4);

MsiRecordSetString(hRecordlist, 1, "LISTBOXPROP");
MsiRecordSetInteger(hRecordlist, 2, r);
MsiRecordSetString(hRecordlist, 3, svPropname);
MsiRecordSetString(hRecordlist, 4, svPropname + "=" + svPropvalue);

// can only temporarily modify running .msi database
MsiViewModify(hViewlist, MSIMODIFY_INSERT_TEMPORARY, hRecordlist);

endwhile;

MsiViewClose(hViewlist);
MsiViewClose(hViewprop);
end;

```



# Operators

An operator is a symbol that specifies a basic operation to be performed using one or two operands—such as addition. The operands can be constants, variables, or function calls. In the example below, the plus sign ( + ) indicates that the variable to the immediate left and the value to the immediate right should be added.

```
nCounter + 1;
```

## Operator Functionality

Most InstallScript operators correspond to operators in C and behave very much like their counterparts in that language. As in C, the function of some operators, such as + and ^ depends on the data type of the operands. For example, when a plus sign is used with numeric operands, addition is performed. When a plus sign is used with string operands, concatenation is performed.

## Expressions

The combination of operator and operands is called an expression. The example above is a simple expression because it includes only one operator. Complex expressions like the one below specify multiple operations:

```
nPrincipal * nRate - nFee;
```

## Evaluating Complex Expressions

In complex expressions, the order in which operations are performed depends on the order of operator precedence. InstallScript operators follow the same order of precedence as C operators. As with C, you can enclose an expression within parentheses to override the order of precedence.

## Results

Most expressions produce a new value based on the existing value of the operands. For that reasons, expressions are also referred to by the data type of the result they produce. Arithmetic expressions produce a numeric value, Boolean expressions produce TRUE or FALSE. String expressions produce a string.

# Address Operator (&)

The address operator is a unary operator that can be used to obtain the memory address of any variable in your script. The operator itself should precede the variable name, with no intervening space. You can use the address operator to assign the address of a variable to a pointer variable or to pass the address of a variable as an argument in a function call. You can send the address to C programs and operate on them as you would any standard C pointer.

In the example below, the address of a data structure is assigned to a pointer variable:

```
typedef DIMENSIONS
begin
    SHORT sLength;
    SHORT sWidth;
end;

DIMENSIONS rectangle;
DIMENSIONS POINTER pointerObject;

begin
```

```
pointerObject = &rectangle;

// . .

end;
```



**Caution** • If you use the address operator with a local variable, be aware that the local variable exists only during the life of the function in which it is declared. After the function returns, the address of the local variable will no longer be valid.

## Append to Path Operator (^)

Use the append to path ( ^ ) operator when you are combining two paths or a path and a file name. The operator automatically checks to see if you have added the proper number of backslashes when appending a file name or a subdirectory to a path.

If you type:

```
szStringVar = "C:\\MyPath\\" ^ "YourPath\\FileName";
```

the output from szStringVar is:

**C:\\MyPath\\YourPath\\FileName**

If you forgot to add the backslashes after MYPATH, as shown below:

```
szStringVar = "C:\\MyPath" ^ "YourPath\\FileName";
```

the result of the operation is still valid. The InstallScript ^ operator adds the backslashes for you.



**Note** • Do not use the ^ operator to join strings other than paths, such as registry keys. For other types of strings, use the concatenate operator (+).

## Arithmetic Operators (+, -, \*, /)

Arithmetic operators perform mathematical operations such as addition and subtraction with operands. There are two types of mathematical operators, unary and binary. Unary operators perform an action with a single operand. Binary operators perform actions with two operands. In complex expressions—those that include two or more operations—the order of evaluation depends on operator precedence.

## Arithmetic Operator Precedence

When the InstallScript compiler encounters a complex expression—one that includes two or more simple expressions—it evaluates those expressions one at a time.

## Operator Precedence

The order in which expressions are evaluated is determined by operator precedence. The compiler evaluates arithmetic operators using the same order of precedence that the C language uses:

1. Negative ( - ) unary.
2. Multiplication and division.
3. Addition and subtraction.
4. Left to Right Processing

If an expression contains two or more operators with the same precedence level, the operator to the left is processed first. For example, in the expression  $15 / 3 * 7$ , the InstallScript compiler first performs the division  $(15 / 3)$ , then multiplies the result by 7.

## Using Parentheses to Affect Processing

When a lower order precedence operation must be performed first, it should be surrounded by parentheses. For example, if the addition must be performed before the multiplication in the expression  $30 / 3 + 7$ , place parentheses around  $3 + 7$ . The parentheses in the expression  $30 / (3 + 7)$  cause the compiler to add 3 and 7 first, yielding 10, and then divide 30 by 10 for a final result of 3.

### Nested Parentheses

You can nest parentheses within an expression. InstallShield allocates 20 temporary locations for calculating nested expressions. Therefore, you can nest 19 levels of operations within parentheses. The InstallShield Script Compiler performs the innermost operation first and works its way outward.

### Example Expression

For example, in the expression  $36 - (3 * (2 + 6 - 4))$ , the InstallScript compiler first performs the operation  $2 + 6$ , which yields 8, then subtracts 4 from 8, yielding 4, then multiplies 3 by 4, yielding 12, and finally subtracts 12 from 36, yielding 24.



**Note** • Within the inner parentheses, InstallShield performed the addition operation first because it was the further left of two operators with equal precedence.

## Binary Arithmetic Operators

The InstallScript compiler recognizes the binary arithmetic operators listed in the following table.

**Table 1** • Binary Arithmetic Operators

Symbol	Operation	Example	Description
+	Addition	$x + y$	Adds two operands.
-	Subtraction	$x - y$	Subtracts the second operand from the first operand.

**Table 1 • Binary Arithmetic Operators (cont.)**

Symbol	Operation	Example	Description
*	Multiplication	x * y	Multiplies two operands.
/	Division	x / y	Divides the first operand by the second operand.  Because InstallShield does not have floating-point data types, the fractional part of the answer is dropped. For example, 5/2 = 2.5 -- > 2.
%	Modulus	19 % 5	Divides the first operand by the second operand and returns the remainder. For example, nResult = 19 % 5; results in nResult = 4.



**Tip •** You should include a space both before and after an arithmetic operator to make your script more readable and to maintain a consistent appearance.

## Unary Arithmetic Operators

Unary operators are arithmetic operators that perform an action on a single operand. The InstallScript compiler recognizes two unary operators, negative ( - ) and positive ( + ).

### Negative

The negative unary operator reverses the sign of an expression from positive to negative or from negative to positive. The net result of using the negative unary operator before an expression is the same as multiplying the expression by -1.

### Positive

The positive unary operator has the same net result on an expression as multiplying that expression by 1. It does not change the sign of a negative number to positive.

## Assignment Operator

Use the assignment operator ( = ) to copy a constant, literal, variable, expression result, or function result to a variable of the same type, as shown in the example code fragment below:

```
STRING szName;
LONG nValue;
BOOL bDone;
HWND hInstance;
INT iStyle;
LIST LISTINFO;
```

```

szName = "InstallShield";
nValue = 15;
bDone = FALSE;
hInstance = 0;
iStyle = DLG_MSG_STANDARD|DLG_CENTERED;

```

```
LISTINFO = ListCreate(StringList);
```

InstallShield automatically sizes string variables that are declared without an explicit length, as in the example above. By default, InstallShield autosizes the string variable to 256 bytes. If you assign to that variable a string longer than 256 bytes (including the null terminator), InstallShield increases the amount of memory reserved for that string variable.

If you declare a string variable with an explicit length, you must make it long enough to receive the string you assign to it. In the example below, the string literal contains 51 characters. Therefore, both `szStringVarA` and `szStringVarB` must have a declared length of at least 52, which is just large enough to accommodate the string itself and the null terminator that is added automatically to the end of the string.

```

STRING szStringVarA[52], szStringVarB[52];
szStringVarA = "This is a sample string that is 51 characters long.";
szStringVarB = szStringVarA;

```



**Caution** • Unlike C++, *InstallScript* does not support multiple assignment operations in a single statement. In *InstallScript*, the statement `a = b = c` is equivalent to the C++ statement `a = b == c`. That is, the first operator is interpreted as an assignment operator; the second is interpreted as a relational operator. If `b` is equal to `c`, the value 1 (TRUE) is assigned to `a`; if `b` is not equal to `c`, the value 0 (FALSE) is assigned to `a`.

## Bit Operators (&, |, ^, ~, <<, >>)

Bit, or bitwise, operators allow you to manipulate individual bits in a numeric variable. In order to use the operators effectively, you need to be familiar with binary notation. This topic gives you an overview of binary operators, but it does not try to teach you binary notation.

Bit operators work like logical operators, with one exception: logical operators work with expressions, bit operators work with bits. The *InstallScript* compiler recognizes the bitwise operators listed in the table below:

**Table 2 • Bit Operators**

Symbol	Operator	Explanation
&	BitAND	BitAND sets a bit in the result to 1 (TRUE) only if the bits in both operands are 1s; otherwise, the result is 0 (FALSE).
	BitOR	Bit inclusive OR sets a bit in the result to 0 only if the bits in the operands are 0; otherwise, the result is 1.
^	BitXOR	Bit exclusive OR sets a bit in the result to 1 if the corresponding bits in the operands are different (one 1, the other 0). Otherwise, the result is 0.

**Table 2 • Bit Operators (cont.)**

Symbol	Operator	Explanation
~	BitNOT	<p>BitNOT is a unary operator that reverses every bit in the operand, changing every 1 to a 0 and vice versa, as in the following example:</p> <p>~00001100</p> <p>The result is 11110011.</p>
<<	Shift left	<p>Moves bits a specified number of bits to the left. For example, the expression shown below moves bits three spaces to the left:</p> <p>00001100 &lt;&lt; 3</p> <p>The result is 01100000.</p>
>>	Shift right	<p>Moves bits a specified number of bits to the right. For example, the expression shown below moves bits two spaces to the right:</p> <p>00001100 &gt;&gt; 2</p> <p>The result is 00000011.</p>

Shift operations work the same way in InstallScript that they do in the C language. When you shift by two bits to the right (>> 2), the two bit values furthest to the right are lost. The other bit values are shifted to the right two places, and the sign bit is shifted into the empty bits.

You can use shift operations to multiply and divide a value by a power of 2. Left-shifting an integer by *n* has the same effect as multiplying the number by 2 to the power of *n*. Right-shifting an integer by *n* has the same effect as dividing the number by 2 to the power of *n*.

## BYREF Operator

By default, the parameters of a user-defined function are passed by value; that is, a copy of the data specified by each parameter is passed to the function. Because the function operates on a copy, the original data cannot be changed by the function.



**Note •** *There is one exception: By default, string variables passed to DLL functions are passed by reference, so the value of the variable can be changed by the function.*

When you want a user-defined function to operate directly on a variable it receives in a parameter rather than on a copy of that variable, you must specify the BYREF operator with the parameter type declaration in the function prototype, as shown below:

```
prototype StrInvert( BYREF STRING );
```

The BYREF operator indicates that the parameter is to be passed by reference—that is, the actual variable is to be passed to the function, and any changes made to that variable will be visible to the caller when the function returns. Parameters that are passed by reference are often referred to as variable parameters because they require a variable. Constants and literals cannot be passed by reference.

## Using BYREF With Multiple Parameters

When a user-defined function has more than one parameter, the BYREF operator must be specified with each variable parameter. In the example below, the first and third parameters are passed by reference and the second parameter is passed by value:

```
prototype StrChangeChar( BYREF STRING, CHAR, BYREF BOOL);
```

## Limitations

User-defined structure members cannot be passed by reference. Attempting to do so results in a compiler error. Instead, you must pass a pointer to the entire structure by reference and then use the structure pointer operator (->) to access or modify the data elements of the structure.

An autosized string variable that is passed by reference to a function will not be autosized within the called function. If the function attempts to assign a value whose length is greater than the current size of that parameter, run-time error 401 occurs. To avoid this error, declare strings with a specific size when they are to be passed by reference to a function.

# BYVAL Operator

The keyword BYVAL can be specified in the parameter specification of a function prototype to indicate a parameter that is passed by value rather than by reference, as in the example below:

```
prototype DisplayString( BYVAL STRING );
```

When a parameter is passed by value, the function receives a copy of the value and any changes it makes to that value are local to the function.

By default, the parameters of a user-defined function are passed by value; so generally it is not necessary to specify this keyword in a function prototype. (There is one exception: by default, string variables passed to DLL functions are passed by reference, so the value of the variable can be changed by the function.)

When you want a user-defined function to operate directly on a variable it receives in a parameter rather than on a copy of that variable, you must specify the BYREF operator.

# Concatenate Operator (+)

Use the concatenate string operator ( + ) to join one string to the end of another. Concatenating two strings results in a new, third string. In the following example, two string constants are concatenated and the resulting string value is assigned to the string variable szThirdString; after the statement has been executed, the value of szThirdString is "First string Second string".

```
szThirdString = "First string " + "Second string";
```

Operands in a concatenation expression may be string literals, string constants, or string variables. In the example below, a string constant and a string literal are concatenated. The resulting string value is assigned to szThirdString.

```
#define FIRST_STRING "This is the first string"

STRING szThirdString;

// . . .

szThirdString = FIRST_STRING + "Second string";
```



**Caution** • When assigning the result of a concatenation expression to a string variable, you must ensure that the concatenated string is not too long for the string variable to which it is assigned. A statement that assigns a string to a variable of insufficient size produces run-time error 401.

## Indirection Operator (\*)

The indirection operator is a unary operator that can be used to obtain the value stored at the memory location referenced by a pointer variable. The indirection operator must precede the pointer variable name, with no intervening space.

In the following example, nvalue is a number and pnumber is a pointer to a number. The assignment statement is used to copy to nvalue the number being pointed to by pnumber.

```
nvalue = *pnumber;
```

The indirection operator can also be used to pass a value to a function that takes a number value as a parameter, as in the following example:

```
somefunction(*pnumber);
```

The following limitations apply to the indirection operator:

- The indirection operator can be used with number pointers only.
- The indirection operator cannot be used to assign a value to a memory location.
- The indirection operator cannot be used as an argument to a function whose parameter has been defined with the BYREF operator.
- The indirection operator cannot be used to declare a pointer.
- The indirection operator cannot be used to declare a variable parameter in a function declaration.

## Logical Operators (&&, ||, !)

Logical operators allow you to ask more than one relational question at the same time. For example, using a logical operator you can ask if y is greater than 7 and szFilePath contains "C:\\Program Files\\Company Name". Logical operators return either a TRUE (1) or FALSE (0) value. Like relational operators, they are used most often in if and while statements.



The InstallScript compiler recognizes the logical operators listed in the table below:

**Table 3 • Logical Operators**

Symbol	Operation	Example	Description
&&	AND	exp1 && exp2	True only if both exp1 and exp2 are true; otherwise, false.
	OR	exp1    exp2	True if either exp1 or exp2 is true; false (0) only if both are false.
!	NOT	!exp1	False if exp1 is true; true if exp1 is false.

Logical operators have a lower precedence than arithmetic or relational operators. Among logical operators, the AND operator has higher precedence than the OR operator.



**Caution •** Unlike C++, InstallShield performs complete Boolean evaluations of logical expressions. Consider the following if statement:

```
if (iVar = 10) && (MyFunction( ) = 0) then
    MessageBox("That is so true.", INFORMATION);
endif;
```

MyFunction will be called even if the expression to the left of the logical operator is false. To obtain the effect of short circuit Boolean evaluations (in which the expression to the right of the && is resolved only if the expression to the left of the && is true), use a nested if statement, as shown below:

```
if (iVar = 10) then
    if (MyFunction( ) = 0) then
        MessageBox("That is so true.", INFORMATION);
    endif;
endif;
```

## Member Operator (.)

Use the member operator to reference individual elements in a structure variable. The member operator must appear between the structure variable name and the element name, with no intervening space. In the example below, a literal value is assigned to each element in a structure variable.

```
typedef SETTINGSREC
begin
    BOOL bSwitchOn;
    STRING szMssg[255];
    INT nVal;
end;

SETTINGSREC settings;

settings.bSwitchOn = FALSE;
settings.szMssg = "Off";
settings.nVal = 0;
```

## Relational Operators (<, >, =, <=, >=, !=)

Relational operators compare one expression to another within the context of a conditional statement, such as an if or while statement. For example, the following statement asks “Is x greater than 20?”:

```
if (x > 20) then
```

The answer to the questions can be only TRUE (1) or FALSE (0). The InstallScript compiler recognizes the relational operators listed in the table below:

**Table 4 • Relational Operators**

Symbol	Operation	Example	Description
=	Equal	x = y	True if x is equal to y.
>	Greater than	x > y	True if x is greater than y.
<	Less than	x < y	True if x is less than y.
>=	Greater than or equal to	x >= y	True if x is greater than or equal to y.
<=	Less than or equal to	x <= y	True if x is less than or equal to y.
!=	Not equal to	x != y	True if x is not equal to y.

When you use a relational operator in an if...else statement, the program will follow one of two actions:

- If the expression is TRUE, statements that follow the if *are* executed. Statements that follow the else *are not* executed.
- If the expression is FALSE, statements that follow the if *are not* executed. Statements that follow the else *are* executed.

Relational operators have an overall lower precedence than arithmetic operators. Among just relational operators, less than, less than or equal to, greater than, and greater than or equal to have precedence over equal and not equal.



**Caution •** You cannot use assignment and relational operators in the same conditional expression. For example, the following will fail:

```
if ((ListID = ListCreate (NUMBERLIST)) = LIST_NULL) then . . . endif;
```



**Tip •** Unlike C, which uses == to test for equality, InstallScript's assignment operator and relational operator use the same symbol (=).

## Relational Operator Precedence

Relational operators have an overall lower precedence than arithmetic operators. This means that InstallShield performs all arithmetic operations before beginning to evaluate logical operations. Logical expressions are evaluated from left to right, unless the order of operations is modified by parentheses. Among relational operators the order of precedence is as follows:

- First, less than ( < ), less than or equal to ( <= ), greater than ( > ), and greater than or equal to ( >= ).
- Then, equal to ( = ) and not equal to ( != ).

Therefore, when combining arithmetic and relational expressions, the InstallShield Script Compiler evaluates precedence in the following order:

1. Negative (minus) unary.
2. Multiplication and division.
3. Addition and subtraction.
4. Less than ( < ), less than or equal to ( <= ), greater than ( > ), and greater than or equal to ( >= ).
5. Equal to ( = ) and not equal to ( != ).

In the expression `6 + 7 > y`, the InstallScript compiler adds 6 and 7 together and then compares the result (13) to `y`. To change the order of precedence, use parentheses. For example, in the expression `6 + (7 > y)`, the InstallScript compiler first determines if 7 is greater than `y`. If it is, then 1 (the numeric value of TRUE) is added to 6. If it is not, then 0 (the numeric value of FALSE) is added to 6.

When a statement contains arithmetic, relational, and logical operators, the compiler evaluates precedence in the following order:

1. Negative (minus) unary has first precedence.
2. Multiplication and division have second precedence.
3. Addition and subtraction have third precedence.
4. NOT ( ! ) has fourth precedence.
5. Less than ( < ), less than or equal to ( <= ), greater than ( > ), and greater than or equal to ( >= ) have fifth precedence.
6. Equal to ( = ) and not equal to ( != ) have sixth precedence.
7. AND ( && ) has seventh precedence.
8. OR ( || ) has eighth precedence.

Use logical operators in if and while statements the same way you used relational operators. The example below adds `nExampleSize` and `nHelpSize` if `bInstallExample` and `bInstallHelp` are true.

```
if (bInstallExample && bInstallHelp) then
    nTotalSize = nExampleSize + nHelpSize;
endif;
```

The next example sets `bPublicFile` to TRUE if *either* `bInstallProgram1` or `bInstallProgram2` is TRUE.

```
if (bInstallProgram1 || bInstallProgram2) then
    bPublicFile = TRUE;
endif;
```



**Note** • You cannot use the && or || operators within an argument to a function. Instead, follow the above example and assign the value of the logical expression to a Boolean variable and then call the function with the variable as an argument.

## String Operators (^, +, %)

The string operators allow you to directly manipulate strings without the use of functions. Note that string operators are not case-sensitive. The InstallScript compiler supports the following string operations:

**Table 5 • String Operators**

Operation	Description
Append to Path Operator (^)	Adds additional paths to a path or a file name.
Concatenate Operator (+)	Appends one string to the end of another string.
Find String Operator (%)	Locates a substring in another string.
String Constant Operator (@)	Access a string constant that is used for one of the project's string entries.



**Note** • Do not use parentheses to enclose expressions on either side of a string operator. For example, avoid statements like the following:

```
szPath = szTestPath ^ (AUTOFILE + ".bat");
```

Instead, create expressions without parentheses for use with string operators, as shown below:

```
szFile = AUTOFILE + ".bat";    szPath = szTestPath ^ szFile;
```

## String Constant Operator (@)

String entries enable you to access text strings for a given language from within your InstallScript code. That is, to keep the InstallScript code for your installation completely separate from any language-specific strings that you may want to display during the installation, you can refer to each string by using its string identifier. When you use string identifiers in your InstallScript, you must precede them with the at sign (@).

The String Editor view shows the collection of language-independent identifiers and corresponding language-specific values for your project. To learn more, see [Localizing the End-User Interface](#).

When you build a project that includes an InstallScript file (.rul) and the InstallScript code contains one or more references to string entries that use the @ operator, InstallShield validates the string entries at build time. If a string identifier in the project's InstallScript file is not defined in String Editor view, InstallShield displays build warning - 7174.



**Note** • The @ operator is case-insensitive when it comes to string identifiers. Therefore, when you use a string identifier in your script, you do not necessarily need to match the case of the string identifier that is specified in the String Editor view. However, mixing case may prevent InstallShield from matching the string entries in the script to the corresponding string entries in the String Editor view at build time. Therefore, it is recommended that you use uppercase for all instances of string identifiers.



**Tip** • As an alternative to the @ operator, you can use the **LoadStringFromStringTable** function if you want to provide your own missing string error handling.

## Structure Pointer Operator (->)

Use the structure pointer operator to reference individual elements in a structure by means of a pointer variable. The structure pointer operator must appear between the pointer variable name and the element name, with no intervening space. In the example below, a literal value is assigned to each element in a structure.

```
typedef  DIMENSIONS
begin
    SHORT sLength;
    SHORT sWidth;
end;
DIMENSIONS Table;
NUMBER   nvNumValue;
DIMENSIONS POINTER   pointerObject;
begin
    pointerObject = &Table;
    pointerObject->sLength = 500;
    pointerObject->sWidth = 750;
```



**Caution** • You can use only one structure pointer in an expression. If structure A contains a member (Bptr) that is a pointer to structure B, which contains a member (Cptr) that is a pointer to structure C, you cannot reference a member of C from A. The expression A.Bptr->Cptr->Cmember is invalid in InstallScript.

## Find String Operator (%)

Use the boolean Find String operator ( %) to determine if one string is a substring of another string. The following example tests szStringVarA to determine if it contains the string "sample." If it does, then MessageBox is called to display a message.

```
szStringVarA = "This is a sample string.";
if (szStringVarA % "sample") then
    MessageBox("Variable contains 'string'.", INFORMATION);
endif;
```

The character comparison is not case-sensitive. In the following example, the message box will be displayed:

```
typedef  DIMENSIONS
begin
    SHORT sLength;
```

```
    SHORT sWidth;  
end;  
  
DIMENSIONS Table;  
NUMBER    nvNumValue;  
DIMENSIONS POINTER    pointerObject;  
  
begin  
    pointerObject = &Table;  
    pointerObject->sLength = 500;  
    pointerObject->sWidth = 750;
```



---

**Note** • The `InstallScript` function [StrFind](#) also determines if a substring is contained within another string. If the substring is found, `StrFind` returns its position within the string.

# Objects and Object Handlers

This section describes the objects and object handlers that InstallScript supports.

## Objects

InstallScript supports the following objects.

**Table 1 • InstallScript Objects**

Object	Description
<b>Err Object</b>	Use for exception handling.
<b>Objects Object</b>	Obtain access to your project's InstallShield objects by either index or name.
<b>Reboot Object</b>	Pass command-line arguments to the installation after a reboot. This information is written to the appropriate registry key at the end of the installation, so that the installation runs after reboot.
<b>TextSub Object</b>	Make text substitutions.

## Err Object

The Err object is used for exception handling. It has the following properties and methods:

## Properties

**Table 2 • Err Object Properties**

Property	Description
<b>Number</b>	A number that identifies the error.
<b>Source</b>	A string that identifies the source of the error.
<b>Description</b>	A string that describes the error.
<b>HelpFile</b>	A string that specifies the fully qualified file name of a help file containing additional information about the error.
<b>HelpContext</b>	A number that specifies the identifier of a help topic containing additional information about the error.
<b>LastDllError</b>	This property holds the return value of the Windows API function <b>GetLastError</b> .

## Methods

**Table 3 • Err Object Methods**

Method	Description
<b>Clear</b>	Clears the values of the Err object's properties.
<b>Raise</b>	<p>If executed after the try keyword, script processing passes to the next exception handler (that is, code inside a catch/endcatch block); otherwise processing passes to the setup engine's built-in exception handler. Also resets the values of Err object properties as follows:</p> <ul style="list-style-type: none"> <li>• <code>Err.Raise( )</code>;—Does not reset any Err object property values.</li> <li>• <code>Err.Raise( nNumber )</code>;—Resets the value of <code>Err.Number</code> to <code>nNumber</code>.</li> <li>• <code>Err.Raise( nNumber, szSource )</code>;—Resets the value of <code>Err.Number</code> to <code>nNumber</code> and <code>Err.Source</code> to <code>szSource</code>.</li> <li>• <code>Err.Raise( nNumber, szSource, szDesc )</code>;—Resets the value of <code>Err.Number</code> to <code>nNumber</code>, <code>Err.Source</code> to <code>szSource</code>, and <code>Err.Description</code> to <code>szDesc</code>.</li> <li>• <code>Err.Raise( nNumber, szSource, szDesc, szHelpFile )</code>;—Resets the value of <code>Err.Number</code> to <code>nNumber</code>, <code>Err.Source</code> to <code>szSource</code>, <code>Err.Description</code> to <code>szDesc</code>, and <code>Err.HelpFile</code> to <code>szHelpFile</code>.</li> <li>• <code>Err.Raise( nNumber, szSource, szDesc, szHelpFile, nHelpContext )</code>;—Resets the value of <code>Err.Number</code> to <code>nNumber</code>, <code>Err.Source</code> to <code>szSource</code>, <code>Err.Description</code> to <code>szDesc</code>, <code>Err.HelpFile</code> to <code>szHelpFile</code>, and <code>Err.HelpContext</code> to <code>nHelpContext</code>.</li> </ul>



## Objects Object



**Project** • This information applies to *InstallScript* projects.

The Objects object is used for getting access to your project's InstallShield objects by either index or name; for example:

```
set obj1 = Objects(1);
set obj2 = Objects("New MFC 6.2 Runtime 1");
```

### Properties

**Table 4 •** Objects Object Properties

Property	Description
<b>Count</b>	The number of InstallShield objects included in your project.

## Reboot Object



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The Reboot object is used for passing command-line arguments to the **Setup.exe** file after a restart; this information is written to the appropriate registry key at the end of the installation, so that the installation runs after the system is restarted. For example, the following statement runs the installation in debug mode after a restart:

```
Reboot.CommandLine = "-d";
```

Note that the specified argument or arguments are added to the existing command line; they do not replace the existing command line. Note also that currently you cannot change the existing command line or remove text from it.

### Passing Parameters to the InstallScript Variable CMDLINE in an InstallScript MSI Installation

As with any **Setup.exe** command line, in an InstallScript MSI installation, if you are trying to pass information to the InstallScript variable CMDLINE variable when the **Setup.exe** file is launched after the restart, you must include the information after specifying the -z option. For example, the following adds **TEST1 TEST2** to the reboot string and thus the CMDLINE variable when the **Setup.exe** file is launched after the restart.

```
Reboot.CommandLine = -z"TEST1 TEST2"
```

Note that since multiple -z parameters are not supported, you must specify all information that is intended for the CMDLINE variable in a single Reboot.CommandLine call for your InstallScript MSI installation.

## TextSub Object



**Project** • This information applies to InstallScript projects.

The TextSub object is used for making text substitutions. It has the following property and method:

### Properties

**Table 5** • TextSub Object Properties

Property	Description
<b>Value( szIdentifier )</b>	The value of the string associated with the identifier szIdentifier.

### Methods

**Table 6** • TextSub Object Methods

Method	Description
<b>Substitute( szString )</b>	Replaces all angle-bracketed identifiers in szString with their associated strings.

### Example

The following code:

```
TextSub.Value( "SUBBED" ) = "substituted text";
szString = "123<SUBBED>456<UNSUBBED>789";
TextSub.Substitute( szString );
```

gives szString the value "123substituted text456<UNSUBBED>789".

## Object Handlers

InstallScript supports the following object handlers.

**Table 7** • InstallScript Object Handlers

Object Handler	Description
<b>InitProperties</b>	Executed when the object is inserted in a project. It initializes the values of the object script's variables that are used to read or write the values of object properties.
<b>ReadProperties</b>	Executed when the object project is opened and when an installation that includes the object is launched. It calls the appropriate ReadxxxxProperty functions to retrieve the property values stored in the property bag object.

**Table 7 •** InstallScript Object Handlers (cont.)

Object Handler	Description
<b>WriteProperties</b>	Executed when the object project is saved or built. It calls the appropriate WritexxxxProperty functions to save the property values to the property bag object.

## InitProperties



**Project •** This information applies to InstallScript projects.

The InitProperties handler is executed when the object is inserted in a project. It initializes the values of the object script's variables that are used to read or write the values of object properties.

If you add a property to the object project by using the Add New Property dialog box, an appropriate statement, based on your entry in the dialog box's Default Value edit box, is automatically placed in this handler.

## ReadProperties



**Project •** This information applies to InstallScript projects.

The ReadProperties handler is executed when the object project is opened and when a setup that includes the object is launched. It calls the appropriate ReadxxxxProperty functions to retrieve the property values stored in the property bag object. (Property values are saved to the property bag object by the WritexxxxProperty functions, which are called by the WriteProperties handler.)

If you add a property to the object project by using the Add New Property dialog box, an appropriate ReadxxxxProperty function call is automatically placed in this handler.

## WriteProperties



**Project •** This information applies to InstallScript projects.

The WriteProperties handler is executed when the object project is saved or built. It calls the appropriate WritexxxxProperty functions to save the property values to the property bag object. (Property values are retrieved from the property bag object by the ReadxxxxProperty functions, which are called by the ReadProperties handler.)

If you add a property to the object project by using the Add New Property dialog box, an appropriate WritexxxxProperty function call is automatically placed in this handler.

# Exception Handling

Exception handling lets you separate error handling from the rest of your script code. InstallScript supports exception handling with the Err object and the keywords try, catch, and endcatch.

If an exception is raised during execution of code that follows the try keyword, script processing passes to the next exception handler (that is, code inside a catch/endcatch block). After an exception handler has executed, processing passes to the line after its endcatch keyword. If no exception is raised by code that follows the try keyword, the code in the exception handler is skipped and processing resumes with the line after the endcatch keyword.

An exception can be raised by a call to the Err object's Raise method, which takes from zero to five arguments. You can retrieve the values of those arguments in the exception handler by checking the value of the corresponding Err object properties.



**Note** • Raised errors must be negative numbers; otherwise, the installation engine converts the error into the appropriate COM error. Therefore, any error code thrown should be a negative number.

The following code sample demonstrates exception handling:

```
askfile:
AskText( "Path to file?", "", svPathName );

try
    if (!Is( FILE_EXISTS, svPathName )) then
        /* If file does not exist, raise an exception.
           (ERR_NOT_EXIST must have been given a value
            in a #define statement. The error number
            must be a negative number.) */
        Err.Raise( ERR_NOT_EXIST );
    endif;

    if GetFileInfo ( svPathName, FILE_SIZE,
        nvFileSize, svResult )<0 then
        /* If file information could not be obtained,
           raise an exception. (ERR_NO_INFO must have been
            given a value in a #define statement. The error
            number must be a negative number.) */
        Err.Raise ( ERR_NO_INFO );
    endif;

    SprintfBox ( INFORMATION, "File Size", "Size of %s is %ld.",
        svPathName, nvFileSize );
catch
    /* Exception handler. */
    nTemp = Err.Number;
    /* Handle the exception based on its cause. */
    switch (nTemp)
        case ERR_NOT_EXIST:
            if AskYesNo( svPathName +
                " does not exist. Enter another path?", YES )=YES then
                bTryAgain = TRUE;
            endif;
        case ERR_NO_INFO:
            MessageBox ( "Could not get size of " +
                svPathName, INFORMATION );
            bTryAgain = FALSE;
```

```
        endswitch;  
    endcatch;  
  
    if bTryAgain then  
        bTryAgain = FALSE;  
        goto askfile;  
    endif;
```

Try/catch/endcatch blocks can be nested as in the following example:

```
try  
    /* Normal processing, part 1. */  
    try  
        /* Normal processing, part 2. */  
        catch  
            /* Exception handling for part 2. */  
            endcatch;  
        /* Normal processing, part 3. */  
    catch  
        /* Exception handling for parts 1 and 3. */  
    endcatch;
```



# Built-In Functions (A-D)

For a list of functions by category, see [Built-In Functions by Category](#).

## AddFolderIcon

The [CreateShortcut](#) function supersedes the **AddFolderIcon** function.

The **AddFolderIcon** function lets you perform tasks such as the following:

- Create a shortcut or folder on the Start menu, the Programs menu, or the desktop. Use the `szProgramFolder` parameter to specify the appropriate location for the shortcut or folder.
- Create a cascading submenu on the Startup menu, and include a shortcut in the submenu.



---

**Note** • The shortcut target must be present on the target system before **AddFolderIcon** can be called.


**AddFolderIcon** does not support the creation of Internet shortcuts.

### Syntax

```
AddFolderIcon ( szProgramFolder, szItemName, szCommandLine, szWorkingDir, szIconPath, nIcon,  
                szShortCutKey, nFlag );
```



## Parameters

**Table 1 • AddFolderIcon Parameters**

Parameter	Description
<b>szProgramFolder</b>	<p>Specify the name of the folder that should contain the shortcut, or specify the name of the program folder that you want to create. If the folder does not exist, the installation creates it. For this parameter, you can specify a subfolder in a multi-level cascading menu. If the subfolder does not exist, <b>AddFolderIcon</b> creates the subfolder and, if necessary, its parent folders.</p> <p>To add the shortcut to a specific folder, specify the fully qualified path—for example:</p> <p><b>C:\ProgramData\Microsoft\Windows\Start Menu\Programs</b></p> <p>To add a shortcut to the Programs menu on the Start menu, you can pass a null string ("") in this parameter.</p> <p>You can pass one of the following InstallScript system variables in this parameter:</p> <ul style="list-style-type: none"> <li>● <b>FOLDER_DESKTOP</b>—Adds the shortcut to the desktop.</li> <li>● <b>FOLDER_STARTUP</b>—Adds the shortcut to the Startup menu.</li> <li>● <b>FOLDER_STARTMENU</b>—Adds the shortcut to the Start menu.</li> <li>● <b>FOLDER_PROGRAMS</b>—Adds the shortcut to the Start\Programs menu.</li> </ul> <p>You can also specify a path relative to a folder that is identified by an InstallScript system variable—for example:</p> <p><b>FOLDER_PROGRAMS ^ "ACCESSORIES\GAMES"</b></p>
<b>szItemName</b>	<p>Specify the name of the shortcut. Calling <b>AddFolderIcon</b> to add a shortcut to a program folder also creates a link file in the links directory that is specified by szCommandLine. Note that Explorer shell does not allow the following characters in item names: /, \, :, ?, &lt;, &gt;, or  .</p>
<b>szCommandLine</b>	<p>Specify one of the following:</p> <ul style="list-style-type: none"> <li>● The fully qualified name of the executable file that is associated with the shortcut, including any command-line parameters. This is added to the Target value on the shortcut's Properties dialog box. To add a shortcut to the Start Programs menu, enter the fully qualified path of the links directory, which is where your application stores its icon link files.</li> <li>● The fully qualified path if szItemName is a subfolder.</li> </ul> <p> <b>Caution</b> • If the command line includes a long file name, it must be enclosed in quotes. Command-line parameters, however, should not be surrounded with quotation marks. For that reason, it is advisable to build the szCommandLine string from two separate strings.</p>



**Table 1 • AddFolderIcon Parameters (cont.)**

Parameter	Description
<b>szWorkingDir</b>	<p>Specify the working directory for the shortcut target.</p> <p>If szItemName is a subfolder, this parameter is not applicable.</p> <p><b>AddFolderIcon</b> writes this directory in the Start In box on the Shortcut tab of the shortcut's Properties dialog box. If you pass a null string ("") in this parameter, the function leaves this Start In box blank, and the path in the Target box is used.</p> <p></p> <p><b>Caution •</b> Do not call <b>LongPathToQuote</b> to enclose this path in quotation marks. InstallShield automatically encloses these paths in quotation marks.</p>
<b>szIconPath</b>	<p>Specify the fully qualified path to the file that contains the icon that you want to be displayed for the shortcut.</p> <p>If szItemName is a subfolder, this parameter is not applicable.</p> <p></p> <p><b>Caution •</b> Do not call <b>LongPathToQuote</b> to enclose this path in quotation marks. InstallShield automatically encloses these paths in quotation marks.</p>
<b>nIcon</b>	<p>Specify the icon index in the executable file that is specified by szIconPath. An icon index of 0 refers to the first icon in the file, an icon index of 1 refers to the second icon, and so on. If you are not using an icon, specify <b>0</b> in this parameter.</p> <p>If szItemName is a subfolder, this parameter is not applicable.</p>
<b>szShortCutKey</b>	<p>Specify the shortcut key (in the form of a string) that you want to be assigned to your shortcut. You can set szShortCutKey for the shortcut so that end users can press the appropriate hot keys to launch the shortcut.</p> <p>For example, if you want end users to be able to launch the product by pressing the CTRL key, the ALT key, and then the 1 key on the numeric keyboard, pass "Ctrl + Alt + 1" in this parameter.</p> <p>If szItemName is a subfolder, this parameter is not applicable.</p>
<b>nFlag</b>	<p>Pass one or more of the following predefined constants in this parameter. To pass two or more predefined constants in this parameter, combine those constants with the bitwise OR operator ( ).</p> <ul style="list-style-type: none"> <li>• <b>REPLACE</b>—Indicates that the current icon or shortcut in the folder is to be replaced.</li> <li>• <b>RUN_MAXIMIZED</b>—Indicates that the program should be maximized when launched.</li> <li>• <b>RUN_MINIMIZED</b>—Indicates that the program should be minimized when launched.</li> <li>• <b>NULL</b>—Indicates no options.</li> </ul>

## Return Values

**Table 2 • AddFolderIcon Return Values**

Return Value	Description
0	Indicates that the function successfully added or replaced the shortcut in the specified folder and associated the executable file with it.
< 0	Indicates that the function was unable to add or replace the shortcut and associate the executable file with it.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

## AddFolderIcon Examples

Choose one of the following examples:

- [Place a shortcut to an executable file on the Start menu and the Start Programs menu.](#) (AddFolderIcon Example 1)
- [Create a cascading submenu on the Startup menu and add a shortcut to the menu.](#) (AddFolderIcon Example 2)
- [Place a subfolder on the desktop and a shortcut pointing to an executable file in the new folder.](#) (AddFolderIcon Example 3)

### AddFolderIcon Example 1



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the AddFolderIcon function.
*
* This example places a shortcut to an executable file on the
* Start menu and the Start Programs menu.
*
* Note: Before running this script, set the preprocessor
*       constants so that they reference the fully qualified
*       names of the Windows Notepad executable and a valid
*       text file on the target system.
*
\*-----*/

#define PROGRAM "C:\\Windows\\Notepad.exe"
#define PARAM   "C:\\Windows\\Readme.txt"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"
```

```

export prototype ExFn_AddFolderIcon(HWND);

function ExFn_AddFolderIcon(hMSI)
    STRING szProgramFolder, szItemName, szCommandLine, szWorkingDir;
    STRING szShortCutKey, szProgram, szParam, szIconPath;
    NUMBER nIcon;
begin

    // Set up parameters for call to AddFolderIcon.
    szProgramFolder = FOLDER_STARTMENU;
    szItemName      = "Notepad Example 1";
    szProgram       = PROGRAM;
    szParam         = PARAM;

    LongPathToQuote (szProgram, TRUE);

    LongPathToShortPath (szParam);

    szCommandLine = szProgram + " " + szParam;
    szWorkingDir  = "";
    szIconPath    = "";
    nIcon         = 0;
    szShortCutKey = "";

    // Add a shortcut to the Start menu.
    if (AddFolderIcon (szProgramFolder, szItemName, szCommandLine, szWorkingDir,
                      szIconPath, nIcon, szShortCutKey, REPLACE) < 0) then
        MessageBox ("AddFolderIcon failed.", SEVERE);
    else
        sprintfBox (INFORMATION, "AddFolderIcon", "%s created successfully.",
                    szItemName);
    endif;

    szProgramFolder = "";
    szItemName      = "Notepad Example 2";

    // Add a shortcut to the Programs menu.
    if (AddFolderIcon (szProgramFolder, szItemName, szCommandLine, szWorkingDir,
                      szIconPath, nIcon, szShortCutKey, REPLACE) < 0) then
        MessageBox ("AddFolderIcon failed.", SEVERE);
    else
        sprintfBox (INFORMATION, "AddFolderIcon", "%s created successfully.",
                    szItemName);
    endif;

end;

```

## AddFolderIcon Example 2



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script

```

```

*
* Demonstrates the AddFolderIcon function.
*
* This example creates a cascading submenu on the Startup menu
* and adds a shortcut for an executable file to it.
*
* Note: Before running this script, set the preprocessor
*       constants so that they reference the fully qualified
*       names of the Windows Notepad executable file and a
*       valid text file on the target system.
*
\*-----*/

#define PROGRAM "C:\\Windows\\Notepad.exe"
#define PARAM   "C:\\Windows\\Readme.txt"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_AddFolderIcon(HWND);

function ExFn_AddFolderIcon(hMSI)
    STRING  szProgramFolder, szItemName, szCommandLine, szWorkingDir;
    STRING  szIconPath, szShortCutKey, szProgram, szParam;
    NUMBER  nIcon, nFlag, nResult;
begin

    // Set the fully qualified name of the Startup submenu.
    szProgramFolder = FOLDER_STARTUP ^ "SubMenu Example";

    // Construct the shortcut's command-line property.
    szProgram = PROGRAM;
    szParam   = PARAM;

    LongPathToQuote (szProgram, TRUE);

    LongPathToShortPath (szParam);

    szCommandLine = szProgram + " " + szParam;

    // Set up the shortcut's other properties to pass to AddFolderIcon.
    szItemName = "Notepad Example1";
    szWorkingDir = "";
    szIconPath   = "";
    nIcon        = 0;
    szShortCutKey = "";
    nFlag        = REPLACE|RUN_MAXIMIZED;

    // Add the shortcut to the submenu; create the submenu if necessary.
    nResult = AddFolderIcon (szProgramFolder, szItemName, szCommandLine,
                             szWorkingDir, szIconPath, nIcon,
                             szShortCutKey, nFlag);

    // Report the results.
    if (nResult < 0) then
        MessageBox ("AddFolderIcon failed.", SEVERE);
    else
        sprintfBox (INFORMATION, "AddFolderIcon", "%s created successfully.",
                    szItemName);
    endif
end

```

```
endif;

end;
```

## AddFolderIcon Example 3



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the AddFolderIcon function.
*
* This example places a subfolder on the desktop and an icon
* pointing to an executable in the new folder. The folder is
* a shortcut that points to an actual directory. From this
* folder the user can execute a shortcut that runs the program.
*
* Note: Before running this script, set the preprocessor
* constants so that they reference the fully qualified
* names of the Windows Notepad executable and a valid
* text file on the target system.
*
\*-----*/

#define FOLDER "C:\\Windows\\"
#define PROGRAM "C:\\Windows\\Notepad.exe"
#define PARAM "C:\\Windows\\Readme.txt"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_AddFolderIcon(HWND);

function ExFn_AddFolderIcon(hMSI)
    STRING szProgramFolder, szItemName, szCommandLine, szWorkingDir;
    STRING szIconPath, szShortCutKey;
    STRING szProgram, szParam, szFolderDir;
    NUMBER nIcon, nFlag, nResult;
begin

    // szProgramFolder is the Desktop on the local system.
    szProgramFolder = FOLDER_DESKTOP;
    szItemName = "Example folder";

    // Create the folder which the folder icon will point to.
    szFolderDir = FOLDER ^ szItemName;
    CreateDir(szFolderDir);

    // The command line for the folder icon must be the folder path, and
    // it must be enclosed in quotation marks if the path is longer than
    // eight characters.

    szCommandLine = szFolderDir;
```

```

LongPathToQuote(szCommandLine, TRUE);

szWorkingDir = "";
szIconPath   = "";
nIcon        = 0;
szShortCutKey = "";
nFlag        = REPLACE|RUN_MINIMIZED;

// Create the folder icon, and show the folder it points to.
nResult = AddFolderIcon (szProgramFolder, szItemName, szCommandLine,
                        szWorkingDir, szIconPath, nIcon, szShortCutKey,
                        nFlag);

if (nResult < 0) then
    MessageBox("AddFolderIcon failed.", SEVERE);
else
    sprintfBox (INFORMATION, "AddFolderIcon", "%s created successfully.",
                szItemName);
endif;

// Display the folder just created.
ShowProgramFolder (szFolderDir, SW_SHOW);

// Add the Example icon to the newly created folder.
szProgramFolder = szFolderDir;
szItemName      = "Notepad Example";

// Make sure the white space is not seen as a delimiter.
szProgram      = PROGRAM;
LongPathToQuote (szProgram, TRUE);

szParam = PARAM;
LongPathToShortPath (szParam);

szCommandLine = szProgram + " " + szParam;
szWorkingDir = "";
szIconPath   = "";
nResult = AddFolderIcon (szProgramFolder, szItemName, szCommandLine,
                        szWorkingDir, szIconPath, nIcon, szShortCutKey,
                        nFlag);

if (nResult < 0) then
    MessageBox ("AddFolderIcon failed.", SEVERE);
else
    sprintfBox (INFORMATION, "AddFolderIcon", "%s created successfully.",
                szItemName);
endif;

end;

```

# AddProfString

The **AddProfString** function unconditionally adds a profile string to an .ini file. Use AddProfString only to add non-unique keys, such as those found in the [386Enh] section of the System.ini file (device = ...). AddProfString adds the line KEY=VALUE to the end of the specified .ini file section. It does not replace or update an existing key. To update an existing non-unique key, call [ReplaceProfString](#). To add a unique key or to update an existing unique key's value in an .ini file, call [WriteProfString](#).

## Syntax

```
AddProfString ( szFileName, szSectionName, szKeyName, szValue );
```

## Parameters

**Table 3 • AddProfString Parameters**

Parameter	Description
<b>szFileName</b>	Specifies the name of the .ini file to which the profile string is to be added. If szFileName is unqualified (that is, if a drive designation and path are not included), InstallShield searches for the file in the Windows folder. If the file does not exist, it is created in the specified folder; if a path is not included in file name, the file is created in the Windows folder. If the file name is qualified with a path that does not exist, AddProfString fails.
<b>szSectionName</b>	Specifies the name of a section in the .ini file section. The profile string is inserted at the end of that section. If the section does not exist, InstallShield creates it. The section name should <i>not</i> be enclosed within delimiting brackets ( [ ] ). Note that the profile string is inserted even if the key specified by szKeyName already exists in that section.
<b>szKeyName</b>	Specifies the name of the key to insert. The value of this parameter will appear to the left of the equal sign in the profile string (szKeyName = szValue).
<b>szValue</b>	Specifies the value to assign to the key. The value of this parameter will appear to the right of the equal sign in the profile string (szKeyName = szValue).

## Return Values

**Table 4 • AddProfString Return Values**

Return Value	Description
<b>0</b>	AddProfString successfully added the specified profile string to the .ini file.
<b>&lt; 0</b>	AddProfString was unable to add the profile string.

## Additional Information

- AddProfString does not use the Windows API to change the .ini files. The Windows API cannot handle the types



of changes possible with AddProfString.

- Changes made to .ini files can be logged for uninstallation. However, there are some important restrictions to be aware of. For more information, see Uninstalling Initialization (.ini) File Entries.

## AddProfString Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the functions AddProfString and GetProfString.
*
* This script adds a profile string to a file; then it
* retrieves and displays the string that was added.
*
* Note: The first time you run this script, it will create a
*       file named ISExempl.ini in the root of drive C. You
*       can delete that file when you have finished analyzing
*       this script.
*
\*-----*/

#define EXAMPLE_INI "C:\\ISExempl.ini"

// The new section, key, and value to add to the file.
#define NEW_SECTION "New Section"
#define NEW_KEY      "New Key"
#define NEW_VALUE    "Test"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_AddProfString(HWND);

function ExFn_AddProfString(hMSI)
    STRING svResult;
begin
    // Add the profile string to the file.
    if (AddProfString (EXAMPLE_INI, NEW_SECTION, NEW_KEY, NEW_VALUE) != 0) then
        // Display an error message if the string could not be added.
        MessageBox ("AddProfString failed.", SEVERE);
    else
        // Retrieve the value of a key from the file.
        if (GetProfString (EXAMPLE_INI, NEW_SECTION, NEW_KEY, svResult) != 0) then
            // Display an error message if the string could not be retrieved.
            MessageBox ("GetProfString failed.", SEVERE);
        else
            // Display the key and its current value.
            MessageBox (NEW_KEY + "=" + svResult, INFORMATION);
        endif;
    endif;
end;

```

```
endif;

end;
```

## AdminAskPath



**Project** • This information applies to InstallScript MSI projects.

The **AdminAskPath** function displays a dialog that prompts the end user to enter the path to a destination location for an administrative installation (when the end user runs an InstallScript MSI project **Setup.exe** with the **/a** argument).

### Syntax

```
AdminAskPath ( szMsg, szDefaultPath, svResultPath );
```

### Parameters

**Table 5** • AdminAskPath Parameters

Parameter	Description
<b>szMsg</b>	Specifies the message to display in this dialog. To display the default instructions for this dialog, pass a null string ("" ) in this parameter.
<b>szDefaultPath</b>	Specifies the default path to display in the edit field. The end user can modify this string. The default implementation of OnAdminInstallUIBefore passes INSTALLDIR in this parameter.
<b>svResultPath</b>	Returns the resulting path, regardless of whether the user accepts the default path, modifies it, or selects an alternate path from the Choose Folder dialog. The default implementation of OnAdminInstallUIBefore passes INSTALLDIR in this parameter.

### Return Values

**Table 6** • AdminAskPath Return Values

Return Value	Description
<b>NEXT (1)</b>	Indicates that the user clicked the Next button.
<b>BACK (12)</b>	Indicates that the user clicked the Back button.

### Additional Information

**AdminAskPath** uses the **AskPath** dialog, and it uses the same dialog resources as **AskPath**. Therefore, any changes you make to the layout of **AskPath** in the Dialog Editor are reflected in **AdminAskPath**.

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## AdminAskPath Example



**Project** • This information applies to InstallScript MSI projects.

```
//-----
//
// InstallShield Example Script
//
// Demonstrates the AdminAskPath function.
//
// In the OnAdminInstallUIBefore event of an InstallScript MSI project,
// prompt the user for the target directory into which files should be
// uncompressed and copied.
//
//-----

function OnAdminInstallUIBefore( )
    int nResult;

begin
    Dlg_SdWelcome:
        SdWelcome("", "");

    Dlg_AdminAskPath:
        // prompt the user for the target path, storing it in INSTALLDIR
        nResult = AdminAskPath("", INSTALLDIR, INSTALLDIR);
        if (nResult = BACK) goto Dlg_SdWelcome;

        // prepare the status dialog
        SetStatusExStaticText(SdLoadString(IDS_IFX_STATUSEX_STATICTEXT_FIRSTUI));
        Enable(STATUSEX);
end;
```

## AskDestPath



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **AskDestPath** function displays a dialog that enables the end user to specify a destination folder for the files to be installed by your installation. The dialog also includes a Browse button that enables the end user to select an existing folder on the system.

To open the Choose Folder dialog from the Choose Destination Location dialog, the end user must click the Browse button. The Choose Folder dialog displays a list of all available folders. The end user can select an existing folder or enter a new folder name. If the end user enters the name of a folder that does not exist, a message box opens to enable the end user to create the folder.



---

**Note** • Installations that run in silent mode should create the new folder if it does not exist before calling *AskDestPath*. This ensures that the confirmation dialog is not displayed. Without this step, two response files are required to handle the two possible conditions.


The folder selected by the end user must be writable; non-writable folders are not accepted. If you want the end user to be able to select folders that are not writable, call the *AskPath* function instead.

## Syntax

```
AskDestPath ( szTitle, szMsg, svDir, nReserved );
```

## Parameters

**Table 7 • AskDestPath Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title Choose Destination Location, pass a null string ("" ) in this parameter.
<b>szMsg</b>	Specifies the message to display in the dialog. To pass multiple lines of static text in this parameter, insert newline escape sequences ( \n ) where needed to break the line. To display the default instructions for this dialog, pass a null string ("" ) in this parameter.
<b>svDir</b>	Specifies the default path to display when the dialog is opened; returns the path to the folder selected by the end user.   <b>Note •</b> If the default folder specified by svDir does not already exist on the end user's system, it is not created unless the end user clicks the Browse button and follows the steps to create it from the Choose Folder dialog. Therefore, whenever you specify a default folder that you intend to use before calling <a href="#">FeatureMoveData</a> (which creates the folder if necessary), you must call <a href="#">ExistsDir</a> when AskDestPath returns in order to determine whether that folder exists. If it does not exist, call <a href="#">CreateDir</a> to create it on the end user's system. Note that <a href="#">FeatureTransferData</a> is called automatically in an installation running an event-based script.
<b>nReserved</b>	The value of this parameter must be 0 (zero).

## Return Values

**Table 8 • AskDestPath Return Values**

Return Value	Description
<b>NEXT (1)</b>	Indicates that the end user clicked the Next button.
<b>BACK (12)</b>	Indicates that the end user clicked the Back button.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## AskDestPath Example



**Project •** This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the AskDestPath function.
 *
 * This script calls AskDestPath to get the path to the location
 * where the installation will install files. That path is then
 * displayed in a message box.
 *
\*-----*/

#define TITLE_TEXT "AskDestPath Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_AskDestPath(HWND);

function ExFn_AskDestPath(hMSI)
    STRING szTitle, szMsg, svDir;
    NUMBER nReturn;
begin

    // Set a default path for the installation.
    svDir = INSTALLDIR;

    // Get a destination directory. Pass a null string in the
    // second parameter to display the default message.
    nReturn = AskDestPath (TITLE_TEXT, "", svDir, 0);

    if (nReturn < 0) then
        // Report the error.
        MessageBox ("AskDestPath failed.", SEVERE);
    elseif (nReturn = NEXT) then
        // Display the selected destination directory name.
        MessageBox ("You selected " + svDir + ".", INFORMATION);
    endif;

end;

```

## AskOptions



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **AskOptions** function formats and displays a dialog that prompts the end user to select one or more options. The dialog displays up to nine selection controls—either check boxes or option buttons—depending on the value of `nValue`.

The default title for this dialog is Select Features. To change the contents of the title bar, call [SetDialogTitle](#) before calling AskOptions.



---


**Note** • You cannot use the [PlaceWindow](#) function in conjunction with the AskOptions function. By default, the dialog appears in the center of the desktop, unless the background window mode is enabled. If the installation is in window mode, the dialog appears in the center of the background window.

## Syntax

```
AskOptions ( nValue, szMsg, szText1, bvCheck1, szText2, bvCheck2[, szTextn, bvCheckn] [,..., ...] );
```


## Parameters

**Table 9 • AskOptions Parameters**

Parameter	Description
<b>nValue</b>	<p>Specifies the type of controls to display. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>● <b>EXCLUSIVE</b>—Specifies option buttons, which enable the end user to select only one option.</li> <li>● <b>NONEXCLUSIVE</b>—Specifies check boxes, which enable the end user to select more than one option.</li> </ul>
<b>szMsg</b>	<p>Specifies the message to display in the dialog. You can use this message to describe the options and/or ask the user to choose one or more options. If the message is too long for one line, use newline escape sequences ( \n ) to insert line breaks.</p>  <p><b>Note •</b> Your operating system dictates the maximum message string length. By default, the display of szMsg text is limited to two lines by the AskOptions dialog resource in _Isres.dll. To display more than two lines of szMsg text, you can create a custom dialog from the AskOptions dialog.</p>
<b>szText1</b>	<p>Specifies a text label to display next to the first check box or option button. The maximum number of characters that can be displayed depends on the font you use; be sure to check that the string specified fits in the static text field of the dialog. If the string does not fit, either shorten it or call <a href="#">SdAskOptions</a> instead.</p> <p>To create an accelerator key, insert an ampersand (&amp;) before the character you want to designate for that purpose. The character is displayed with an underline to indicate its function. For example, to make Alt + C the accelerator key for Custom, pass "&amp;Custom"; to make Alt + S the accelerator key for Custom, pass "Cu&amp;stom."</p>
<b>bvCheck1</b>	<p>Specifies the initial status of the first check box or option button when the dialog is opened; returns the status of the first check box or option button when the dialog is closed. The following constants are passed and returned in this parameter:</p> <ul style="list-style-type: none"> <li>● <b>TRUE</b>—The first check box or option button is selected.</li> <li>● <b>FALSE</b>—The first check box or option button is not selected.</li> </ul>
<b>szText2</b>	<p>Specifies a text label to display next to the second check box or option button. The maximum number of characters that can be displayed depends on the font you use; be sure to check that the string specified fits in the static text field of the dialog. If the string does not fit, either shorten it or call <a href="#">SdAskOptions</a> instead.</p> <p>Create an accelerator in the same manner as you did for szText1.</p>



**Table 9 • AskOptions Parameters (cont.)**

Parameter	Description
<b>bvCheck2</b>	<p>Specifies the initial status of the second check box or option button when the dialog is opened; returns the status of the second check box or option button when the dialog is closed. The following constants are passed and returned in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>TRUE</b>—The first check box or option button is selected.</li> <li>• <b>FALSE</b>—The first check box or option button is not selected.</li> </ul> <p>Up to seven additional options can be defined. Each additional option is indicated by a pair of parameters—a string parameter that defines a label and a number variable that indicates the state of the option when AskOptions returns. To set the initial state of an option, assign TRUE or FALSE to the number variable before calling AskOptions.</p> <p></p> <p><b>Note</b> • If <i>nValue</i> is <i>EXCLUSIVE</i> and the initial state of more than one option is set to <i>TRUE</i>, AskOptions preselects the first option in the parameter list that is set to <i>TRUE</i>.</p>

## Return Values

**Table 10 • AskOptions Return Values**

Return Value	Description
<b>NEXT (1)</b>	Indicates that the end user clicked the Next button. The states of the controls are returned in the individual bvCheck variables.
<b>BACK (12)</b>	Indicates that the end user clicked the Back button. The states of the controls are returned in the individual bvCheck variables.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## AskOptions Example



**Project** • This information applies to the following project types:

- InstallScript
- InstallScript MSI

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the AskOptions function.
```

```

*
* The AskOptions dialog is displayed twice. First, it is
* displayed with check boxes; then it is displayed with option
* buttons. The example shows the maximum number of options
* allowed--nine.
*
\*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_AskOptions(HWND);

function ExFn_AskOptions(hMSI)
    STRING szMsg, szText1, szText2, szText3, szText4, szText5;
    STRING szText6, szText7, szText8, szText9;
    NUMBER nReturn, nValue, nvCheck1, nvCheck2, nvCheck3, nvCheck4;
    NUMBER nvCheck5, nvCheck6, nvCheck7, nvCheck8, nvCheck9;
begin

    szMsg = "Select from the options below.";

    szText1 = "Option 1";
    szText2 = "Option 2";
    szText3 = "Option 3";
    szText4 = "Option 4";
    szText5 = "Option 5";
    szText6 = "Option 6";
    szText7 = "Option 7";
    szText8 = "Option 8";
    szText9 = "Option 9";

    nvCheck1 = TRUE;
    nvCheck2 = FALSE;
    nvCheck3 = FALSE;
    nvCheck4 = FALSE;
    nvCheck5 = FALSE;
    nvCheck6 = FALSE;
    nvCheck7 = FALSE;
    nvCheck8 = FALSE;
    nvCheck9 = FALSE;

    // Display the check box (NONEXCLUSIVE) dialog.
    nValue = NONEXCLUSIVE;

    AskOptions (nValue, szMsg,
                szText1, nvCheck1,
                szText2, nvCheck2,
                szText3, nvCheck3,
                szText4, nvCheck4,
                szText5, nvCheck5,
                szText6, nvCheck6,
                szText7, nvCheck7,
                szText8, nvCheck8,
                szText9, nvCheck9);

    // Display the option button (EXCLUSIVE) dialog.
    nValue = EXCLUSIVE;
    AskOptions (nValue, szMsg,

```

```

        szText1, nvCheck1,
        szText2, nvCheck2,
        szText3, nvCheck3,
        szText4, nvCheck4,
        szText5, nvCheck5,
        szText6, nvCheck6,
        szText7, nvCheck7,
        szText8, nvCheck8,
        szText9, nvCheck9);

end;

```

## AskPath



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **AskPath** function displays a dialog that prompts the end user to enter the path to a destination location. The dialog contains a single-line edit field in which you can display a default path. The end user has three options:

- Accept the default path.
- Edit the default path.
- Display the Choose Folder dialog to select a folder.

To open the Choose Folder dialog, the end user must click the Browse button. The Choose Folder dialog displays a list of all available folders. The end user can select an existing folder or enter a new folder name. If the end user enters the name of a folder that does not exist, the folder is created.



**Caution** • *AskPath* does not verify the existence of the path entered by the end user. After calling *AskPath*, call *CreateDir* to create the path.



**Note** • You cannot use the *PlaceWindow* function in conjunction with the *AskPath* function. By default, the dialog opens in the center of the desktop, unless the background window mode is enabled. If the installation is in window mode, the dialog opens in the center of the background window.

The default title for the dialog is *Choose Destination Location*. To change the title, call *SetDialogTitle* before calling *AskPath*.

The *AskPath* function accepts the name of a folder that exists but is not writable. To limit the end user's selection to writable folders, call the *AskDestPath* function instead.


### Syntax

```
AskPath ( szMsg, szDefPath, svResultPath );
```

## Parameters

**Table 11 • AskPath Parameters**

Parameter	Description
<b>szMsg</b>	Specifies the message to display in this dialog. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>szDefPath</b>	Specifies the default path to display in the edit field. The end user can modify this string.
<b>svResultPath</b>	Returns the resulting path, regardless of whether the user accepts the default path, modifies it, or selects an alternate path from the Choose Folder dialog. AskPath adds a backslash to the end of the path before placing it into svResultPath. If necessary, that backslash can be removed by calling <a href="#">StrRemoveLastSlash</a> after AskPath returns. If the user clicks the Back button, the value of svResultPath will be unpredictable. Therefore, if you are using the same variable for both szDefPath and svResultPath, be sure to reinitialize that variable when the return value from AskPath is BACK.



**Note •** The edit field displayed in the dialog scrolls to accommodate long strings. Because the number of characters that can be entered into the edit field is not limited, you should declare the variable passed in svResultPath without an explicit size. If the string variable is not large enough to store the text entered by the user, the string is truncated and an error message is displayed. Because this function appends a backslash and a NULL terminator to the end of the string, the size of the string must be at least two characters longer than the path entered by the user.

## Return Values

**Table 12 • AskPath Return Values**

Return Value	Description
<b>NEXT (1)</b>	Indicates that the end user clicked the Next button.
<b>BACK (12)</b>	Indicates that the end user clicked the Back button; svResultPath is set to a null string ("").

## Additional Information

**AdminAskPath** uses the **AskPath** dialog, and it uses the same dialog resources as **AskPath**. Therefore, any changes you make to the layout of **AskPath** in the Dialog Editor are reflected in **AdminAskPath**.

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## AskPath Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the AskPath function.
 *
 * This script obtains the path to a folder on the
 * end user's computer. If the path does not exist, it creates
 * a folder at that location if indicated by the
 * end user. Finally, it displays the selected path.
 *
 \*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_AskPath(HWND);

function ExFn_AskPath(hMSI)
    STRING szMsg, svResultPath[101];
    BOOL bTargetDirOk;
begin

    // Disable the Back button in installation dialogs.
    Disable (BACKBUTTON);

    // Create the message to display in the AskPath dialog.
    szMsg = "Specify a folder for the application.";

    // Initialize valid path indicator.
    bTargetDirOk = FALSE;

    repeat
        // Get a path from the user. The default path is
        // the current value of the system variable INSTALLDIR.
        if (AskPath (szMsg, INSTALLDIR, svResultPath) = NEXT) then
            // Does the path entered by the user exist on the
            // target system?
            if (ExistsDir (svResultPath) = 0) then
                // If it exists, set indicator to exit the loop.
                bTargetDirOk = TRUE;
            else
                // If the path doesn't exist, ask if it should be created.
                if (AskYesNo ("Folder does not exist. Create it?",YES) = YES) then
                    // Attempt to create the folder (directory).
                    if (CreateDir (svResultPath) = 0) then
                        // If the folder was created, set indicator to exit the loop.
                        bTargetDirOk = TRUE;
                    else

```

```

        // Inform the end user that the folder was not created.
        MessageBox ("Unable to create " + svResultPath, WARNING);
    endif;
endif;
endif;
endif;
until bTargetDirOk;

// Display the name of the target folder.
MessageBox ("The target folder is " + svResultPath, INFORMATION);

// You'd also enable the Back button for subsequent dialogs.
Enable (BACKBUTTON);

end;

```

## AskText



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **AskText** function displays a dialog that contains one static text field and one edit box. Specify default text for the static text field in the parameter `szQuestion`. Specify default text for the edit box in the parameter `szDefault`.



**Note** • You cannot use the [PlaceWindow](#) function in conjunction with the *AskText* function. By default, the dialog appears in the center of the desktop, unless the background window mode is enabled. If the installation is in window mode, the dialog opens in the center of the background window.


The default title of this dialog is *Enter Information*. To change the contents of the title bar, call [SetDialogTitle](#) before calling *AskText*.

### Syntax

```
AskText ( szQuestion, szDefault, svResult );
```

## Parameters

**Table 13 • AskText Parameters**

Parameter	Description
<b>szQuestion</b>	Specifies the question or statement to display. If the length of the string in this parameter exceeds the width of the static text field, one or more line breaks will be inserted into the string so that it displays on multiple lines in the dialog. If you prefer, you can format the string manually by inserting one or more newline escape sequences ( \n ) into it. This parameter does not have a default value.
<b>szDefault</b>	Specifies the default text for the edit box. The edit box scrolls, if necessary, to accommodate a long string.
<b>svResult</b>	Returns the text entered by the end user when the Next button is clicked to close the dialog. If the user clicks the Back button, the value of svResult will be unpredictable. Therefore, if you are using the same variable for both szDefault and svResult, be sure to reinitialize that variable when the return value from AskText is BACK.
	 <p><b>Note</b> • The string variable that you pass in svResult must be large enough to accommodate the text entered into the edit box. For that reason, you should use the autosize method to declare the variable.</p>

## Return Values

**Table 14 • AskText Return Values**

Return Value	Description
<b>NEXT (1)</b>	Indicates that the end user clicked the Next button.
<b>BACK (12)</b>	Indicates that the end user clicked the Back button.

## Additional Information

The dialog that is displayed by the AskText function cannot be displayed with a skin; it appears the same regardless of whether you have specified a skin.

## AskText Example



**Project** • This information applies to the following project types:

- InstallScript
- InstallScript MSI

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the AskText function.
 *
 * This script gets a company name from the end user.
 *
\*-----*/

#define MSG_TEXT      "Please enter your company name."
#define DEFAULT_COMPANY "My Software Company"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_AskText(HWND);

function ExFn_AskText(hMSI)
    STRING svCompany, szTitle;
    NUMBER nResult;
begin
    // Get the company name.
    nResult = AskText (MSG_TEXT, DEFAULT_COMPANY, svCompany);

    if nResult = NEXT then
        // Display the company name that was entered by the user.
        MessageBox ("Company: " + svCompany, INFORMATION);
    endif;

end;

```

## AskYesNo



**Project** • This information applies to the following project types:

- Basic MSI
- InstallScript
- InstallScript MSI

The **AskYesNo** function presents a message box that displays a question that the end user can answer by clicking a Yes or No button. The **AskYesNo** message box contains four items:

- Question mark icon
- Question text
- Yes button
- No button





**Note** • The default title is **Question**. To change the contents of the title bar, call [SetDialogTitle](#) before calling **AskYesNo**.

The **AskYesNo** message box is created by a direct call to the corresponding Windows API function, which displays a system modal dialog. Once a modal dialog is displayed, it retains focus until the end user closes it.

Windows displays this dialog; therefore, the installation cannot change the text of the buttons on the dialog. Windows displays the button text—"Yes" and "No" on English-based systems—in the language of the operating system; no manual localization of this text is required. If you need to display a more flexible dialog, call a Windows API function directly or use a custom dialog.

## Syntax

```
AskYesNo ( szQuestion, nDefault );
```

## Parameters

**Table 15** • AskYesNo Parameters

Parameter	Description
<b>szQuestion</b>	Specifies the question to display in the message box. If the message is too large to fit on one line, embed newline escape characters ( \n ) in the message to insert line breaks.
<b>nDefault</b>	Specifies the button that is selected by default. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li><b>YES</b>—The Yes button is highlighted when the dialog opens.</li> <li><b>NO</b>—The No button is highlighted when the dialog opens.</li> </ul>

## Return Values

**Table 16** • AskYesNo Return Values

Return Value	Description
<b>YES (1)</b>	Indicates that the user clicked the Yes button.
<b>NO (0)</b>	Indicates that the user clicked the No button.

## Additional Information

The dialog that is displayed by the AskYesNo function cannot be displayed with a skin; it appears the same regardless of whether you have specified a skin.

## AskYesNo Example



**Project** • This information applies to the following project types:

- Basic MSI
- InstallScript
- InstallScript MSI



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the AskYesNo function.
*
* This script asks the user whether or not to display the
* ReadMe file. If yes, the script launches the Windows
* Notepad to open a ReadMe file.
*
* Note: Before running this script, set the preprocessor
*       constants so that they reference the fully qualified
*       names of the Windows Notepad executable and a valid
*       text file on the target system.
*
\*-----*/

#define PROGRAM "C:\\Windows\\Notepad.exe"
#define PARAM   "C:\\Windows\\Readme.txt"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_AskYesNo(HWND);

function ExFn_AskYesNo(hMSI)
begin
    // Display the AskYesNo dialog. The default is set to Yes.
    if (AskYesNo("Installation complete. Would you like to read the Readme " +
        "file now?", YES) = YES) then
        LaunchApp(PROGRAM, PARAM);
    endif;
end;
```

# BatchAdd

The **BatchAdd** function inserts a SET command or other DOS command into a batch file that has been loaded into memory with [BatchFileLoad](#). The parameter `nOptions` allows you to add the new command as the first or last statement in the file, replace an existing statement with the new command, or specify that the new command be added before or after an existing statement.

Before calling `BatchAdd`, you must call `BatchFileLoad` to load the file to be modified into memory. After you modify the file, call [BatchFileSave](#) to save it to disk.


Do not mix the Ez batch file functions with the advanced batch file functions. After calling `BatchFileLoad`, you cannot use Ez batch file functions until you have called `BatchFileSave` to save the file.

## Syntax


```
BatchAdd ( szKey, szValue, szRefKey, nOptions );
```

## Parameters

**Table 17 • BatchAdd Parameters**

Parameter	Description
<b>szKey</b>	Specifies the keyword to add to the batch file. PATH, TEMP, and MYENV are examples of valid keys for this parameter.
<b>szValue</b>	<p>Specifies the value of the key to be added to the batch file. This string must be no longer than 512 bytes; passing a string longer than 512 bytes will cause an installation error. To add a longer string, use the <a href="#">FileGrep</a> and <a href="#">FileInsertLine</a> functions.</p> <div>  <p><b>Caution •</b> Batch files do not support long paths completely. If you are using this function to add a line that contains a long path, call <a href="#">LongPathToShortPath</a> to convert the long path to its short path equivalent before adding it to the string to be placed in the batch file. For information on long paths and long file names, refer to <i>Long File Name Format</i>.</p> </div>
<b>szRefKey</b>	Specifies the reference key relative to which you are adding szKey in the batch file.

**Table 17 • BatchAdd Parameters (cont.)**

Parameter	Description
<b>nOptions</b>	<p>Specifies where in the file to insert the line. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>BEFORE</b>—The statement is added before the first line that contains szRefKey. If szRefKey is a null string (""), the statement is added as the first line of the file.</li> <li>• <b>AFTER</b>—The statement is added after the last line that contains szRefKey. If szRefKey is a null string (""), the statement is added as the last line of the file.</li> <li>• <b>REPLACE</b>—The statement replaces an existing line in the file. If multiple lines with same key exist, only the last line is replaced. If szKey does not exist in the file, a new line will be added after szRefKey. If szRefKey is a null string (""), the new line is added as the last line of the file.</li> </ul> <p>When the statement to be added is not a SET command, pass a null string ("") in szKey, pass the complete command in szValue, and use the OR operator to combine the constant COMMAND with one of the other option constants, as shown below:</p> <pre>BatchAdd("", "PAUSE", "", COMMAND   AFTER);</pre> <div>  <p><b>Note</b> • BatchAdd automatically adds the DOS keyword SET to the beginning of the statement to be inserted unless you use the OR operator to combine the constant COMMAND with the value you pass in nOptions. If you do not explicitly specify REPLACE in nOptions, the specified statement is added even if a duplicate line exists in the batch file.</p> </div>

## Return Values

**Table 18 • BatchAdd Return Values**

Return Value	Description
0	BatchAdd successfully added a SET statement or other command to the batch file.
< 0	BatchAdd was unable to add the SET statement or other command to the batch file.

## Additional Information

An InstallScript reference key is either an environment variable, a DOS command, or a program file name. Environment variables are keywords such as PATH, COMSPEC, LIB, or other predefined or user-defined identifiers. The value of an environment variable is established by using the DOS SET command. Statements that appear in a batch file must be either DOS commands, program names (with or without command-line parameters), or comments. Refer to your operating system manual for a detailed definition of commands and environment variables.

## BatchAdd Example

The following example applies to:

### InstallScript/InstallScript MSI Installations

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the BatchAdd function.
*
* This example script adds three statements to a batch file.
* First, it adds a PATH statement. Next, it adds a command to
* set the environment variable EXENV. Then it adds a command
* to launch SHARE.EXE, placing it before the existing command
* to start Windows. Finally, it backs up the original file
* and saves the edited file under its original name.
*
* If any of the calls to BatchAdd fails, the setup exits
* without saving changes to the batch file.
*
* Note: Before running this script, create a batch file
*       named ISExempl.bat in the root of drive C. For
*       best effect, that file should include the following
*       lines:
*
*       PATH=C:\Windows
*       Win
*
\*-----*/

#define EXAMPLE_BAT "C:\\ISEXAMPL.BAT"
```

```

#define EXAMPLE_BAK "ISEXAMPL.BAK"

    STRING szPath;

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"
    function OnBegin()
    begin

        // Load the batch file to be edited.
        if (BatchFileLoad (EXAMPLE_BAT) < 0) then
            MessageBox ("Unable to load " + EXAMPLE_BAT+".", SEVERE);
            abort;
        endif;

        // Add a SET PATH command that appends the value of an existing
        // search path to C:\EXAPP\BIN.
        szPath = "C:\\EXAPP\\BIN;%PATH%";

        if (BatchAdd ("PATH", szPath, "PATH", AFTER) < 0) then
            MessageBox ("First call to BatchAdd failed", WARNING);
            abort;
        endif;

        // Add the line SET EXENV = C:\OTHERAPP\BIN. If the
        // environment variable EXENV already exists in the batch
        // file, the last SET EXENV statement is replaced.
        szPath = "C:\\OTHERAPP\\BIN";

        if (BatchAdd ("EXENV", szPath, "EXENV", REPLACE) < 0) then
            MessageBox ("Second call to BatchAdd failed", WARNING);
            abort;
        endif;

        // Add the command SHARE.EXE before the command WIN.
        if (BatchAdd ("", "SHARE.EXE", "WIN", BEFORE | COMMAND) < 0) then
            MessageBox ("Third call to BatchAdd failed", WARNING);
            abort;
        endif;

        // Save the updated file; back up the original file.
        if (BatchFileSave(EXAMPLE_BAK) < 0) then
            MessageBox ("Unable to save " + EXAMPLE_BAK + ".", SEVERE);
        else
            MessageBox ("Batch file saved. Backup created.", INFORMATION);
        endif;

    end;

```

The following example applies to:

## Basic MSI Installations



**Tip** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the BatchAdd function.
*
* This example script adds three statements to a batch file.
* First, it adds a PATH statement. Next, it adds a command to
* set the environment variable EXENV. Then it adds a command
* to launch SHARE.EXE, placing it before the existing command
* to start Windows. Finally, it backs up the original file
* and saves the edited file under its original name.
*
* If any of the calls to BatchAdd fails, the setup exits
* without saving changes to the batch file.
*
* Note: Before running this script, create a batch file
*       named ISEXAMPL.bat in the root of drive C. For
*       best effect, that file should include the following
*       lines:
*
*       PATH=C:\Windows
*       Win
*
\*-----*/

#define EXAMPLE_BAT "C:\\ISEXAMPL.BAT"
#define EXAMPLE_BAK "ISEXAMPL.BAK"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_BatchAdd(HWND);

function ExFn_BatchAdd(hMSI)
    STRING szPath;
begin

    // Load the batch file to be edited.
    if (BatchFileLoad (EXAMPLE_BAT) < 0) then
        MessageBox ("Unable to load " + EXAMPLE_BAT+".", SEVERE);
        abort;
    endif;

    // Add a SET PATH command that appends the value of an existing
    // search path to C:\\EXAPP\\BIN.
    szPath = "C:\\EXAPP\\BIN;%PATH%";

    if (BatchAdd ("PATH", szPath, "PATH", AFTER) < 0) then
        MessageBox ("First call to BatchAdd failed", WARNING);
        abort;
    endif;

    // Add the line SET EXENV = C:\\OTHERAPP\\BIN. If the
    // environment variable EXENV already exists in the batch
    // file, the last SET EXENV statement is replaced.
    szPath = "C:\\OTHERAPP\\BIN";

    if (BatchAdd ("EXENV", szPath, "EXENV", REPLACE) < 0) then

```



```

        MessageBox ("Second call to BatchAdd failed", WARNING);
        abort;
    endif;

    // Add the command SHARE.EXE before the command WIN.
    if (BatchAdd ("", "SHARE.EXE", "WIN", BEFORE | COMMAND) < 0) then
        MessageBox ("Third call to BatchAdd failed", WARNING);
        abort;
    endif;

    // Save the updated file; back up the original file.
    if (BatchFileSave(EXAMPLE_BAK) < 0) then
        MessageBox ("Unable to save " + EXAMPLE_BAK + ".", SEVERE);
    else
        MessageBox ("Batch file saved. Backup created.", INFORMATION);
    endif;

end;

```

## BatchDeleteEx

The **BatchDeleteEx** function deletes lines in a batch file that contain the value specified in szKey.




---

**Note** • Before calling *BatchDeleteEx*, you must call *BatchFileLoad* to load the file to be modified into memory. After you modify the file, call *BatchFileSave* to save it to disk.

Do not mix the Ez batch file functions with the advanced batch file functions. After calling *BatchFileLoad*, you cannot use Ez batch file functions until you have called *BatchFileSave* to save the file.

### Syntax

```
BatchDeleteEx ( szKey, nOptions );
```

## Parameters

**Table 19 • BatchDeleteEx Parameters**

Parameter	Description
<b>szKey</b>	Specifies the reference keyword that identifies the line or lines to be deleted.
<b>nOptions</b>	<p>Indicates whether szKey specifies an environment variable in a SET statement or a command. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li><b>0</b>—Specifies that szKey is an environment variable in a SET statement. An environment variable is either a predefined identifier (such as PATH, COMSPEC, and LIB), or a user-defined identifier. For example, the following statement would be deleted if the value of szKey was "LIBPATH" and nOptions was set to 0:  SET LIBPATH=C:\Lang\Lib</li> <li><b>COMMAND</b>—Specifies that szKey is either a DOS command or a program file name.</li> </ul>

## Return Values

**Table 20 • BatchDeleteEx Return Values**

Return Value	Description
<b>0</b>	BatchDeleteEx successfully deleted lines containing the specified value.
<b>&lt; 0</b>	BatchFileLoad was unable to delete lines containing the specified value.

## BatchDeleteEx Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the BatchDeleteEx function.
*
```

```

* This example script deletes lines from a batch file. First,
* it calls BatchFileLoad to load the file. Next, it deletes
* all lines with a PATH command. Then it deletes all lines
* that reference MyApp.exe (for example, C:\MyApps\MyApp.exe).
* Finally, it backs up the original file and saves the
* edited file under its original name.
*
* Note: Before running this script, create a batch file
*       named ISExempl.bat in the root of drive C. For
*       best effect, that file should include the
*       following lines:
*
*       SET PATH=C:\Windows
*       C:\MyApps\MyApp.exe
*
\*-----*/

#define EXAMPLE_BAT "C:\\ISExempl.bat"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_BatchDeleteEx(HWND);

function ExFn_BatchDeleteEx(hMSI)
    STRING szBackupFile;
begin

    // Load or create the batch file to be edited.
    if (BatchFileLoad (EXAMPLE_BAT) < 0) then
        MessageBox ("Unable to load " + EXAMPLE_BAT + ".", SEVERE);
        abort;
    endif;

    // Delete all SET PATH= commands.
    BatchDeleteEx ("PATH", 0);

    // Delete all lines with references to MyApp.exe.
    BatchDeleteEx ("MyApp.exe", COMMAND);

    // Save the edited batch file.
    if (BatchFileSave("Example.bak") < 0) then
        MessageBox ("Unable to save " + EXAMPLE_BAT + ".", SEVERE);
    else
        MessageBox ("Batch file saved.", INFORMATION);
    endif;

end;

```

## BatchFileLoad

The **BatchFileLoad** function loads a copy of the specified batch file into memory so that other advanced batch file functions can be called to operate on the file. Specify the name of the batch file you want to edit in `szBatchFile` or pass a null string ("" ) in `szBatchFile` to edit the default batch file, which is set initially by InstallShield to the `bootup Autoexec.bat` file used by the system.

Note that you can call `BatchFileLoad` to create a new batch file. To do so, pass in `szBatchFile` the name of a file that does not exist. Then call other batch functions to edit the new file. Finally, call [BatchFileSave](#) to save the new file to disk.



---

**Note** • Before using any of the advanced batch file functions, you must call `BatchFileLoad` to load the file to be modified into memory. After you modify the file, call `BatchFileSave` to save it to disk. To obtain the fully qualified file name of the batch file that will be used by default in the installation script, call [BatchGetFileName](#). To specify a different batch file to be used by default in the installation script, call [BatchSetFileName](#).

Do not mix the Ez batch file functions with the advanced batch file functions. After calling `BatchFileLoad`, you cannot use Ez batch file functions until you have called `BatchFileSave` to save the file.

## Syntax

```
BatchFileLoad ( szBatchFile );
```

## Parameters

**Table 21 • BatchFileLoad Parameters**

Parameter	Description
<b>szBatchFile</b>	<p>Specifies the fully qualified name of the batch file to load into memory. To load the current default batch file, pass a null string (""). If you specify a file in this parameter, that file becomes the default batch file. After calling this function, you can use all of the advanced batch file functions to manipulate this file.</p> <p>To create a new batch file with BatchFileLoad, pass in szBatchFile the name of a file that does not exist. Then call other batch functions to edit the new file.</p>

## Return Values

**Table 22 • BatchFileLoad Return Values**

Return Value	Description
<b>0</b>	BatchFileLoad successfully initialized the batch file buffer. If szConfigFile specified an existing batch file, the file was loaded into the buffer; otherwise, an empty buffer was created.
<b>&lt; 0</b>	BatchFileLoad was unable to initialize the batch file buffer.

## BatchFileLoad Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the BatchFileLoad and BatchFileSave functions.
*
* This example script shows how to open a batch file for
* editing, how to create a backup of the original file, and
* how to save and close the edited file.
*
* To demonstrate how the file backup feature of BatchFileSave
* prevents the overwriting of existing files, this script loads
```

```

* and saves two different batch files. The first batch file is
* backed up with a specific file name. The second is backed up
* with a wildcard extension so that BatchFileSave will
* generate a unique file extension consisting of three digits.
*
* Note: Before running this script, create two batch files
*       (ISExamp1.bat and ISExamp2.bat) in the root of drive C.
*       For best effect, you should delete or move any other
*       files named ISExamp1.* or ISExamp2.*.
*
\*-----*/

// Names of batch files and backup files used in this example.
#define EXAMPLE1 "ISExamp1"
#define EXAMPLE2 "ISExamp2"

// Full names of batch files.
#define EXAMPLE1_BAT "C:\\\" + EXAMPLE1 + ".bat"
#define EXAMPLE2_BAT "C:\\\" + EXAMPLE2 + ".bat"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_BatchFileLoad(HWND);

function ExFn_BatchFileLoad(hMSI)
begin
    // Load EXAMPLE1_BAT.
    if (BatchFileLoad (EXAMPLE1_BAT) < 0) then
        MessageBox ("Unable to load " + EXAMPLE1_BAT + ".", SEVERE);
        abort;
    endif;

    // Use other batch file functions here to edit the first file.
    // Back up the original file with the extension "bak"; save the
    // edited file under its original name. If ISExamp1.bak already
    // exists, BatchFileSave will generate a numbered extension.
    if (BatchFileSave (EXAMPLE1 + ".bak") < 0) then
        MessageBox ("Unable to save " + EXAMPLE1_BAT + ".", SEVERE);
        abort;
    else
        MessageBox (EXAMPLE1_BAT + " saved.", INFORMATION);
    endif;

    // Load EXAMPLE2_BAT.
    if (BatchFileLoad (EXAMPLE2_BAT) < 0) then
        MessageBox ("Unable to load " + EXAMPLE2_BAT + ".", SEVERE);
        abort;
    endif;

    // Use other batch file functions here to edit the second file.
    // Back up the original batch file with a numbered extension
    // and save the edited file under its original name.
    if (BatchFileSave (EXAMPLE2 + ".*") < 0) then
        MessageBox ("Unable to save " + EXAMPLE2_BAT + ".", SEVERE);
        abort;
    else
        MessageBox (EXAMPLE2_BAT + " saved.", INFORMATION);
    endif;
end

```

```
endif;

end;
```

## BatchFileSave

The **BatchFileSave** function saves to disk a batch file that has been loaded into memory with the function [BatchFileLoad](#). The file is saved under its original name. If a file name is specified in `szBackupFile`, the original file is renamed with that file name before the edited file is written to disk. If `szBackupFile` contains a null string (""), the original file is replaced with the modified file. If you do not call `BatchFileSave` when you are finished modifying a batch file with advanced batch file functions, all modifications will be lost.




---


**Note** • Do not mix the Ez batch file functions with the advanced batch file functions. After calling *BatchFileLoad*, you cannot use Ez batch file functions until you have called *BatchFileSave* to save the file.

### Syntax

```
BatchFileSave ( szBackupFile );
```

## Parameters

**Table 23 • BatchFileSave Parameters**

Parameter	Description
<b>szBackupFile</b>	<p>Specifies whether a backup copy of the original file as it existed before editing should be saved.</p> <ul style="list-style-type: none"> <li>• If no backup file should be created, specify a null string in this parameter.</li> <li>• If the original file should be backed up with a specific name, pass that file name in this parameter. The file name must be unqualified (that is, do not specify a drive and/or path). Note that if a file with the specified name already exists, BatchFileSave will generate a unique file extension, as described in the next bullet item.</li> <li>• If the original file should be backed up with an installation-generated file extension, specify the wildcard character "*" as the file extension (for example, "Batch."). The installation will then assign a numeric value, starting at 001, as the extension. If a file already exists with that extension, the extension's value will be increased by one until a unique file name is created.</li> </ul> <p>Once the backup has been created, InstallShield stores the backup file name in the system variable <a href="#">INFOFILENAME</a>.</p> <div>  <p><b>Note</b> • If the batch file specified by the last call to BatchFileLoad did not exist, then the backup file is identical to the batch file created by the call to BatchFileSave. If szBackupFile specifies the name of the original batch file, then a backup file is not created.</p> </div>



## Return Values

**Table 24 • BatchFileSave Return Values**

Return Value	Description
0	BatchFileSave successfully saved the batch file in memory to disk.
< 0	BatchFileSave was unable to save the batch file to disk.

## BatchFileSave Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the BatchFileLoad and BatchFileSave functions.
*
* This example script shows how to open a batch file for
* editing, how to create a backup of the original file, and
* how to save and close the edited file.
*
* To demonstrate how the file backup feature of BatchFileSave
* prevents the overwriting of existing files, this script loads
* and saves two different batch files. The first batch file is
* backed up with a specific file name. The second is backed up
* with a wildcard extension so that BatchFileSave will
* generate a unique file extension consisting of three digits.
*
* Note: Before running this script, create two batch files
*       (ISExamp1.bat and ISExamp2.bat) in the root of drive C.
*       For best effect, you should delete or move any other
*       files named ISExamp1.* or ISExamp2.*.
*
\*-----*/

// Names of batch files and backup files used in this example.
#define EXAMPLE1 "ISExamp1"
#define EXAMPLE2 "ISExamp2"

// Full names of batch files.
#define EXAMPLE1_BAT "C:\\\" + EXAMPLE1 + ".bat"
#define EXAMPLE2_BAT "C:\\\" + EXAMPLE2 + ".bat"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_BatchFileSave(HWND);
```

```

function ExFn_BatchFileSave(hMSI)
begin

    // Load EXAMPLE1_BAT.
    if (BatchFileLoad (EXAMPLE1_BAT) < 0) then
        MessageBox ("Unable to load " + EXAMPLE1_BAT + ".", SEVERE);
        abort;
    endif;

    // Use other batch file functions here to edit the first file.
    // Back up the original file with the extension "bak"; save the
    // edited file under its original name. If ISExamp1.bak already
    // exists, BatchFileSave will generate a numbered extension.
    if (BatchFileSave (EXAMPLE1 + ".bak") < 0) then
        MessageBox ("Unable to save " + EXAMPLE1_BAT + ".", SEVERE);
        abort;
    else
        MessageBox (EXAMPLE1_BAT + " saved.", INFORMATION);
    endif;

    // Load EXAMPLE2_BAT.
    if (BatchFileLoad (EXAMPLE2_BAT) < 0) then
        MessageBox ("Unable to load " + EXAMPLE2_BAT + ".", SEVERE);
        abort;
    endif;

    // Use other batch file functions here to edit the second file.
    // Back up the original batch file with a numbered extension
    // and save the edited file under its original name.
    if (BatchFileSave (EXAMPLE2 + ".*") < 0) then
        MessageBox ("Unable to save " + EXAMPLE2_BAT + ".", SEVERE);
        abort;
    else
        MessageBox (EXAMPLE2_BAT + " saved.", INFORMATION);
    endif;

end;

```

## BatchFind

The **BatchFind** function searches a batch file for one or more occurrences of the reference key specified in szRefKey. If you specify the constant RESTART in nOptions, the first occurrence of the reference key is returned. To find the next occurrence of szRefKey, call this function repeatedly with nOptions set to CONTINUE.



**Note** • Before calling *BatchFind*, you must call *BatchFileLoad* to load the file to be modified into memory. After you modify the file, call *BatchFileSave* to save it to disk.

Do not mix the Ez batch file functions with the advanced batch file functions. After calling *BatchFileLoad*, you cannot use Ez batch file functions until you have called *BatchFileSave* to save the file.

### Syntax

```
BatchFind ( szRefKey, svResult, nOptions );
```

## Parameters

**Table 25 • BatchFind Parameters**

Parameter	Description
<b>szRefKey</b>	Specifies the reference key to search for. The reference key can be an environment variable, a DOS command, or a program name. If the reference key is a file name and you do not specify a file extension, the function returns all reference keys with the base file name. For example, if you specify Win.com, the search looks for this reference key only. If you specify Win, the reference keys Win.exe, Win.dll, Win.sys, and so on will be returned if they exist in the batch file.
<b>svResult</b>	Specifies the value of the reference key that was found in the batch file.
<b>nOptions</b>	<p>Specifies where to start the search; pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>CONTINUE</b>—Starts the search from the current position in the batch file.</li> <li>• <b>RESTART</b>—Starts the search from the beginning of the batch file.</li> </ul> <p>When the reference key you are searching for is a DOS command or program name (not an environment variable), use the OR operator to combine the constant COMMAND with CONTINUE or RESTART, as shown below:</p> <pre>BatchFind ("SCAN.EXE", svResult, COMMAND   RESTART);</pre>

## Return Values

**Table 26 • BatchFind Return Values**

Return Value	Description
<b>0</b>	BatchFind successfully found the value of szRefKey and returned it in svResult.
<b>&lt; 0</b>	BatchFind was unable to find the value of szRefKey and return it in svResult.

## BatchFind Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the BatchFind function.
 *
 * This example script searches a batch file and reports whether
 * or not the file includes a command that references SHARE.EXE.
 * It then finds and displays all PATH and SET PATH statements.
 *
 * Note: Before running this script, create a batch file
 *       named ISExempl.bat and store it in the root of
 *       drive C. The batch file should include a command
 *       to launch Share.exe, and it should contain at least
 *       one PATH or SET PATH= statement.
 *
 \*-----*/

#define TARGET_BATCH "C:\\ISExempl.bat"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_BatchFind(HWND);

function ExFn_BatchFind(hMSI)
    STRING svResult;
    NUMBER nResult;
begin

    // Load the target batch file.
    if (BatchFileLoad (TARGET_BATCH ) < 0) then
        MessageBox ("Unable to load " + TARGET_BATCH + ".", SEVERE);
        abort;
    endif;

    // Check for a SHARE.EXE command.
    nResult = BatchFind ("SHARE.EXE", svResult, COMMAND);

    if (nResult < 0) then
        MessageBox ("SHARE.EXE command not found.", WARNING);
    else
        MessageBox ("SHARE.EXE command found.", INFORMATION);
    endif;

    // Find the first PATH or SET PATH= statement. Pass RESTART in
    // the third parameter to begin searching at the top of the file.
    nResult = BatchFind ("PATH", svResult, RESTART);

    if (nResult < 0) then
        MessageBox ("PATH command not found.", WARNING);

```

```

else

    // Loop while PATH commands are found.
    while (nResult = 0)
        MessageBox (svResult, INFORMATION);

        // Find the next PATH command. Pass CONTINUE in the
        // third parameter to continue the search with the
        // statement that follows the last match.
        nResult = BatchFind ("PATH", svResult, CONTINUE);
    endwhile;

    MessageBox ("No more PATH commands.", WARNING);

endif;

end;

```

## BatchGetFileName

The **BatchGetFileName** function retrieves the fully qualified name of the default batch file, which is set initially by InstallShield to the bootup Autoexec.bat file used by the system. To specify a different batch file to be used by default in the script, call [BatchSetFileName](#).




---

**Note** • Do not mix the Ez batch file functions with the advanced batch file functions. After calling [BatchFileLoad](#), you cannot use Ez batch file functions until you have called [BatchFileSave](#) to save the file.

### Syntax

```
BatchGetFileName ( svFileName );
```

## Parameters

**Table 27** • BatchGetFileName Parameters

Parameter	Description
<b>svFileName</b>	Returns the fully qualified name of the default batch file in svFileName.

## Return Values

**Table 28** • BatchGetFileName Return Values

Return Value	Description
<b>0</b>	BatchGetFileName successfully retrieved the fully qualified name of the default batch file.
<b>&lt; 0</b>	BatchGetFileName was unable to retrieve the fully qualified name of the default batch file.

## BatchGetFileName Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the BatchGetFileName and BatchSetFileName
* functions.
*
* This example script retrieves the fully qualified name of the
* default batch file, which initially is the Autoexec.bat file
* on the boot drive. It then makes C:\ISExampl.bat the default
* batch file. Finally, it retrieves the name of the default
* batch file again to show that it has been changed.
*
\*-----*/

#define DEFAULT_BATCH_FILE "C:\\ISExampl.bat"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_BatchGetFileName(HWND);

function ExFn_BatchGetFileName(hMSI)
    STRING svFilename;
begin

```

```

// Get the name of the default batch file.
if (BatchGetFileName (svFilename) < 0) then
    // Report the error; then abort.
    MessageBox ("First call to BatchGetFileName failed.", SEVERE);
    abort;
else
    // Display the name of the default batch file.
    MessageBox ("The default batch file is " + svFilename + ".",
        INFORMATION);
endif;

// Make C:\ISExempl.bat the default batch file.
if (BatchSetFileName(DEFAULT_BATCH_FILE) < 0) then
    // Report the error.
    MessageBox ("Unable to set new default batch file.", SEVERE);
else

    // Verify that the default batch file has been changed.
    if (BatchGetFileName(svFilename) < 0) then
        // Handle an error.
        MessageBox ("Second call to BatchGetFileName failed.", SEVERE);
    else
        // Display the name of the default batch file.
        MessageBox ("Now the default batch file is " + svFilename + ".",
            INFORMATION);
    endif;
endif;

end;

```

## BatchMoveEx

The **BatchMoveEx** function moves the line specified by szMove from one location to another in a batch file. The parameter nOptions specifies whether to position the line at the beginning or end of the batch file, or before or after the line specified by szRefKey.



**Note** • Before calling *BatchMoveEx*, you must call *BatchFileLoad* to load the file to be modified into memory. After you modify the file, call *BatchFileSave* to save it to disk.

Do not mix the Ez batch file functions with the advanced batch file functions. After calling *BatchFileLoad*, you cannot use Ez batch file functions until you have called *BatchFileSave* to save the file.

### Syntax

```
BatchMoveEx ( szMove, szRefKey, nOptions, nMoveOption );
```

## Parameters

**Table 29 • BatchMoveEx Parameters**

Parameter	Description
<b>szMove</b>	Specifies the reference key that identifies the line to be moved.
<b>szRefKey</b>	Specifies the key that identifies the reference line used to position the line being moved. If szRefKey is a null string (""), the line specified by szMove is moved to the beginning or end of the file, depending on the value of nOptions.
<b>nOptions</b>	<p>Specifies where to move the line. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>BEFORE</b>—The line specified by szMove is moved before the line containing the reference key in szRefKey. If szRefKey is a null string (""), the line specified by szMove is moved to the beginning of the file.</li> <li>• <b>AFTER</b>—The line specified by szMove is moved after the line containing the reference key in szRefKey. If szRefKey is a null string (""), the line specified by szMove is moved to the end of the file.</li> </ul> <p>When the reference key you are searching for is a DOS command or program name (not an environment variable), use the OR operator to combine the constant COMMAND with BEFORE or AFTER, as shown below:</p> <pre>BatchMoveEx ("PATH", "SCAN.EXE", BEFORE   COMMAND, 0);</pre>
<b>nMoveOption</b>	<p>Specifies whether szMove is a command or an environment variable. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>0</b>—Specifies that szMove is an environment variable.</li> <li>• <b>COMMAND</b>—Specifies that szMove is a command.</li> </ul>



## Return Values

**Table 30 • BatchMoveEx Return Values**

Return Value	Description
0	BatchMoveEx successfully moved the specified line in the batch file.
< 0	BatchMoveEx was unable to move the line in the batch file.

## BatchMoveEx Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the BatchMoveEx function.
*
* This example script moves lines within a batch file. First,
* it calls BatchFileLoad to load the file. Next, it moves
* the first PATH command to the end of the file. Then it
* moves the first statement that references Share.exe ahead
* of the statement that launches Windows.
*
* Note: Before running this script, create a batch file
*       named ISExempl.bat in the root of drive C. For
*       best effect, the first line in that file should
*       be a PATH command; the next statement should
*       launch Windows; the last statement should execute
*       Share.exe.
*
\*-----*/

#define TARGET_BATCH "C:\\ISExempl.bat"
#define BACKUP_BATCH "ISExempl.bak"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_BatchMoveEx(HWND);

function ExFn_BatchMoveEx(hMSI)
begin
    // Load the batch file to be edited.
    if BatchFileLoad (TARGET_BATCH) < 0 then
        MessageBox ("Unable to load " + TARGET_BATCH+".", SEVERE);
        abort;
```

```

endif;

// Move the PATH statement to the end of the file.
if (BatchMoveEx ("PATH", "", AFTER, 0) < 0) then
    MessageBox ("Unable to move PATH statement.", SEVERE);
else
    MessageBox ("PATH statement moved to end of file.", INFORMATION);
endif;

// Move the SHARE.EXE command before the WIN statement.
if (BatchMoveEx ("SHARE.EXE", "WIN", BEFORE|COMMAND, COMMAND) < 0) then
    MessageBox ("Unable to move SHARE.EXE statement.", SEVERE);
else
    MessageBox ("SHARE.EXE statement moved before WIN statement.",
                INFORMATION);
endif;

// Save the updated file; back up the original file.
if BatchFileSave (BACKUP_BATCH) < 0 then
    MessageBox ("Unable to save " + BACKUP_BATCH+".", SEVERE);
else
    MessageBox ("Batch file saved. Backup created.", INFORMATION);
endif;

end;

```

## BatchSetFileName

The **BatchSetFileName** function specifies the name of the batch file to be used by Ez batch file functions and by [BatchFileLoad](#) when it is called with a null string ("") as its parameter. In InstallScript, this file is referred to as the default batch file. During installation initialization, the default batch file is set to the bootup Autoexec.bat file used by the system.

BatchSetFileName simply assigns the name of the default batch file. It does not verify that the specified file exists, nor does it load the file into memory. Because of this, the function will succeed even if the file name is invalid or the specified file does not exist. An invalid file name causes subsequent Ez batch file and advanced batch file functions to fail.




---

**Note** • Do not mix the Ez batch file functions with the advanced batch file functions. After calling *BatchFileLoad*, you cannot use Ez batch file functions until you have called *BatchFileSave* to save the file.

### Syntax

```
BatchSetFileName ( szBatchFile );
```

## Parameters

**Table 31** • BatchSetFileName Parameters

Parameter	Description
<b>szBatchFile</b>	Specifies the fully qualified name of the batch file to be used by default in the installation script.

## Return Values

**Table 32** • BatchSetFileName Return Values

Return Value	Description
<b>0</b>	BatchSetFileName successfully set the specified file as the default batch file.
<b>&lt; 0</b>	BatchSetFileName was unable to set the file as the default batch file.

## BatchSetFileName Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the BatchGetFileName and BatchSetFileName
* functions.
*
* This example script retrieves the fully qualified name of the
* default batch file, which initially is the Autoexec.bat file
* on the boot drive. It then makes C:\ISExempl.bat the default
* batch file. Finally, it retrieves the name of the default
* batch file again to show that it has been changed.
*
\*-----*/

#define DEFAULT_BATCH_FILE "C:\\ISExempl.bat"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_BatchSetFileName(HWND);

function ExFn_BatchSetFileName(hMSI)
    STRING svFilename;
begin

```

```

// Get the name of the default batch file.
if (BatchGetFileName (svFilename) < 0) then
    // Report the error; then abort.
    MessageBox ("First call to BatchGetFileName failed.", SEVERE);
    abort;
else
    // Display the name of the default batch file.
    MessageBox ("The default batch file is " + svFilename + ".",
        INFORMATION);
endif;

// Make C:\ISExempl.bat the default batch file.
if (BatchSetFileName(DEFAULT_BATCH_FILE) < 0) then
    // Report the error.
    MessageBox ("Unable to set new default batch file.", SEVERE);
else

    // Verify that the default batch file has been changed.
    if (BatchGetFileName(svFilename) < 0) then
        // Handle an error.
        MessageBox ("Second call to BatchGetFileName failed.", SEVERE);
    else
        // Display the name of the default batch file.
        MessageBox ("Now the default batch file is " + svFilename + ".",
            INFORMATION);
    endif;
endif;

end;

```

## CalculateAndAddFileCost

The **CalculateAndAddFileCost** function determines the cost of the specified file and adds it to the current value of nvCostHigh and/or nvCostLow. This allows you to calculate and add up the cost of multiple files by calling the function multiple times in a loop. Set nvCostHigh and nvCostLow to zero before calling the function to determine the cost of a single file. This function is typically used when you need to determine the cost of file of a known size so this cost can then be passed to FeatureAddCost.



**Note** • Note that this function does not actually set any information to be used directly by the installation. You must call FeatureAddCost (as appropriate) after calling this function to add the additional cost to an existing feature.

### Syntax

```
CalculateAndAddFileCost ( nFileSizeHigh, nFileSizeLow, szTargetDir, nClusterSize, nvCostHigh, nvCostLow
);
```

## Parameters

**Table 33** • CalculateAndAddFileCost Parameters

Parameter	Description
<b>nFileSizeHigh</b>	The upper 31 bits of the file size (in bytes). Typically retrieved using <code>GetFileInfo</code> .
<b>nFileSizeLow</b>	The lower 31 bits of the file size (in bytes). Typically retrieved using <code>GetFileInfo</code> .
<b>szTargetDir</b>	If <code>nClusterSize</code> is 0, the target folder for the file. This path is used to determine the cluster size of the target drive. If <code>nClusterSize</code> is non-zero, this parameter is ignored.
<b>nClusterSize</b>	Specifies the cluster size of the target drive. If this parameter is 0, the function determines this information from <code>szTargetDir</code> .
<b>nvCostHigh</b>	The upper 31 bits of the installation cost (in bytes) of this file is added to the current value of this variable.
<b>nvCostLow</b>	The lower 31 bits of the installation cost (in bytes) of this file is added to the current value of this variable.

## Return Values

**Table 34** • CalculateAndAddFileCost Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	Indicates that the function was successful.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was not successful.

# CallDLLFx



**Tip** • The `CallDLLFx` function is supported only for compatibility with scripts created in previous versions of InstallShield. Consider using the more flexible method described in *Calling a .dll File Function* instead of using the `CallDLLFx` function.

The `CallDLLFx` function calls a function within a specified .dll file.

## Syntax

```
CallDLLFx ( szDLL, szFunction, lvValue, svValue );
```

The function called must use the following fixed definition, where `hwnd` is the main window handle for the main InstallShield window:

```
LONG WINAPI YourFunction (HWND hwnd, LPLONG lpIValue, LPSTR lpszValue);
```

## Parameters

**Table 35 • CallDLLFx Parameters**

Parameter	Description
<b>szDLL</b>	Specifies the fully qualified file name of the .dll file that contains the function to execute.
<b>szFunction</b>	Specifies the name of the function in the .dll file specified in <code>szDLL</code> .
<b>lvValue</b>	Specifies a long integer variable to pass by reference to the .dll function.
<b>svValue</b>	Specifies a string variable to pass to the .dll function.

## Return Values

The `CallDLLFx` function returns a long integer from the function in the .dll file.

## CallDLLFx Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the CallDLLFx function.
*
* Note: This script requires that the constant DLL_FILE be set
*       to the fully qualified name of a .dll file that contains
*       a function called Test whose format matches the
*       prototype declaration below. That function should
*       modify the values passed in the third and fourth
*       parameters and then return those values in the same
*       parameters.
*
\*-----*/
#define ID_NEXT 1 // Return value if user clicks 'Next' button
```

```

#define ID_CANCEL 2    // Return value if user clicks 'Cancel' button
#define ID_BACK 4     // Return value if user clicks 'Back' button

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_CallDLLFx(HWND);

function ExFn_CallDLLFx(hMSI)
    INT    nValue, nResult;
    STRING szString, szResult, szDLL, szValue, szReturn;
begin

    // Set the setup window title.
    SetTitle ("CallDLLFx Example", 18, WHITE);

    // Set the location of the .dll.
    szDLL = SUPPORTDIR ^ "MYDLL.DLL";

    // Set up parameters for call to .dll function.
    nValue = 3000;
    szString = "Test String";

    // Show inputs to users.
    sprintfBox (INFORMATION, "", "Before - nValue: %i , szString: %s",
    nValue, szString);

    // Call .dll function; pass by value.
    nResult = CallDLLFx(szDLL, "Test", nValue, szString);

    // Show values returned by .dll function.
    sprintfBox(INFORMATION, "", "Returned - nValue: %i , szString: %s",
    nValue, szString);

end;

```

## ChangeDirectory

The **ChangeDirectory** function sets the current directory.




**Note** • After you call *ChangeDirectory* to make a specified directory the current directory, that directory cannot be deleted. Before you can delete that directory, you must call *ChangeDirectory* again to set a different current directory.

### Syntax

```
ChangeDirectory ( szPath );
```

## Parameters

**Table 36** • ChangeDirectory Parameters

Parameter	Description
<b>szPath</b>	Specifies the name of the directory to set as the current directory. That name can be either a fully qualified directory name or a UNC path; it must not include a trailing backslash. If necessary, call <a href="#">StrRemoveLastSlash</a> before calling ChangeDirectory.
 <p><b>Note</b> • When you are specifying a file in your script, always specify the full path (using the appropriate InstallShield system variable, for example, SRCDIR) rather than depend on the current folder having the appropriate value. The script internally executes code that can change the current folder, so its value may not be what you expect.</p>	

## Return Values

**Table 37** • ChangeDirectory Return Values

Return Value	Description
<b>0</b>	ChangeDirectory successfully set the specified directory as the current directory.
<b>&lt; 0</b>	ChangeDirectory was unable to set the specified directory as the current directory.

## ChangeDirectory Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ChangeDirectory function.
*
* This example script makes the Windows folder the current
* directory and then launches NotePad to display the file
* Readme.txt.
```



```

*
\*-----*/
// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_ChangeDirectory(HWND);

function ExFn_ChangeDirectory(hMSI)
begin

    // Make the Windows folder the default directory. Note that
    // the InstallShield system variable WINDIR points
    // to the Windows folder.

    ChangeDirectory (WINDIR);
    // Launch Notepad to view the Windows Readme.txt file.
    LaunchApp ("Notepad.exe", "Readme.txt" );

end;

```

## CharReplace

The **CharReplace** function replaces all instances of the character cFind with cReplace in the string svString, except characters whose string index is less than nStart. The string index of the first character in a string is 0.

### Syntax

```
CharReplace ( svString, cFind, cReplace, nStart );
```

## Parameters

**Table 38 • CharReplace Parameters**

Parameter	Description
<b>svString</b>	Specifies the string whose characters will be replaced, and returns the modified string.
<b>cFind</b>	Specifies the character to be replaced. Use the <a href="#">STRTOCHAR</a> function to specify a char literal as cFind; for example:  <code>CharReplace( svString, STRTOCHAR('a'), STRTOCHAR('e'), nStart );</code>
<b>cReplace</b>	Specifies the character to replace cFind. Use the <a href="#">STRTOCHAR</a> function to specify a char literal as cReplace; for example:  <code>CharReplace( svString, STRTOCHAR('a'), STRTOCHAR('e'), nStart );</code>
<b>nStart</b>	Specifies the string index at which to begin searching for cFind. Note that the string index of the first character in svString is 0 (zero). If you want to replace all instances of cFind in svString, specify 0 for nStart.

## Return Values

**Table 39 • CharReplace Return Values**

Return Value	Description
<b>X</b>	The total number of replacements of cFind by cReplace.
<b>&lt; ISERR_SUCCESS</b>	The function failed to replace the characters.

## Additional Information

cFind or cReplace can be null characters ('\0'). To handle null-delimited strings, specify cFind or cReplace as StrToChar('\0'); note that the constant NULL is 0, not '\0', and cannot be used to specify the null character. When cFind or cReplace is '\0', CharReplace automatically sets the last two characters of the string buffer to '\0' before returning; therefore, the size of the string (which should be set explicitly) should be at least the number of characters to be stored plus two.

## CharReplace Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
//-----
//
// InstallScript Example Script
//
```

```
// Demonstrates the CharReplace function.
//
// This sample shows a sample path string before and after replacing
// every backslash character with a forward slash.
//
//-----

function OnBegin( )
    STRING path_to_convert;
begin

    // example path to convert
    path_to_convert = FOLDER_COMMON_APPDATA;

    MessageBox("Path before conversion: " + path_to_convert,
        INFORMATION);

    // replace backslashes with forward slashes
    CharReplace(path_to_convert,
        STRTOCHAR('\\'), STRTOCHAR('/'), 0);

    MessageBox("Path after conversion: " + path_to_convert,
        INFORMATION);

end;
```

## CloseFile

The **CloseFile** function closes a file that has been opened with a call to `OpenFile`. You cannot read from or write to a file after you close it.

### Syntax

```
CloseFile ( nvFileHandle );
```

## Parameters

**Table 40 • CloseFile Parameters**

Parameter	Description
<b>nvFileHandle</b>	Specifies the handle of the file to be closed.

## Return Values

**Table 41 • CloseFile Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully closed the file.
<b>&lt; 0</b>	Indicates that the function was unable to close the file.

## CloseFile Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the OpenFile and CloseFile functions.
*
* OpenFile is called to open a file, which is then read into
* a list. The file is then closed and the list is displayed.
*
* Note: Before running this script, set the preprocessor
*       constants so that they reference an existing file
*       in an existing directory.
*
\*-----*/

#define EXAMPLE_FILE "Readme.txt"
#define EXAMPLE_DIR "C:\\Windows"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_CloseFile(HWND);

function ExFn_CloseFile(hMSI)
    STRING    svLine;
    NUMBER    nvFileHandle;
    LIST      listID;

```

```

begin

    // Set the file mode to normal.
    OpenFileMode (FILE_MODE_NORMAL);

    // Open the text file.
    if (OpenFile (nvFileHandle, EXAMPLE_DIR, EXAMPLE_FILE) < 0) then
        MessageBox ("OpenFile failed.", SEVERE);
        abort;
    endif;

    // Create an empty string list.
    listID = ListCreate (STRINGLIST);

    // Read lines from the text file into the string list.
    while GetLine (nvFileHandle, svLine) = 0
        ListAddString (listID, svLine, AFTER);
    endwhile;

    // Close the file.
    if (CloseFile (nvFileHandle) < 0) then
        MessageBox ("CloseFile failed.", SEVERE);
    endif;

    // Display the text that was read from the file.
    SdShowInfoList ("", "", listID);

end;

```

## CmdGetHwndDlg

The **CmdGetHwndDlg** function retrieves the window handle of the dialog identified by szDialogName. The dialog already must have been defined with **EzDefineDialog** or **DefineDialog** and initialized by calling **WaitOnDialog**.

**CmdGetHwndDlg** is typically called in the DLG\_INIT routine for a custom dialog. The handle of the dialog is assigned to a HWND variable to be used by other functions that need it.



**Note** • When a dialog is initialized with the **WaitOnDialog** function, a window handle is assigned to it; that handle is associated with the dialog only until it is closed by a call to **EndDialog**. If you call **WaitOnDialog** to open a dialog that has been opened and closed previously in your script, you must call **CmdGetHwndDlg** again to get the new handle. The old handle is no longer valid.

### Syntax

```
CmdGetHwndDlg ( szDialogName );
```

## Parameters

**Table 42** • CmdGetHwndDlg Parameters

Parameter	Description
<b>szDialogName</b>	Specifies the name of a dialog that has been defined with <b>EzDefineDialog</b> or <b>DefineDialog</b> .

## Return Values

**Table 43** • CmdGetHwndDlg Return Values

Return Value	Description
<b>&gt; 0</b>	The window handle of the dialog that was specified by szDialogName.
<b>&lt; 0</b>	<b>CmdGetHwndDlg</b> was unable to retrieve the handle. Verify that szDialogName refers to a dialog that has been properly defined and initialized.

## CmdGetHwndDlg Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the CmdGetHwndDlg function.
*
* This example displays a custom dialog. On initialization
* of the dialog, the script calls CmdGetHwndDlg to retrieve
* the dialog's window handle so that it can perform the
* following operations:
*
* -- Change the text of the window's title bar.
* -- Disable and enable buttons in the dialog.
* -- Send messages to maximize and restore the dialog window.
*
* The "custom" dialog used in this script is actually the
* InstallShield Sd dialog that is displayed by the
* built-in function SdBitmap. Because this dialog is stored in
* the file _isres.dll, which is already compressed in the
* installation, it can be used in a script as a custom dialog.
* Note that the script changes the static text of the dialog's
* Back and Next buttons to fit the requirements of the example.
*
\*-----*/

// Dialog controls
#define RES_DIALOG_ID      12027    // ID of the custom dialog
```

```

#define RES_PBUT_RESTORE      1    // ID of dialog's Next button
#define RES_PBUT_CANCEL      9    // ID of dialog's Cancel button
#define RES_PBUT_MAXIMIZE    12   // ID of dialog's Back button

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_CmdGetHwndDlg(HWND);

function ExFn_CmdGetHwndDlg(hMSI)
    STRING szDialogName;
    NUMBER nResult, nCmdValue, hwndDlg;
    BOOL bDone;
    HWND hwndDlg;
begin
    // Specify a name to identify the custom dialog in
    // this installation.
    szDialogName = "CustomDialog";

    // Define the dialog. Pass a null string in the second parameter
    // to get the dialog from _ISUSER.DLL or _ISRES.DLL. Pass a null
    // string in the third parameter because the dialog is identified
    // by its ID in the fourth parameter.
    nResult = EzDefineDialog (szDialogName, "", "", RES_DIALOG_ID);

    if (nResult < 0) then
        // Report an error; then terminate.
        MessageBox ("Error in defining dialog", SEVERE);
        abort;
    endif;

    // Initialize the indicator used to control the loop.
    bDone = FALSE;

    // Loop until done.
    repeat
        // Display the dialog and return the next dialog event.
        nCmdValue = WaitOnDialog (szDialogName);

        // Respond to the event.
        switch (nCmdValue)
            case DLG_CLOSE:
                // The user clicked the window's Close button.
                Do (EXIT);
            case DLG_ERR:
                MessageBox ("Unable to display dialog. Setup canceled.", SEVERE);
                abort;
            case DLG_INIT:
                // Initialize the back, next, and cancel button enable/disable
                // states for this dialog and replace %P, %VS, %VI with
                // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
                // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
                hwndDlg = CmdGetHwndDlg(szDialogName);
                SdGeneralInit(szDialogName, hwndDlg, 0, "");

                // Set the static text of the buttons.
                CtrlSetText (szDialogName, RES_PBUT_MAXIMIZE, "&Maximize");
                CtrlSetText (szDialogName, RES_PBUT_RESTORE, "&Restore");
        endswitch
    until bDone;
end

```

```

        // Disable the Restore button using a call from Winsub.
        _WinSubEnableControl (hwndDlg, RES_PBUT_RESTORE, 0);
    case RES_PBUT_RESTORE:
        // Restore the window to its normal size.
        SendMessage (hwndDlg, WM_SYSCOMMAND, SC_RESTORE, 0);

        // Disable the Restore button using a call from Winsub.
        _WinSubEnableControl (hwndDlg, RES_PBUT_RESTORE, 0);

        // Enable the Maximize button using a call from Winsub.
        _WinSubEnableControl (hwndDlg, RES_PBUT_MAXIMIZE, 1);
    case RES_PBUT_MAXIMIZE:
        // Maximize the dialog's window.
        SendMessage (hwndDlg, WM_SYSCOMMAND, SC_MAXIMIZE, 0);

        // Disable the Maximize button using a call from Winsub.
        _WinSubEnableControl (hwndDlg, RES_PBUT_MAXIMIZE, 0);

        // Enable the Restore button using a call from Winsub.
        _WinSubEnableControl (hwndDlg, RES_PBUT_RESTORE, 1);
    case RES_PBUT_CANCEL:
        // The user clicked the Cancel button.
        Do (EXIT);
    endswitch;

until bDone;

// Close the dialog.
EndDialog (szDialogName);

// Free the dialog from memory.
ReleaseDialog (szDialogName);

end;

```

## CoCreateObject



**Project** • This information applies to InstallScript projects.

The **CoCreateObject** function initializes the COM object named by szProgID and returns a reference that can be assigned to a variable of type OBJECT by using the set keyword.

### Syntax

```
CoCreateObject ( szProgID );
```



## Parameters

**Table 44 • CoCreateObject Parameters**

Parameter	Description
<b>szProgID</b>	Specifies the program ID of the COM object to be initialized.

## Return Values

The reference can be assigned to a variable of type OBJECT by using the set keyword.

## Additional Information

- To check whether the object was initialized successfully, call the [IsObject](#) function.
- Any object variable can be released by setting the object variable to the value of NOTHING or reassigning the object with the **CoCreateObject**, **CoCreateObjectDotNet**, **CoGetObject**, or **DotNetCoCreateObject** functions. However, this does not automatically unload the library referenced by the object. You must call the Windows API, CoFreeLibrary, manually to free the library. Otherwise, the library remains loaded until the installation finishes. For more information, see [Extending Your Installation with COM Objects](#).

# CoCreateObjectDotNet



**Project** • The following project types support the **CoCreateObjectDotNet** function:

- *InstallScript*
- *InstallScript MSI*
- *Basic MSI with InstallScript custom actions*

The **CoCreateObjectDotNet** function has been deprecated. Calling this function is the same as calling the **DotNetCoCreateObject** function with a null string ("") for the szAppDomain parameter.

For more information, see [DotNetCoCreateObject](#).

# CoGetObject



**Project** • This information applies to *InstallScript* projects.

The **CoGetObject** function returns a reference to the specified COM object (as Visual Basic's GetObject function does); that reference can be assigned to a variable of type OBJECT by using the set keyword.

## Syntax

```
CoGetObject ( szFilename, szProgID );
```

## Parameters

**Table 45 • CoGetObject Parameters**

Parameter	Description
<b>szFilename</b>	Specifies the fully qualified name of the COM object. This parameter can be a null string (""), if szProgID is non-null.
<b>szProgID</b>	Specifies the program ID of the COM object. This parameter can be a null string (""), if szFilename is non-null.

## Return Values

- A reference that can be assigned to a variable of type OBJECT by using the set keyword.
- Any object variable can be released by setting the object variable to the value of NOTHING or reassigning the object with the **CoCreateObject**, **CoCreateObjectDotNet**, **CoGetObject**, or **DotNetCoCreateObject** functions. However, this does not automatically unload the library referenced by the object. You must call the Windows API, CoFreeLibrary, manually to free the library. Otherwise, the library remains loaded until the installation finishes. For more information, see Extending Your Installation with COM Objects.

## Additional Information

To check whether the object was initialized successfully, call the [IsObject](#) function.

## CoGetObject Example

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the CoGetObject function.
*
* This example shows how to create a virtual
* directory on IIS server.
*
\*-----*/

#include "ifx.h"

#define VIRTUALDIR "My Virtual Dir"
#define VIRTUALDIRPATH "c:\inetpub\wwwroot\MyDir"

function OnBegin()
OBJECT objIIS_Root, objVirtDir;

begin

    set objIIS_Root = CoGetObject("IIS://localhost/W3SVC/1/Root", "");
    if (IsObject(objIIS_Root)) then
        try
```

```

set objVirtDir = objIIS_Root.Create("IISWebVirtualDir", VIRTUALDIR);
if (IsObject(objVirtDir)) then
    objVirtDir.Path = VIRTUALDIRPATH;
    objVirtDir.AccessRead = TRUE;
    objVirtDir.AccessScript = TRUE;
    objVirtDir.SetInfo();
    objVirtDir.AppCreate(TRUE);
    objVirtDir.SetInfo();
endif;
catch
    MessageBox("Unable to create Virtual Directory.", INFORMATION);
endcatch;
endif;

end;

```

## ConfigAdd

The **ConfigAdd** function adds a statement to the system configuration file that has been loaded into memory with [ConfigFileLoad](#). You can specify the position of the statement relative to a reference key, or you can add the statement as the first or last line of the file. You can also replace an existing line in the file.




---

**Note** • Before calling *ConfigAdd*, you must first call *ConfigFileLoad* to load the system configuration file into memory. After you edit the file, call [ConfigFileSave](#) to save the file.

Do not mix the Ez configuration file functions with the advanced configuration file functions. After calling *ConfigFileLoad*, you cannot use the Ez configuration file functions until you call *ConfigFileSave* to save your changes.

### Syntax

```
ConfigAdd ( szKey, szValue, szRefKey, nOptions );
```

## Parameters

**Table 46 • ConfigAdd Parameters**

Parameter	Description
<b>szKey</b>	Specifies the keyword in the statement that is being added to the system configuration file.
<b>szValue</b>	Specifies the value of the keyword that is being added to the system configuration file.
<b>szRefKey</b>	Specifies the reference key relative to which you are adding szKey in the system configuration file. If you pass a null string ("") in this parameter, the line is added as the first or last line in the file, depending on which predefined constant is passed in nOptions.
<b>nOptions</b>	<p>Specifies whether the line is to be added before or after the line containing the reference key, or whether the line replaces an existing line. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>BEFORE</b>—The statement is added before the line containing szRefKey. If szRefKey is a null string (""), the statement is added as the first line of the file.</li> <li>• <b>AFTER</b>—The statement is added after the line containing szRefKey. If szRefKey is a null string (""), the statement is added as the last line of the file.</li> <li>• <b>REPLACE</b>—The statement replaces an existing line in the file. If multiple lines with same key exist, only the last line is replaced. If a line to be replaced does not exist in the file, the new line is added as the last line of the file.</li> </ul>

## Return Values

**Table 47 • ConfigAdd Return Values**

Return Value	Description
<b>0</b>	ConfigAdd successfully added the statement to the specified system configuration file.
<b>&lt; 0</b>	ConfigAdd was unable to add the statement to the specified system configuration file.

## Additional Information

When the ConfigAdd function replaces a line in a system configuration file, it compares the reference keys in the two lines. A reference key is a substring that identifies the line. For example, in the following statement, the

reference key is Kybrd.drv.

```
DEVICE=C:\Windows\System\Kybrd.drv /1024 /C:345
```

In the next statement, the reference key is PATH:

```
SET PATH=C:\Windows;C:\Windows\System
```

## ConfigAdd Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ConfigAdd function.
*
* This example script adds two statements to a configuration
* file. First, it calls ConfigFileLoad to load the file for
* editing. Next, it adds a DEVICE statement. Then it adds a
* DEVICEHIGH statement. Finally, it backs up the original file
* and saves the edited file.
*
* Note: Before running this script, create a configuration
*       file named ISExempl.sys in the root of drive C.
*       That file should include the following lines:
*
*       DEVICE=C:\Exapp\Exapp.sys
*       DEVICE=C:\Otherapp.exe
*
\*-----*/

#define EXAMPLE_SYS "C:\\ISExempl.sys"
#define EXAMSYS_BAK "ISExempl.bak"

// Variables to pass as parameters to ConfigAdd.

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ConfigAdd(HWND);

function ExFn_ConfigAdd(hMSI)
    STRING szKey, szValue, szRefKey;
begin

    // Load the target config file into memory.
    if ConfigFileLoad (EXAMPLE_SYS) < 0 then
        MessageBox ("Unable to load " + EXAMPLE_SYS + ".", SEVERE);
        abort;
    endif;

    // Set up parameters for the first call to ConfigAdd.
    szKey    = "DEVICE";
```

```

szValue = "C:\\Exapp\\Exapp2.sys";
szRefKey = "Exapp.sys";

// Add the line DEVICE=C:\\Exapp\\Exapp2.SYS before the first
// statement that references Exapp.sys.
if (ConfigAdd (szKey, szValue, szRefKey, BEFORE) < 0) then
    MessageBox ("First call to ConfigAdd failed.", WARNING);
    abort;
endif;

// Set up parameters for the second call to ConfigAdd.
szKey = "DEVICEHIGH";
szValue = "C:\\Otherapp\\Otherapp.exe";
szRefKey = "Otherapp.exe";

// Replace the last existing line that references OtherApp.exe
// with the statement DEVICEHIGH=C:\\Otherapp\\Otherapp.exe.
if (ConfigAdd (szKey, szValue, szRefKey, REPLACE) < 0) then
    MessageBox ("Second call to ConfigAdd failed.", WARNING);
    abort;
endif;

// Backup the original file and save the edited file.
if ConfigFileSave (EXAMSYS_BAK) < 0 then
    MessageBox ("Unable to save " + EXAMPLE_SYS + ".", SEVERE);
else
    MessageBox (EXAMPLE_SYS + " was updated and saved.", INFORMATION);
endif;

end;

```

## ConfigDelete

The **ConfigDelete** function removes lines from the system configuration file that has been loaded into memory by a call to [ConfigFileLoad](#). The parameter `szKey` specifies a reference key that identifies the lines to be deleted. After using advanced configuration functions to edit a system configuration file, you must call [ConfigFileSave](#) to save your changes.



**Note** • Do not mix the Ez batch file functions with the advanced batch file functions. After calling *BatchFileLoad*, you cannot use Ez batch file functions until you have called [BatchFileSave](#) to save the file.

### Syntax

```
ConfigDelete ( szKey );
```

## Parameters

**Table 48 • ConfigDelete Parameters**

Parameter	Description
<b>szKey</b>	Specifies the reference key that identifies the line or lines to delete. Common reference keys include Himem.sys, FILES, and STACKS.

## Return Values

**Table 49 • ConfigDelete Return Values**

Return Value	Description
<b>0</b>	ConfigDelete successfully deleted the lines containing the reference key from the system configuration file.
<b>&lt; 0</b>	ConfigDelete was unable to delete the specified lines.

## ConfigDelete Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ConfigDelete function.
*
* This example script deletes statements from a configuration
* file. First, it calls ConfigFileLoad to load the file for
* editing. Next, it deletes lines that contain a FILES
* statement. Finally, it backs up the original file and
* saves the edited file.
*
* Note: Before running this script, create a configuration
*       file named ISExempl.sys in the root of drive C.
*       That file should include at least one FILES statement.
*
\*-----*/

#define TARGET_CONFIG "C:\\ISExempl.sys"
#define BACKUP_CONFIG "ISExempl.bak"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

```

```

export prototype ExFn_ConfigDelete(HWND);

function ExFn_ConfigDelete(hMSI)
    STRING szMsg;
begin

    // Load the target config file to be edited.
    if (ConfigFileLoad (TARGET_CONFIG) < 0) then
        MessageBox ("Unable to load " + TARGET_CONFIG + ".", SEVERE);
        abort;
    endif;

    // Remove all lines in the file that contain the key "FILES".
    if (ConfigDelete ("FILES") < 0) then
        MessageBox ("ConfigDelete failed.", SEVERE);
    else
        // Back up the original file and save the edited file.
        if ConfigFileSave (BACKUP_CONFIG) < 0 then
            MessageBox ("Unable to save " + TARGET_CONFIG + ".", SEVERE);
        else
            MessageBox (TARGET_CONFIG + " was updated and saved.", INFORMATION);
        endif;
    endif;
end;

```

## ConfigFileLoad

The **ConfigFileLoad** function loads a copy of the specified system configuration file into memory so that other advanced configuration file functions can be called to operate on the file. Specify the name of the system configuration file you want to edit in `szConfigFile` or pass a null string ("" ) in `szConfigFile` to edit the default system configuration file, which is set initially by the installation to the bootup **Config.sys** file that is used by the system.



**Note** • Before using any of the advanced configuration file functions, you must first call **ConfigFileLoad** to load the system configuration file into memory. After you modify the file, call [ConfigFileSave](#) to save it to disk. To obtain the fully qualified name of the default system configuration file, call [ConfigGetFileName](#). To make another file the default system configuration file, call [ConfigSetFileName](#).

Note that you cannot call **ConfigFileLoad** to create a new configuration file. To create a new configuration file, use [CreateFile](#) and [CloseFile](#); this creates an empty file. Then use **ConfigFileLoad** and other functions to load and modify the file as needed.

Do not mix the Ez configuration file functions with the advanced configuration file functions. After calling the **ConfigFileLoad** function, you cannot use the Ez configuration file functions until you use the **ConfigFileSave** function to save your changes.

### Syntax

```
ConfigFileLoad ( szConfigFile );
```



## Parameters

**Table 50** • ConfigFileLoad Parameters

Parameter	Description
<b>szConfigFile</b>	Specifies the fully qualified name of the system configuration file to load into memory. To load the default system configuration file, pass a null string ("") in this parameter.

## Return Values

**Table 51** • ConfigFileLoad Return Values

Return Value	Description
<b>0</b>	ConfigFileLoad successfully initialized the configuration file buffer. If szConfigFile specified an existing configuration file, the file was loaded into the buffer; otherwise, an empty buffer was created.
<b>&lt; 0</b>	ConfigFileLoad was unable to initialize the configuration file buffer.

## ConfigFileLoad Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ConfigFileLoad and ConfigFileSave functions.
*
* This example script shows how to open a configuration file
* for editing, how to create a backup of the original file,
* and how to save and close the edited file.
*
* To demonstrate how the file backup feature of ConfigFileSave
* prevents the overwriting of existing files, this script loads
* and saves two different configuration files. The first file
* is backed up with a specific file name. The second is backed
* up with a wildcard extension so that ConfigFileSave will
* generate a unique file extension consisting of three digits.
*
* Note: Before running this script, create two files
*       (ISExamp1.sys and ISExamp2.sys) in the root of
*       drive C. For best effect, you should delete or
*       move any other files named ISExamp1.* or ISExamp2.*.
*
\*-----*/

// Names of Config files and backup files used in this example.
```

```

#define EXAMPLE1 "ISEXAMP1"
#define EXAMPLE2 "ISEXAMP2"

// Full names of Config files.
#define EXAMPLE1_SYS "C:\\\" + EXAMPLE1 + ".sys"
#define EXAMPLE2_SYS "C:\\\" + EXAMPLE2 + ".sys"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_ConfigFileLoad(HWND);

function ExFn_ConfigFileLoad(hMSI)
begin

    // Load EXAMPLE1_SYS.
    if (ConfigFileLoad (EXAMPLE1_SYS) < 0) then
        MessageBox ("Unable to load " + EXAMPLE1_SYS + ".", SEVERE);
        abort;
    endif;

    // Use other Config functions here to edit the first file.

    // Back up the original file with the extension 'bak'; save the
    // edited file under its original name.  If ISExamp1.bak already
    // exists, ConfigFileSave will generate a numbered extension.
    if (ConfigFileSave (EXAMPLE1 + ".bak") < 0) then
        MessageBox ("Unable to save " + EXAMPLE1_SYS + ".", SEVERE);
        abort;
    else
        MessageBox (EXAMPLE1_SYS + " saved.", INFORMATION);
    endif;

    // Load EXAMPLE2_SYS.
    if (ConfigFileLoad (EXAMPLE2_SYS) < 0) then
        MessageBox ("Unable to load " + EXAMPLE2_SYS + ".", SEVERE);
        abort;
    endif;

    // Use other Config file functions here to edit the second file.

    // Back up the original Config file with a numbered extension
    // and save the edited file under its original name.
    if (ConfigFileSave (EXAMPLE2 + ".*") < 0) then
        MessageBox ("Unable to save " + EXAMPLE2_SYS + ".", SEVERE);
        abort;
    else
        MessageBox (EXAMPLE2_SYS + " saved.", INFORMATION);
    endif;

end;

```

# ConfigFileSave

The **ConfigFileSave** function saves to disk a system configuration file that has been loaded into memory with the function [ConfigFileLoad](#). The file is saved under its original name. If a file name is specified in szBackupFile, the original file is renamed with that file name before the edited file is written to disk. If szBackupFile contains a null string (""), the original file is replaced with the modified file. If you do not call ConfigFileSave when you are finished modifying a system configuration file with advanced configuration file functions, all modifications will be lost.



---

**Note** • Do not mix the Ez configuration file functions with the advanced configuration file functions. After calling the ConfigFileLoad function, you cannot use the Ez configuration file functions until you use the ConfigFileSave function to save your changes.

## Syntax

```
ConfigFileSave ( szBackupFile );
```

## Parameters

**Table 52 • ConfigFileSave Parameters**

Parameter	Description
<b>szBackupFile</b>	<p>Specifies whether or not a backup copy of the original file as it existed before editing should be saved, according to the following criteria:</p> <ul style="list-style-type: none"> <li>• If no backup file should be created, specify a null string in this parameter.</li> <li>• If the original file should be backed up with a specific name, pass that file name in this parameter. The file name must be unqualified (that is, do not specify a drive and/or path). Note that if a file with the specified name already exists, ConfigFileSave generates a unique file extension, as described in the next bullet item.</li> <li>• If the original file should be backed up with an installation-generated file extension, specify the wildcard character (*) as the file extension (for example, "Config.*"). The installation then assigns a numeric value, starting at 001, as the extension. If a file already exists with that extension, the extension's value is increased by one until a unique file name is created.</li> </ul> <p>Once the backup has been created, InstallShield stores the backup file name in the system variable <a href="#">INFOFILENAME</a>.</p>

## Return Values

**Table 53 • ConfigFileSave Return Values**

Return Value	Description
<b>0</b>	ConfigFileSave successfully wrote the file from memory to disk.
<b>&lt; 0</b>	ConfigFileSave was unable to write the file to disk.

## ConfigFileSave Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ConfigFileLoad and ConfigFileSave functions.
*

```

```

* This example script shows how to open a configuration file
* for editing, how to create a backup of the original file,
* and how to save and close the edited file.
*
* To demonstrate how the file backup feature of ConfigFileSave
* prevents the overwriting of existing files, this script loads
* and saves two different configuration files. The first file
* is backed up with a specific file name. The second is backed
* up with a wildcard extension so that ConfigFileSave will
* generate a unique file extension consisting of three digits.
*
* Note: Before running this script, create two files
*       (ISExamp1.sys and ISExamp2.sys) in the root of
*       drive C. For best effect, you should delete or
*       move any other files named ISExamp1.* or ISExamp2.*.
*
\*-----*/

// Names of Config files and backup files used in this example.
#define EXAMPLE1 "ISEXAMP1"
#define EXAMPLE2 "ISEXAMP2"

// Full names of Config files.
#define EXAMPLE1_SYS "C:\\\" + EXAMPLE1 + ".sys"
#define EXAMPLE2_SYS "C:\\\" + EXAMPLE2 + ".sys"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_ConfigFileSave(HWND);

function ExFn_ConfigFileSave(hMSI)
begin
    // Load EXAMPLE1_SYS.
    if (ConfigFileLoad (EXAMPLE1_SYS) < 0) then
        MessageBox ("Unable to load " + EXAMPLE1_SYS + ".", SEVERE);
        abort;
    endif;

    // Use other Config functions here to edit the first file.

    // Back up the original file with the extension 'bak'; save the
    // edited file under its original name. If ISExamp1.bak already
    // exists, ConfigFileSave will generate a numbered extension.
    if (ConfigFileSave (EXAMPLE1 + ".bak") < 0) then
        MessageBox ("Unable to save " + EXAMPLE1_SYS + ".", SEVERE);
        abort;
    else
        MessageBox (EXAMPLE1_SYS + " saved.", INFORMATION);
    endif;

    // Load EXAMPLE2_SYS.
    if (ConfigFileLoad (EXAMPLE2_SYS) < 0) then
        MessageBox ("Unable to load " + EXAMPLE2_SYS + ".", SEVERE);
        abort;
    endif;

    // Use other Config file functions here to edit the second file.

```

```

// Back up the original Config file with a numbered extension
// and save the edited file under its original name.
if (ConfigFileSave (EXAMPLE2 + ".*") < 0) then
    MessageBox ("Unable to save " + EXAMPLE2_SYS + ".", SEVERE);
    abort;
else
    MessageBox (EXAMPLE2_SYS + " saved.", INFORMATION);
endif;

end;

```

## ConfigFind

The **ConfigFind** function searches a system configuration file that has been loaded into memory with the function [ConfigFileLoad](#). The parameter `szRefKey` is a reference key that specifies the search target in that file. If the reference key is found, its value is returned in `svResult`. To find all occurrences of `szRefKey`, call this function repeatedly with `nOptions` set to `CONTINUE`. To restart the search from the top of the file, specify the constant `RESTART` in `nOptions`. After you edit the file, call [ConfigFileSave](#) to save it.




---


**Note** • Do not mix the Ez configuration file functions with the advanced configuration file functions. After calling the *ConfigFileLoad* function, you cannot use the Ez configuration file functions until you use the *ConfigFileSave* function to save your changes.

### Syntax

```
ConfigFind (szRefKey, svResult, nOptions);
```

## Parameters

**Table 54 • ConfigFind Parameters**

Parameter	Description
<b>szRefKey</b>	Specifies the reference key to search for. If the reference key is a file name without a file extension, all file extensions are included in the search. For example, if you specify Win.com, the search looks only for that reference key. If you specify WIN, the files Win.exe, Win.dll, Win.sys, etc., are all returned.
<b>svResult</b>	Returns the value of the reference key that was found in the system configuration file.
<b>nOptions</b>	<p>Indicates whether to start the search from the beginning of the file or to continue from where the previous search was terminated. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>RESTART</b>—Starts the search from the beginning of the file.</li> <li>• <b>CONTINUE</b>—Starts the search from the current position in the system configuration file.</li> <li>• <b>COMMAND</b>—Indicates that the reference key in szRefKey is a command, not an environment variable. The constant COMMAND can be joined with the constant RESTART or the constant CONTINUE by using the bitwise OR operator ( ), as in the following example:</li> </ul> <pre>ConfigFind("Vga.drv", svResult, CONTINUE   COMMAND);</pre> <p> <b>Note</b> • A system configuration file can contain both environment variables and commands. To distinguish between environment variables and commands with the same name, use the constant COMMAND to specify that you are looking for executable commands.</p>

## Return Values

**Table 55 • ConfigFind Return Values**

Return Value	Description
0	ConfigFind successfully found the specified reference key and returned it in svResult.
< 0	ConfigFind was unable to find the specified reference key.

## ConfigFind Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the ConfigFind function.
 *
 * This example script searches a batch file and reports whether
 * or not the file includes a BUFFERS command. It then finds
 * and displays all commands that reference Abc44.sys.
 *
 * Note: Before running this script, create a configuration
 *       file called ISExempl.sys and store it in the root
 *       of drive C. The configuration file should include
 *       the following lines:
 *
 *       DEVICE=C:\Abc44.sys /e:300
 *       DEVICE=C:\Abc44.sys /s:off
 *       BUFFERS=50
 *
 \*-----*/

#define EXAMPLE_SYS "C:\\ISExempl.sys"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ConfigFind(HWND);

function ExFn_ConfigFind(hMSI)
    STRING svResult;
    NUMBER nResult;
begin
    // Load EXAMPLE_SYS.
    if (ConfigFileLoad (EXAMPLE_SYS) < 0) then
        MessageBox ("Unable to load " + EXAMPLE_SYS + ".", SEVERE);

```



```

        abort;
    endif;

    // Check for a BUFFERS command. RESTART is passed in the
    // third parameter to begin searching at the top of the file.
    nResult = ConfigFind("BUFFERS", svResult, RESTART);

    if (nResult < 0) then
        MessageBox ("BUFFERS command not found.", WARNING);
    else
        MessageBox (svResult, INFORMATION);
    endif;

    // Find the first command that references Abc44.sys.
    nResult = ConfigFind ("Abc44.sys", svResult, COMMAND | RESTART);

    if nResult < 0 then
        MessageBox ("The file Abc44.sys is not referenced.", WARNING);
    else

        // Loop while matching statements are found.
        while nResult = 0
            // Display the matching statement.
            MessageBox (svResult, INFORMATION);

            // Find the next statement that references Abc44.sys.
            nResult = ConfigFind ("Abc44.sys", svResult, CONTINUE);
        endwhile;

        MessageBox ("No more matches on Abc44.sys.", WARNING);

    endif;

end;

```

## ConfigGetFileName

The **ConfigGetFileName** function retrieves the fully qualified name of the default system configuration file, which is set initially by InstallShield to the Config.sys file that was executed when the target system was started. To specify a different batch file to be used by default in the script, call [ConfigSetFileName](#).




**Note** • Do not mix the Ez configuration file functions with the advanced configuration file functions. After calling the [ConfigFileLoad](#) function, you cannot use the Ez configuration file functions until you use the [ConfigFileSave](#) function to save your changes.

### Syntax

```
ConfigGetFileName (svFileName);
```

## Parameters

**Table 56 • ConfigGetFileName Parameters**

Parameter	Description
<b>svFileName</b>	Returns the fully qualified name of the default system configuration file.
 <p><b>Note</b> • In rare circumstances, InstallShield might not be able to determine the fully qualified name of the default configuration file. In that case, svFileName is a null string ("").</p>	

## Return Values

**Table 57 • ConfigGetFileName Return Values**

Return Value	Description
<b>0</b>	ConfigGetFileName successfully retrieved the fully qualified name of the default system configuration file.
<b>&lt; 0</b>	ConfigGetFileName was unable to retrieve the fully qualified name of the default system configuration file.

## ConfigGetFileName Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ConfigGetFileName and ConfigSetFileName
* functions.
*
* This example script retrieves the fully qualified name of the
* default configuration file, which initially is the Config.sys
* file on the boot drive. It then makes C:\ISExempl.sys the
* default configuration file. Finally, it retrieves the name of
* the default configuration file again to show that it has been
* changed.
*
\*-----*/
```

```

#define DEFAULT_CONFIG_FILE "C:\\ISExmpl.sys"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ConfigGetFileName(HWND);

function ExFn_ConfigGetFileName(hMSI)
    STRING svFilename;
begin
    // Get and display the name of the default configuration file.
    if (ConfigGetFileName (svFilename) < 0) then
        // Report the error; then terminate.
        MessageBox ("First call to ConfigGetFileName failed.", SEVERE);
        abort;
    else
        // Display the name of the default configuration file.
        MessageBox ("The default configuration file is " + svFilename + ".",
                    INFORMATION);
    endif;

    // Make C:\\ISExmpl.sys the default configuration file.
    if (ConfigSetFileName (DEFAULT_CONFIG_FILE) < 0) then
        // Report the error.
        MessageBox ("Unable to set new default configuration file.", SEVERE);
    else
        // Verify that the default configuration file has been changed.
        if (ConfigGetFileName (svFilename) = 0) then
            // Display the name of the default configuration file.
            MessageBox ("Now the default configuration file is " + svFilename +
                        ".", INFORMATION);
        else
            // Report the error.
            MessageBox ("Second call to ConfigGetFileName failed.", SEVERE);
        endif;
    endif;

end;

```

## ConfigGetInt

The **ConfigGetInt** function retrieves the integer value of a reference key from a system configuration file that has been loaded into memory with the function [ConfigFileLoad](#). ConfigGetInt retrieves values from commands that have only one value to the right of the equal sign (=).



**Note** • *ConfigGetInt does not work on a command that has more than one value. For example, ConfigGetInt recognizes the statement FILES=20 and returns the number 20, but it does not recognize the statement STACKS=9,128.*

*Before calling ConfigGetInt, you must first call ConfigFileLoad to load the system configuration file into memory. After you edit the file, call [ConfigFileSave](#) to save the file.*

*Do not mix the Ez configuration file functions with the advanced configuration file functions. After calling the ConfigFileLoad function, you cannot use the Ez configuration file functions until you use the ConfigFileSave function to save your changes.*

## Syntax

```
ConfigGetInt ( szKey, nvValue );
```

## Parameters

**Table 58 • ConfigGetInt Parameters**

Parameter	Description
<b>szKey</b>	Specifies the reference key of the statement from which to retrieve the integer value.
<b>nvValue</b>	Returns the integer value of the reference key.

## Return Values

**Table 59 • ConfigGetInt Return Values**

Return Value	Description
<b>0</b>	ConfigGetInt successfully retrieved the integer value.
<b>&lt; 0</b>	ConfigGetInt was unable to retrieve the integer value.

## ConfigGetInt Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ConfigGetInt and ConfigSetInt functions.
*
* This example script gets the current value of the FILES
* command from a configuration file. If a FILES command is not
* found, the command FILES=40 is added to the file. If a FILES
* command is found, its value is tested. If the value is less
* than 40, the command is replaced with the command FILES=40.
*
* Note: Before running this script, create a configuration file
* called ISExmpl.sys in the root directory of drive C.
* That file should include the following line:
```

```

*
*      FILES=20;
*
\*-----*/

#define EXAMPLE_SYS "C:\\ISExampl.sys"
#define EXAMPLE_BAK "ISExampl.bak"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ConfigGetInt(HWND);

function ExFn_ConfigGetInt(hMSI)
    NUMBER nvValue;
    BOOL   bFileChanged;
begin

    // Load the configuration file.
    if (ConfigFileLoad (EXAMPLE_SYS) < 0) then
        MessageBox ("Unable to load " + EXAMPLE_SYS + ".", SEVERE);
        abort;
    endif;

    // Initialize indicator to show if file was updated.
    bFileChanged = FALSE;

    // Find the command "FILES" in the configuration file.
    if (ConfigGetInt ("FILES", nvValue) < 0) then

        // No FILES command found. Add FILES command.
        if ConfigAdd ("FILES", "40", "", AFTER) = 0 then
            MessageBox ("FILES=40 added to " + EXAMPLE_SYS + ".", INFORMATION);
            bFileChanged = TRUE;
        else
            MessageBox ("FILES command not found. Unable to update " +
                EXAMPLE_SYS + ".", SEVERE);
        endif;

    else

        // FILES command found.
        if (nvValue >= 40) then
            // FILES command setting is ok.
            sprintfBox (INFORMATION, "ConfigGetInt Example",
                "FILES=%d; no change required.", nvValue);
        else

            // FILES command needs to be changed.
            if (ConfigSetInt ("FILES", 40) < 0) then
                MessageBox ("Unable to update "+EXAMPLE_SYS + ".", SEVERE);
            else
                MessageBox ("The FILES setting was changed to 40.", INFORMATION);
                bFileChanged = TRUE;
            endif;

        endif;

    endif;

end;

```

```

endif;

// If the file was edited, save it.
if bFileChanged then

    // Back up the original file with the extension 'bak'; save the
    // edited file under its original name. If ISExamp1.bak already
    // exists, ConfigFileSave will generate a numbered extension.
    if (ConfigFileSave (EXAMPLE_BAK) < 0) then
        MessageBox ("Unable to save " + EXAMPLE_SYS + ".", SEVERE);
    else
        MessageBox (EXAMPLE_SYS + " saved.", INFORMATION);
    endif;

endif;

end;

```

## ConfigMove

The **ConfigMove** function moves a line in a system configuration file that has been loaded into memory with the function [ConfigFileLoad](#). The line can be moved to the first or last position in the file or before or after a specific line in the file.




---

**Note** • Before calling the *ConfigMove* function, you must first call *ConfigFileLoad* to load the *Config.sys* file into memory. After you edit the file, call [ConfigFileSave](#) to save the file.

Do not mix the Ez configuration file functions with the advanced configuration file functions. After calling the *ConfigFileLoad* function, you cannot use the Ez configuration file functions until you use the *ConfigFileSave* function to save your changes.

### Syntax

```
ConfigMove ( szMove, szRefKey, nOptions );
```

## Parameters

**Table 60** • ConfigMove Parameters

Parameter	Description
<b>szMove</b>	Specifies the line to move.
<b>szRefKey</b>	Specifies the key that identifies the reference line used to position the line to be moved. The position of the line to be moved is determined by the value of nOptions.
<b>nOptions</b>	<p>Specifies whether you are moving the line in szMove before or after the line that contains the reference key in szRefKey. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>BEFORE</b>—The line specified by szMove is placed before the line containing szRefKey. If szMove is a null string (""), the line is placed before the first line in the system configuration file.</li> <li>• <b>AFTER</b>—The line specified by szMove is placed after the line containing szRefKey. If szMove is a null string (""), the line is placed after the last line in the system configuration file.</li> </ul>

## Return Values

**Table 61** • ConfigMove Return Values

Return Value	Description
<b>0</b>	ConfigMove successfully moved the specified line in the system configuration file.
<b>&lt; 0</b>	ConfigMove was unable to move the line.

## ConfigMove Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
```

```

*
* Demonstrates the ConfigMove function.
*
* This example script moves lines in a configuration file.
* First, it calls ConfigFileLoad to load the file. Next, it
* moves the FILES statement to the end of the file. Then,
* it moves the BUFFERS statement to the end of the file.
* Next, it moves the statement referencing Himem.sys before
* the DOS statement. Finally, it backs up the original file
* and saves the edited file.
*
* Note: Before running this script, create a configuration
*       file called ISExempl.sys in the root of drive C.
*       That file should include the following lines:
*
*       FILES=50
*       DOS=HIGH,UMB
*       DEVICE=C:\WINDOWS\SETVER.EXE
*       BUFFERS=50
*       Device=C:\Windows\Himem.sys
*
\*-----*/

#define TARGET_CONFIG "C:\\ISExempl.sys"
#define BACKUP_CONFIG "ISExempl.bak"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ConfigMove(HWND);

function ExFn_ConfigMove(hMSI)
begin
    // Load the configuration file to be edited.
    if ConfigFileLoad (TARGET_CONFIG) < 0 then
        MessageBox ("Unable to load " + TARGET_CONFIG + ".", SEVERE);
        abort;
    endif;

    // Move the FILES statement to the end of the file.
    if (ConfigMove ("FILES", "", AFTER) < 0) then
        MessageBox ("Unable to move FILES statement.", SEVERE);
    else
        MessageBox ("FILES statement moved to the end of the file.",
                    INFORMATION);
    endif;

    // Move the BUFFERS statement to the end of the file.
    if (ConfigMove ("BUFFERS", "", AFTER) < 0) then
        MessageBox ("Unable to move BUFFERS statement.", SEVERE);
    else
        MessageBox ("BUFFERS statement moved to the end of the file.",
                    INFORMATION);
    endif;

    // Move the Himem.sys statement before the DOS statement.
    if (ConfigMove ("Himem.sys", "DOS", BEFORE) < 0) then
        MessageBox ("Unable to move Himem.sys statement.", SEVERE);
    endif;
end

```



```

else
    MessageBox ("Himem.sys statement moved before DOS statement.",
                INFORMATION);
endif;

// Save the updated file; back up the original file.
if ConfigFileSave (BACKUP_CONFIG) < 0 then
    MessageBox ("Unable to save " + BACKUP_CONFIG+".", SEVERE);
else
    MessageBox ("Config file saved. Backup created.", INFORMATION);
endif;

end;

```

## ConfigSetFileName

The **ConfigSetFileName** function specifies the fully qualified name of the file you want to use as the default system configuration file. During installation initialization, the installation identifies the Config.sys file that was executed when the target system was started and makes it the default system configuration file. If this is the only system configuration file your installation will edit, it is unnecessary to call this function. Ez configuration files will use that file and the advanced configuration function [ConfigFileLoad](#) will open that file when its parameter is a null string ("").

However, if you want to use Ez configuration file functions to modify a configuration file other than the bootup Config.sys file, you must call ConfigSetFileName to change the default system configuration file. For example, suppose you wanted to create a Config.sys file on the target system that would not be used at bootup time. You can set a file name in the application directory. Ez configuration file functions would operate on that file. If you call ConfigFileLoad with a null parameter, that file is loaded into memory, where it can be edited with advanced file functions.




---

**Caution** • The *ConfigSetFileName* function does not load a system configuration file into memory. You must use *ConfigFileLoad* to load a file into memory.

*ConfigSetFileName* does not validate the file name you specify. If you specify an invalid file name, all future configuration file functions fail.




---

**Note** • Do not mix the Ez configuration file functions with the advanced configuration file functions. After calling the *ConfigFileLoad* function, you cannot use the Ez configuration file functions until you use the *ConfigFileSave* function to save your changes.

### Syntax

```
ConfigSetFileName ( szConfigFile );
```

## Parameters

**Table 62** • ConfigSetFileName Parameters

Parameter	Description
<b>szConfigFile</b>	Specifies the fully qualified name of the file to set as the default system configuration file.

## Return Values

**Table 63** • ConfigSetFileName Return Values

Return Value	Description
<b>0</b>	ConfigSetFileName successfully retrieved the specified system configuration file.
<b>&lt; 0</b>	ConfigSetFileName was unable to retrieve the specified file.

## ConfigSetFileName Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ConfigGetFileName and ConfigSetFileName
* functions.
*
* This example script retrieves the fully qualified name of the
* default configuration file, which initially is the Config.sys
* file on the boot drive. It then makes C:\ISeXempl.sys the
* default configuration file. Finally, it retrieves the name of
* the default configuration file again to show that it has been
* changed.
*
\*-----*/

#define DEFAULT_CONFIG_FILE "C:\\ISeXempl.sys"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_ConfigSetFileName(HWND);

function ExFn_ConfigSetFileName(hMSI)
    STRING svFilename;
begin

```

```

// Get and display the name of the default configuration file.
if (ConfigGetFileName (svFilename) < 0) then
    // Report the error; then terminate.
    MessageBox ("First call to ConfigGetFileName failed.", SEVERE);
    abort;
else
    // Display the name of the default configuration file.
    MessageBox ("The default configuration file is " + svFilename + ".",
        INFORMATION);
endif;

// Make C:\ISExampl.sys the default configuration file.
if (ConfigSetFileName (DEFAULT_CONFIG_FILE) < 0) then
    // Report the error.
    MessageBox ("Unable to set new default configuration file.", SEVERE);
else

    // Verify that the default configuration file has been changed.
    if (ConfigGetFileName (svFilename) = 0) then
        // Display the name of the default configuration file.
        MessageBox ("Now the default configuration file is " + svFilename +
            ".", INFORMATION);
    else
        // Report the error.
        MessageBox ("Second call to ConfigGetFileName failed.", SEVERE);
    endif;
endif;

end;

```

## ConfigSetInt

The **ConfigSetInt** function changes a specified integer value in a system configuration file that has been loaded into memory with the function [ConfigFileLoad](#). ConfigSetInt sets values in commands that have only one value to the right of the equal sign (=).



**Note** • This function does not work on a command that has more than one value. For example, ConfigSetInt recognizes the statement `FILES=20` and can change 20 to another value, but it does not recognize the statement `STACKS=9,128`.

Before calling ConfigSetInt, you must first call ConfigFileLoad to load the system configuration file into memory. After you edit the file, call [ConfigFileSave](#) to save the file.

Do not mix the Ez configuration file functions with the advanced configuration file functions. After calling the ConfigFileLoad function, you cannot use the Ez configuration file functions until you use the ConfigFileSave function to save your changes.

### Syntax

```
ConfigSetInt ( szKey, nValue );
```

## Parameters

**Table 64 • ConfigSetInt Parameters**

Parameter	Description
<b>szKey</b>	Specifies the reference keyword of the command whose integer value is to be set.
<b>nValue</b>	Specifies the integer value you want to set for the command in szKey.

## Return Values

**Table 65 • ConfigSetInt Return Values**

Return Value	Description
<b>0</b>	ConfigSetInt successfully set the specified integer in the system configuration file.
<b>&lt; 0</b>	ConfigSetInt was unable to set the specified integer.

## ConfigSetInt Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ConfigGetInt and ConfigSetInt functions.
*
* This example script gets the current value of the FILES
* command from a configuration file. If a FILES command is not
* found, the command FILES=40 is added to the file. If a FILES
* command is found, its value is tested. If the value is less
* than 40, the command is replaced with the command FILES=40.
*
* Note: Before running this script, create a configuration file
*       called ISExempl.sys in the root directory of drive C.
*       That file should include the following line:
*
*       FILES=20;
*
\*-----*/

```

```
#define EXAMPLE_SYS "C:\\ISExempl.sys"
```

```

#define EXAMPLE_BAK "ISExamp1.bak"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ConfigSetInt(HWND);

function ExFn_ConfigSetInt(hMSI)
    NUMBER nvValue;
    BOOL    bFileChanged;
begin
    // Load the configuration file.
    if (ConfigFileLoad (EXAMPLE_SYS) < 0) then
        MessageBox ("Unable to load " + EXAMPLE_SYS + ".", SEVERE);
        abort;
    endif;

    // Initialize indicator to show if file was updated.
    bFileChanged = FALSE;

    // Find the command "FILES" in the configuration file.
    if (ConfigGetInt ("FILES", nvValue) < 0) then

        // No FILES command found. Add FILES command.
        if ConfigAdd ("FILES", "40", "", AFTER) = 0 then
            MessageBox ("FILES=40 added to " + EXAMPLE_SYS + ".", INFORMATION);
            bFileChanged = TRUE;
        else
            MessageBox ("FILES command not found. Unable to update " +
                EXAMPLE_SYS + ".", SEVERE);
        endif;
    else

        // FILES command found.
        if (nvValue >= 40) then
            // FILES command setting is ok.
            sprintfBox (INFORMATION, "ConfigGetInt Example",
                "FILES=%d; no change required.", nvValue);
        else

            // FILES command needs to be changed.
            if (ConfigSetInt ("FILES", 40) < 0) then
                MessageBox ("Unable to update "+EXAMPLE_SYS + ".", SEVERE);
            else
                MessageBox ("The FILES setting was changed to 40.", INFORMATION);
                bFileChanged = TRUE;
            endif;
        endif;
    endif;

    // If the file was edited, save it.
    if bFileChanged then

        // Back up the original file with the extension 'bak'; save the
        // edited file under its original name. If ISExamp1.bak already

```

```

        // exists, ConfigFileSave will generate a numbered extension.
        if (ConfigFileSave (EXAMPLE_BAK) < 0) then
            MessageBox ("Unable to save " + EXAMPLE_SYS + ".", SEVERE);
        else
            MessageBox (EXAMPLE_SYS + " saved.", INFORMATION);
        endif;

    endif;

end;

```

## ConvertSizeToUnits

The **ConvertSizeToUnits** function converts the size specified through nSizeSrcHigh and nSizeSrcLow in nUnitsSrc into nvSizeTargetHigh and nvSizeTargetLow in nUnitsTarget. You can also use this function to determine the best units to use for a particular size value that is not known until the setup is run.

### Syntax

```

ConvertSizeToUnits ( nSizeSrcHigh, nSizeSrcLow, nUnitsSrc, nvSizeTargetHigh, nvSizeTargetLow,
                    nvUnitsTarget );

```

## Parameters

**Table 66 • ConvertSizeToUnits Parameters**

Parameter	Description
<b>nSizeSrcHigh</b>	Specifies the upper 31 bits of the size to be converted.
<b>nSizeSrcLow</b>	Specifies the lower 31 bits of the size to be converted.
<b>nUnitsSrc</b>	The units of nSizeSrcHigh and nSizeSrcLow. Specify one of the following values: <ul style="list-style-type: none"> <li>• <b>BYTES</b>—The value is in bytes.</li> <li>• <b>KBYTES</b>—The value is in kilobytes.</li> <li>• <b>MBYTES</b>—The value is in megabytes.</li> <li>• <b>GBYTES</b>—The value is in gigabytes.</li> <li>• <b>TBYTES</b>—The value is in terabytes.</li> </ul>
<b>nvSizeTargetHigh</b>	Returns the upper 31 bits of converted size.
<b>nvSizeTargetLow</b>	Returns the lower 31 bits of converted size.
<b>nUnitsTarget</b>	Specifies (and returns if -1 is specified) the units of nvSizeTargetHigh and nvSizeTargetLow. Specify one of the following values. (You can also specify -1 or any negative value. In this case, the function sets nUnitsTarget to the smallest units that ensures that nvSizeTargetHigh is 0): <ul style="list-style-type: none"> <li>• <b>BYTES</b>—The value is in bytes.</li> <li>• <b>KBYTES</b>—The value is in kilobytes.</li> <li>• <b>MBYTES</b>—The value is in megabytes.</li> <li>• <b>GBYTES</b>—The value is in gigabytes.</li> <li>• <b>TBYTES</b>—The value is in terabytes.</li> </ul>

## Return Values

**Table 67 • ConvertSizeToUnits Return Values**

Return Value	Description
<b>ISERR_SUCCESS</b>	Indicates that the function was successful.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was not successful.

**Table 67** • ConvertSizeToUnits Return Values (cont.)

Return Value	Description
<b>ISERR_INVALID_ARG</b>	Indicates that either nSizeSrcHigh or nSizeSrcLow is negative.

## ConvertWinHighLowSizeToISHighLowSize

The **ConvertWinHighLowSizeToISHighLowSize** function converts the unsigned 64-bit Windows size specified through nSizeWinHigh and nSizeWinLow to the corresponding 62-bit InstallShield high and low size value.

### Syntax

```
ConvertWinHighLowSizeToISHighLowSize ( nSizeWinHigh, nSizeWinLow, nvSizeISHigh, nvSizeISLow);
```

### Parameters

**Table 68** • ConvertWinHighLowSizeToISHighLowSize Parameters

Parameter	Description
<b>nSizeWinHigh</b>	Specifies the upper 31 bits of the Windows 64-bit size value.
<b>nSizeWinLow</b>	Specifies the lower 31 bits of the Windows 64-bit size value.
<b>nSizeISHigh</b>	Returns the upper 31 bits of the InstallShield 62-bit size value.
<b>nSizeISLow</b>	Returns the lower 31 bits of the InstallShield 62-bit size value.

### Return Values

**Table 69** • ConvertWinHighLowSizeToISHighLowSize Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	Indicates that the function was successful.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was not successful.

## CopyBytes

The **CopyBytes** function copies a specified number of bytes from one string to another string. You can specify the offset indices into the source and destination strings. CopyBytes is useful for working with binary files.



## Syntax

```
CopyBytes ( svDest, nIndexDest, svSrc, nIndexSrc, nCount );
```

## Parameters

**Table 70** • CopyBytes Parameters

Parameter	Description
<b>svDest</b>	Specifies the destination string.
<b>nIndexDest</b>	Specifies the offset index (starting point) in the destination string at which location the bytes are to be inserted. The first byte in the string is at position 0.
<b>svSrc</b>	Specifies the source string. Do not pass an autosized string whose size is greater than 256 characters. Instead, declare the string with an explicit size. For more information about string sizing, see <a href="#">String Size and Autosize</a> .
<b>nIndexSrc</b>	Specifies the offset index (starting point) in the source string at which location bytes begin to be copied. The first byte in the string is at position 0.
<b>nCount</b>	Specifies the total number of bytes you want to copy from svSrc to svDest. This value must be no larger than the size of svSrc - 1. For example, if svSrc was declared with a size of 512 (giving it a maximum string length of 511), then the value passed in nCount must be 511 or less.

## Return Values

**Table 71** • CopyBytes Return Values

Return Value	Description
<b>0</b>	CopyBytes successfully copied a specified number of bytes from one string to another.
<b>&lt; 0</b>	CopyBytes was unable to copy the bytes.

## CopyBytes Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the CopyBytes function.
 *
 * This example script retrieves the current date from the
 * target system. Then, it copies the year from the date and
 * displays the year to the end user.
 *
\*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_CopyBytes(HWND);

function ExFn_CopyBytes(hMSI)
    STRING svDate, svYear;
    NUMBER nvResult, nIndexDate, nIndexYear, nCount;
begin

    // Get the date from the target system.
    GetSystemInfo (DATE, nvResult, svDate);

    // Set up parameters to pass to CopyBytes. The year is
    // in the last four bytes of svDate.
    nIndexYear = 0;
    nCount      = 4;
    nIndexDate  = StrLength(svDate) - nCount;

    // Copy the four bytes representing the year into svYear.
    if (CopyBytes (svYear, nIndexYear, svDate, nIndexDate, nCount) < 0) then
        // Report an error.
        MessageBox ("CopyBytes failed.", SEVERE);
    else
        // Display the year.
        MessageBox ("The year is " + svYear, INFORMATION);
    endif;

end;

```

## CopyCHARArrayToISStringArray

### Description

The **CopyCHARArrayToISStringArray** function copies the strings from an existing array of ANSI character strings (pointed to by pCHARArray) to the existing string array specified by vArray.

### Syntax

```
CopyCHARArrayToISStringArray ( vArray, pCHARArray );
```

## Parameters

**Table 72** • CopyCharArrayToISStringArray Parameters

Parameter	Description
<b>vArray</b>	Specifies the string array to which you want to copy strings.
<b>pCharArray</b>	Specifies a pointer to an array of pointers to ANSI character strings. Typically, this pointer is returned by a previous call to <a href="#">GetCharArrayFromISStringArray</a> .

## Return Values

**Table 73** • CopyCharArrayToISStringArray Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	This function always returns ISERR_SUCCESS.

## Additional Information

If pCharArray was returned by a previous call to GetCharArrayFromISStringArray, be careful when modifying strings contained in the array. Since the length of the strings contained in string arrays are managed internally by the setup, if you change the length of a string the entire string will not be copied back to the original array when you call CopyCharArrayToISStringArray.

# CopyFile

The **CopyFile** function creates a copy of the file specified by szSrcFile. The new file is given the name specified by szTargetFile. You must specify the name of the file that you want to copy onto the target location in the szTargetFile parameter of this function in order for the function to work.

If you use this function to transfer files to [WINSYSDIR64](#), you must first disable file system redirection using [WOW64FSREDIRECTION](#). Otherwise, files being transferred to WINSYSDIR64 are incorrectly redirected to the 32-bit SysWOW64 folder. Since some Windows functionality that could be used by the installation requires that redirection be enabled to work, Windows documentation recommends that you disable redirection only for as long as necessary. It is recommended that you then enable file system redirection as soon as you have completed transferring the necessary files to WINSYSDIR64. To learn more, see Targeting 64-Bit Operating Systems with InstallScript Installations.



**Tip** • It is strongly recommended that you disable the Cancel button using the **Disable** function before calling the **CopyFile** function if the status dialog is displayed during the copy. If you do not disable the Cancel button and the end user cancels during the copy file operation, the OnCancelling event handler is not called. Instead, the copy file operation returns a failure error code, which your script must handle by calling the appropriate event and then relaunching the copy file operation. You can enable and disable the Cancel button using the [Enable](#) and [Disable](#) functions.



**Note** • For file transfer, [XCopyFile](#) is an alternative to **CopyFile**. **XCopyFile** can perform version checking, mark locked .dll and .exe files for update after system restart, and increment registry reference counters for shared .dll and .exe files. You can also use **XCopyFile** to include subdirectories.

If you use unqualified file names and set values for *SRCDIR* and *TARGETDIR* when using **CopyFile**, save the current values using [VarSave](#) before calling **CopyFile** and then restore them using [VarRestore](#).

If the target directory does not exist, **CopyFile** creates it.


You cannot rename groups of files by using wildcards with **CopyFile**. You can, however, rename a single file using **CopyFile**.

## Syntax

```
CopyFile ( szSrcFile, szTargetFile );
```

## Parameters

**Table 74 • CopyFile Parameters**

Parameter	Description
<b>szSrcFile</b>	<p>Specify the name of the file to copy. If the file name is qualified—that is, if it includes a path—<b>CopyFile</b> copies the file from the specified location. If szSrcFile contains an unqualified file name—without path information—<b>CopyFile</b> copies from the directory that is identified by the system variable <a href="#">SRCDIR</a>. To copy groups of files, use wild card characters in this parameter.</p> <p>You can specify a valid URL in this parameter. If you pass a CGI or ASP request (for example, "http://www.mydomain.net/login.asp?name=Me&amp;pwd=wow"), the response is sent to the file that is specified in the szTargetFile parameter. When passing a URL, do not include wildcard characters. To check the validity of a URL, call the following:</p> <pre>Is (VALID_PATH, szURL);</pre>
<b>szTargetFile</b>	<p>Specify the name to give to the copy of the file that is identified by szSrcFile. If the file name is qualified—if it includes a path—<b>CopyFile</b> copies the file to the location specified by the path. If szSrcFile contains an unqualified file name—without path information—the copy is created in the directory specified by the system variable TARGETDIR (in InstallScript installations) or INSTALLDIR (in Basic MSI and InstallScript MSI installations). If the target directory does not exist, it is created.</p> <p>You cannot specify a URL in this parameter. If you do, the function fails and returns ISERR_INVALID_ARG.</p> <div>  <p><b>Note</b> • When a wildcard character is included in the file name that is specified by szSrcFile, the file name part of szTargetFile is not ignored; instead, the szTargetFile value is treated as the target path to which each source file is copied to with its existing name. For example, the following code copies files to a folder named <b>File.txt</b> on the D drive:</p> <pre>CopyFile ("C:\\*.\"", "D:\\File.txt");</pre> <p>If szTargetFile specifies an unqualified file name, the files are copied to the directory that is specified by TARGETDIR (in InstallScript installations) or <b>INSTALLDIR</b> (in Basic MSI and InstallScript MSI installations). For that reason, <b>CopyFile</b> cannot be used to copy and rename a group of files. The source and target directories must be different when szSrcFile contains one or more wildcard characters.</p> </div>

## Return Values

**Table 75 • CopyFile Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully copied the files from source to target directory.
<b>ISERR_INVALID_ARG</b>	Indicates that an invalid argument was passed to the function.
<b>All other negative values</b>	Indicates that the function was unable to copy the requested file.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

## Additional Information

After you modify .ini files with [WriteProfString](#) or [WriteProfInt](#), you must flush the cache buffer under before using **CopyFile**. All .ini files are cached. This behavior can cause a delay in writing changes to the specified files. This delay can interfere with subsequent file operations. To avoid this problem, call **WriteProfString** with null parameters to force Windows to write the data to the .ini file immediately, as shown below:

```
WriteProfString ("C:\\Test.ini", "Windows",
"KeyboardDelay", "100");

// null string ("" ) for all four parameters
WriteProfString ("", "", "", "");

// CopyFile should now have access to updated file.
CopyFile ("C:\\Test.ini", "C:\\Temp\\Test.ini");
```

## CopyFile Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the CopyFile function.
*
* This script copies files in the directory specified by
* SOURCE_DIR to the directory specified by TARGET_DIR.
*
* Note: Before running this script, you must set the
*       preprocessor constants to existing paths on
*       the target system.
*
\*-----*/

#define SOURCE_DIR "C:\\Source"
```

```

#define TARGET_DIR "C:\\Target"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_CopyFile(HWND);

function ExFn_CopyFile(hMSI)
    NUMBER nResult;
begin

    // Copy all files in the source directory, including files
    // in subdirectories, to the target directory.
    nResult = CopyFile(SOURCE_DIR ^ ".*", TARGET_DIR ^ ".*");

    // Report the results of the copy operation.
    switch (nResult)
    case 0:
        MessageBox ("Files successfully copied.", INFORMATION);
    case COPY_ERR_CREATEDIR:
        MessageBox ("A target directory could not be created.", SEVERE);
    case COPY_ERR_MEMORY:
        MessageBox ("Insufficient memory.", SEVERE);
    case COPY_ERR_NODISKSPACE:
        MessageBox ("Insufficint disk space.", SEVERE);
    case COPY_ERR_OPENINPUT:
        MessageBox ("Unable to open the input files in "+ SOURCE_DIR +".",
            SEVERE);
    case COPY_ERR_OPENOUTPUT:
        MessageBox ("Unable to copy the source files.", SEVERE);
    case COPY_ERR_TARGETREADONLY:
        MessageBox ("A target file already exists and cannot be overwritten.",
            SEVERE);
    default:
        MessageBox ("An unspecified error occurred.", SEVERE);
    endswitch;

end;

```

## CreateDir

The **CreateDir** function creates one or more subdirectories on the target drive. You can use a path that contains subdirectories on more than one level, such as **C:\Programs\Winapps\Myapp**. If any subdirectory in the path does not exist, CreateDir creates it. For example, if neither **C:\Programs\Winapps** nor **C:\Programs\Winapps\Myapp** exists, CreateDir creates both subdirectories.



**Caution** • CreateDir fails under the following conditions:

- The path is illegal.
- The drive or any subdirectory in the path is write-protected.
- The drive name is invalid.
- You do not have network privileges to create subdirectories.



## Syntax

```
CreateDir ( szDirPath );
```

## Parameters

**Table 76 • CreateDir Parameters**

Parameter	Description
<b>szDirPath</b>	Specifies the fully qualified path of the subdirectory to create. Separate each level in the path with a backslash by using the backslash escape character (\\).

## Return Values

**Table 77 • CreateDir Return Values**

Return Value	Description
<b>0</b>	Indicates either that the function successfully created the specified directory on the target drive or that the specified directory already exists.
<b>&lt; 0</b>	Indicates that the directory does not already exist and that the function was unable to create it. You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

## CreateDir Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the CreateDir function.
*
* The user is asked to input a valid directory. If the
* specified directory does not exist, CreateDir is called to
* create it. Then CreateDir is called again to create a
* multilevel directory structure beneath the specified
* directory.
*
```

```

\*-----*/

#define DEFAULT_DIR "C:\\ISExampl"
#define SUBDIRS     "N_Dir\\A_Dir"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_CreateDir(HWND);

function ExFn_CreateDir(hMSI)
    STRING svPath;
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Prompt user for a directory to be created.
    AskPath ("Please enter a valid path.", DEFAULT_DIR, svPath);

    // Check to see if that directory already exists.
    if (ExistsDir (svPath) != EXISTS) then

        // The directory does not exist; create it.
        if (CreateDir (svPath) < 0) then
            // Report the error; then abort.
            MessageBox ("Unable to create directory", SEVERE);
            abort;
        else
            // Report success
            sprintfBox (INFORMATION, "CreateDir", "%s created.", svPath);
        endif;

    endif;

    // Create an entire multilevel directory structure
    // beneath the selected directory.
    if (CreateDir (svPath ^ SUBDIRS) < 0) then
        MessageBox ("Failed to create subdirectories beneath" + svPath + ".",
            WARNING);
    else
        sprintfBox (INFORMATION, "CreateDir", "%s created beneath %s.", SUBDIRS,
            svPath);
    endif;

end;

```

## CreateFile

The **CreateFile** function creates a new file. If a file with the same name exists, CreateFile overwrites it. Before you create a file with CreateFile, you must set the file mode with [OpenFileMode](#).

CreateFile leaves the newly created file open in read/write (binary file) or append (text file) mode so you can read from or write to the file using other functions such as [GetLine](#), [ReadBytes](#), [WriteLine](#), and [WriteBytes](#). To write to an existing file, you must first open the file in FILE\_MODE\_APPEND mode using the [OpenFileMode](#) and [OpenFile](#) functions.



**Note** • In addition to read/write or append mode, all newly created files automatically open in `OF_SHARE_DENY_NONE` mode. This means that the files are opened without denying other programs read or write access to the files. This mode has its roots in the Windows API `OpenFile`.


When you finish reading from and writing to a file, you must close the file using the [CloseFile](#) function.

## Syntax

```
CreateFile ( nvFileHandle, szPath, szFileName );
```

## Parameters

**Table 78** • CreateFile Parameters

Parameter	Description
<b>nvFileHandle</b>	Returns the handle of the new file.
<b>szPath</b>	Specifies the fully qualified path of the subdirectory in which to create the new file.
	 <p><b>Note</b> • <i>CreateFile fails if the folder specified in szPath does not exist. You can test for the existence of a folder by calling <code>ExistsDir</code> and create a folder by calling <code>CreateDir</code>.</i></p>
<b>szFileName</b>	Specifies the name of the file to create.

## Return Values

**Table 79** • Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully created the new file.
<b>&lt; 0</b>	Indicates that the function was unable to create the specified file.

## Additional Information

The actions of the `CreateFile` function are not logged for uninstallation when logging is enabled. If you want a file that is created by `CreateFile` to be logged for uninstallation, transfer a starter file with the file name you want to the target system using [XCopyFile](#) while logging is enabled. You enable and disable logging with the [Enable](#) and [Disable](#) functions. `XCopyFile` actions are logged when logging is enabled, so the starter file is logged for uninstallation. After transferring the logged starter file, you can write to or overwrite the logged starter file using `CreateFile` and other file-related functions. The file name must remain unchanged; otherwise, it is not found during uninstallation.

## CreateFile Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the CreateFile and WriteLine functions.
*
* CreateFile is called to create a file to store a string. The
* string is written into the file by the WriteLine function.
*
* Note: Before running this script, set the preprocessor
*       constant EXAMPLE_DIR so that it references an existing
*       directory on the target system. Note that if the file
*       specified by EXAMPLE_FILE already exists, it will be
*       overwritten.
*
\*-----*/

#define EXAMPLE_DIR "C:\\\"
#define EXAMPLE_FILE "ISExempl.txt"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_CreateFile(HWND);

function ExFn_CreateFile(hMSI)
    STRING  szTitle, szMsg;
    NUMBER  nvFileHandle;
begin

    // Set the file mode to append.
    OpenFileMode (FILE_MODE_APPEND);

    // Create a new file and leave it open.
    if (CreateFile (nvFileHandle, EXAMPLE_DIR, EXAMPLE_FILE) < 0) then
        // Report the error.
        MessageBox ("CreateFile failed.", SEVERE);
        abort;
    else
        // Set the message to write to the file.
        szMsg = "This line was appended by an example InstallShield script.";

        // Append the message to the file.
        if (WriteLine(nvFileHandle, szMsg) < 0) then
            // Report the error.
            MessageBox ("WriteLine failed.", SEVERE);
        else
            // Report success.
            szTitle = "CreateFile & WriteLine";
            szMsg  = "Successfully created and wrote to %s.";
            sprintfBox (INFORMATION, szTitle, szMsg, EXAMPLE_FILE);
        end if;
    end if;
end function
```

```

        endif;

    endif;

    // Close the file.
    CloseFile (nvFileHandle);

end;

```

## CreateInstallationInfo

In an event-based script, the **CreateInstallationInfo** function is called automatically after the First UI Before event. It uses the system variables [IFX\\_KEYPATH\\_PRODUCT\\_INFO](#) and [IFX\\_PRODUCT\\_KEY](#) to create an application information key and a per application paths key for the program you are installing. The application information key is created immediately as a result of calling CreateInstallationInfo. The per application paths key is not created until a subsequent call to RegDBSetItem sets a [Path] or [DefaultPath] value under that key.

CreateInstallationInfo provides the product name for display in the Welcome dialog as well as the company name, product name, and version number that MaintenanceStart uses to initialize the uninstallation log file. MaintenanceStart will fail if CreateInstallationInfo is not called before it in the script.

Call CreateInstallationInfo only once in a setup. If you are launching multiple installations using DoInstall, each installation can of course have its own call to CreateInstallationInfo.

CreateInstallationInfo is a special registry-related function, designed to work with certain predefined registry keys. For more information, see [Special Registry-Related Functions](#).




---

**Note** • The InstallScript engine currently does not support writing or reading Add or Remove Programs information for a product in the 64-bit part of the registry. Therefore, using the REGDB\_OPTION\_WOW64\_64KEY option with the [REGDB\\_OPTIONS](#) system variable is not supported for this registry function. Enabling the REGDB\_OPTION\_WOW64\_64KEY option has no effect on where registry entries are created by this function.

### Syntax

```
CreateInstallationInfo ( );
```

### Parameters

None

## Return Values

**Table 80 • CreateInstallationInfo Return Values**

Return Value	Description
0	Indicates that the function was successful.
< 0	Indicates that the function failed to create one or more of the registry keys  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

## Additional Information

- You must call `CreateInstallationInfo` before calling [RegDBSetItem](#) or [RegDBGetItem](#). See the descriptions of those functions for further information.
- The per application paths key is not actually created until `RegDBSetItem` is called to set a value under that key.
- `CreateInstallationInfo` no longer fails when `IFX_COMPANY_NAME`, `IFX_PRODUCT_NAME`, or `IFX_PRODUCT_VERSION` are empty. `CreateInstallationInfo` fails if a remote registry is connected, `IFX_KEYPATH_PRODUCT_INFO` is empty, or the application information registry key cannot be created.

# CreateObject



**Project** • In *InstallScript* projects, the **CreateObject** function has been replaced by the [CoCreateObject](#).

The **CreateObject** function initializes the registered COM object named by `szProgID` and returns a reference that can be assigned to a variable of type `OBJECT` by using the `set` keyword.

To check whether the object was initialized successfully, you can use the keywords `try-catch-endcatch` for more control over exception handling for COM objects.



**Note** • The COM object must be registered on the target system in order for `CreateObject` to work.

## Syntax

```
CreateObject ( szProgID );
```

## Parameters

**Table 81** • CreateObject Parameters

Parameter	Description
<b>szProgID</b>	Specifies the ProgID of the COM object to be initialized.

## Return Values

A reference that can be assigned to a variable of type OBJECT by using the set keyword.

# CreateProgramFolder

The **CreateProgramFolder** function creates a new folder on the target system. The folder is created in the Start Programs menu.

## Syntax

```
CreateProgramFolder ( szFolderName );
```

## Parameters

**Table 82** • CreateProgramFolder Parameters

Parameter	Description
<b>szFolderName</b>	Specifies the name of the folder to add to the target system.

## Return Values

**Table 83** • CreateProgramFolder Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully added the folder to the target system or that the folder already existed.
<b>&lt; 0</b>	Indicates that the function was unable to add the specified program folder.

## CreateProgramFolder Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the CreateProgramFolder function.
 *
 * This script creates a program folder named ExampleFolder on
 * the target system.
 *
 \*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_CreateProgramFolder(HWND);

function ExFn_CreateProgramFolder(hMSI)
    STRING  szFolderName, szTitle, szMsg;
begin

    // Set up parameters for call to CreateProgramFolder.
    szFolderName = "ExampleFolder";
    szTitle      = "CreateProgramFolder";
    szMsg        = "%s created successfully.";

```



```

// Create the program folder.
if (CreateProgramFolder (szFolderName) < 0) then
    MessageBox ("Unable to create program folder", SEVERE);
else
    sprintf (INFORMATION, szTitle, szMsg, szFolderName);
endif;

end;

```

## CreateRegistrySet

The **CreateRegistrySet** function creates the registry entries specified by one or all registry sets in the current media that are *not* associated with a particular component. (The name of the current media is stored in the system variable [MEDIA](#).)

It is not necessary to call **CreateRegistrySet** for registry sets that are associated with one or more components since this function has no effect on these registry sets. Registry entries stored in registry sets that are associated with components are created when the component itself is installed (for example, via a **FeatureTransferData** call).



**Note** • When you enable the [REGDB\\_OPTION\\_WOW64\\_64KEY](#) option, this affects where registry entries from registry sets are created. For example, if this option is enabled when you call the **CreateRegistrySet** function, the registry set is created in the 64-bit part of the registry. Note that this includes the default registry set which is created during file transfer immediately after the **OnMoving** event returns, and registry sets associated with components which are automatically created when the component is installed. Therefore, to avoid the data from these registry sets being created in the 64-bit portion of the registry unintentionally, it is recommended that you set this option only during the calling of the appropriate registry APIs in your script, then disable the option before continuing with the script.

### Syntax

```
CreateRegistrySet (szRegistrySet);
```

## Parameters

**Table 84** • CreateRegistrySet Parameters

Parameter	Description
szRegistrySet	Specifies the name of a registry set that is <i>not</i> associated with a particular component in the current media. To create all registry sets that are defined in the current media and are not associated with a particular component, pass a null string ("" ) in this parameter.

## Return Values

**Table 85** • CreateRegistrySet Return Values

Return Value	Description
0	Indicates that the function was successful.
< 0	Indicates that an unspecified error has occurred.

## Additional Information

- The value of the system variable MEDIA is set to 'DATA' during setup initialization. If you change the value of this variable to refer to a script-created feature set, you must change the value back to 'DATA' before calling **CreateRegistrySet**.
- If a registry set is associated only with components that are not included in the built media (because either they are not included in any features, they are included only in features whose Include in Build property is set to No, or they are included only in features that you have excluded from the built media using the Release Wizard's Features panel), the registry set's entries are not automatically created during file transfer and can be installed by a call to **CreateRegistrySet**.

## CreateRegistrySet Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the CreateShellObjects and CreateRegistrySet
 * functions.
 *
 * Note: To run this example, you must create a project that
 *       specifies registry entries and shell objects in the
 *       Resources pane of the IDE.
 *
 \*-----*/

export prototype ExFn_CreateRegistrySet(HWND);

function ExFn_CreateRegistrySet(hMSI)
    NUMBER ndisk;

```

```

    STRING szPassword;
    STRING svDir;
begin

    // Set the default target for copying files.
    svDir = "C:\\temp";

    // Get a target location from the user.
    AskDestPath ("", "", svDir, 0);

    // Assign the specified target location to
    // the corresponding system variable.
    INSTALLDIR = svDir;

    // Enable the progress indicator.
    SetStatusWindow (0, "");
    Enable (STATUS);
    StatusUpdate (ON, 100);

    // Transfer files.
    if (ComponentMoveData (MEDIA,ndisk,0) < 0 ) then
        MessageBox ("Error in moving data", SEVERE);
        abort;
    endif;

    // Create the Registry sets that were defined in the Resources pane.
    if (CreateRegistrySet ("") < 0) then
        MessageBox ("Unable to create registry set.", SEVERE);
        abort;
    endif;

    // Create the Shell objects that were defined in the Resources pane.
    if (CreateShellObjects ("") < 0) then
        MessageBox ("Unable to create shell objects.", SEVERE);
        abort;
    endif;

end;

```

## CreateShellObjects

The **CreateShellObjects** function creates shortcuts that are included in the current media, but are *not* associated with a particular component. The function also creates the folders that contain the created shortcuts if they do not already exist on the target system. (The name of the current media is stored in the system variable [MEDIA](#).)

It is not necessary to call CreateShellObjects for shortcuts that are associated with one or more components because this function has no effect on these shortcuts. Shortcuts that are associated with components are created when the component itself is installed (for example, via a [FeatureTransferData](#) call). When the shortcuts are created, the folders containing the shortcuts will also be created, if necessary.



**Caution** • This function will have no effect if called from an InstallShield object's script. The creation of shell objects is not supported in an InstallShield Object project.



**Note** • `CreateShellObjects` will not create an empty folder on the target system: If a folder in the Shortcuts view is empty, this function will not create it. This restriction also applies to the shortcuts and corresponding folders that are automatically created when you call `FeatureTransferData`. To create an empty folder, call the [CreateProgramFolder](#) function in your script.

## Syntax

```
CreateShellObjects (szReserved);
```

## Parameters

**Table 86** • CreateShellObjects Parameters

Parameter	Description
<b>szReserved</b>	Pass a null string ("" ) in this parameter. No other value is allowed.

## Return Values

**Table 87** • CreateShellObjects Return Values

Return Value	Description
<b>0</b>	Indicates that the function was successful.
<b>&lt; 0</b>	Indicates that an unspecified error has occurred.

## Additional Information

The value of the system variable MEDIA is set to 'DATA' during setup initialization. If you change the value of this variable to refer to a script-created component set, you must change the value back to 'DATA' before calling `CreateShellObjects`.

This function should be called only after `FeatureTransferData` has been called.

## CreateShellObjects Example

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the CreateShellObjects and CreateRegistrySet
 * functions.
 *
 * Note: To run this example, you must create a project that
 *       specifies registry entries and shell objects in the
 *       Resources pane of the IDE.
 *
```

```

\*-----*/

export prototype ExFn_CreateShellObjects(HWND);

function ExFn_CreateShellObjects(hMSI)
    NUMBER ndisk;
    STRING szPassword;
    STRING svDir;
begin

    //Set the default target for copying files.
    svDir = "C:\\temp";

    // Get a target location from the user.
    AskDestPath (""," ",svDir,0);

    // Assign the specified target location to
    // the corresponding system variable.
    INSTALLDIR = svDir;

    // Enable the progress indicator.
    SetStatusWindow (0, "");
    Enable (STATUS);
    StatusUpdate (ON, 100);

    // Transfer files.
    if (ComponentMoveData (MEDIA,ndisk,0) < 0 ) then
        MessageBox ("Error in moving data",SEVERE);
        abort;
    endif;

    //Create the Registry sets that were defined in the Resources pane.
    if (CreateRegistrySet ("") < 0) then
        MessageBox ("Unable to create registry set.",SEVERE);
        abort;
    endif;

    //Create the Shell objects that were defined in the Resources pane.
    if (CreateShellObjects ("") < 0) then
        MessageBox ("Unable to create shell objects.",SEVERE);
        abort;
    endif;

end;

```

## CreateShortcut

The **CreateShortcut** function lets you perform tasks such as the following:

- Add a shortcut or program folder to locations such as the Start menu, the Programs menu, or the desktop. Use the szShortcutFolder parameter to specify the appropriate location for the shortcut.
- Create a cascading submenu on the Startup menu, and include a shortcut in the submenu.
- Set Windows Shell properties for a shortcut to configure behavior such as disabling the ability to pin the shortcut to the Start menu.



---

**Note** • *The shortcut target must be present on the target system before **CreateShortcut** can be called.*


**CreateShortcut** does not support the creation of Internet shortcuts.

## Syntax



```
CreateShortcut (szShortcutFolder, szName, szCommandLine, szWorkingDir, szIconPath, nIcon, szShortCutKey,  
               nFlag);
```

## Parameters

**Table 88 • CreateShortcut Parameters**

Parameter	Description
<b>szShortcutFolder</b>	<p>Specify the name of the folder that should contain the shortcut, or specify the name of the program folder that you want to create. If the folder does not exist, the installation creates it. For this parameter, you can specify a subfolder in a multi-level cascading menu. If the subfolder does not exist, <b>CreateShortcut</b> creates the subfolder and, if necessary, its parent folders.</p> <p>To add the shortcut to a specific folder, specify the fully qualified path—for example:</p> <p><b>C:\\ProgramData\\Microsoft\\Windows\\Start Menu\\Programs</b></p> <p>To add a shortcut to the Programs menu on the Start menu, you can pass a null string ("") in this parameter.</p> <p>You can pass one of the following InstallScript system variables in this parameter:</p> <ul style="list-style-type: none"> <li>● <b>FOLDER_DESKTOP</b>—Adds the shortcut to the desktop.</li> <li>● <b>FOLDER_STARTUP</b>—Adds the shortcut to the Startup menu.</li> <li>● <b>FOLDER_STARTMENU</b>—Adds the shortcut to the Start menu.</li> <li>● <b>FOLDER_PROGRAMS</b>—Adds the shortcut to the Start\\Programs menu.</li> </ul> <p>You can also specify a path relative to a folder that is identified by an InstallScript system variable—for example:</p> <p><b>FOLDER_PROGRAMS ^ "ACCESSORIES\\GAMES"</b></p>
<b>szName</b>	<p>Specify the name of the shortcut. Calling <b>CreateShortcut</b> to add a shortcut to a program folder also creates a link file in the links directory that is specified by szCommandLine. Note that Windows Shell does not allow the following characters in names: /, \\, :, ?, &lt;, &gt;, or  .</p>
<b>szCommandLine</b>	<p>Specify one of the following:</p> <ul style="list-style-type: none"> <li>● The fully qualified name of the executable file that is associated with the shortcut, including any command-line parameters. This is added to the Target value on the shortcut's Properties dialog box. To add a shortcut to the Start Programs menu, enter the fully qualified path of the links directory, which is where your application stores its icon link files.</li> <li>● The fully qualified path if szName is a subfolder.</li> </ul> <p> <b>Caution</b> • If the command line includes a long file name, it must be enclosed in quotes. Command-line parameters, however, should not be surrounded with quotation marks. For that reason, it is advisable to build the szCommandLine string from two separate strings.</p>

**Table 88 • CreateShortcut Parameters (cont.)**

Parameter	Description
<b>szWorkingDir</b>	<p>Specify the working directory for the shortcut target.</p> <p>If szName is a subfolder, this parameter is not applicable.</p> <p><b>CreateShortcut</b> writes this directory in the Start In box on the Shortcut tab of the shortcut's Properties dialog box. If you pass a null string ("") in this parameter, the function leaves this Start In box blank, and the path in the Target box is used.</p> <p></p> <p><b>Caution •</b> Do not call <b>LongPathToQuote</b> to enclose this path in quotation marks. InstallShield automatically encloses these paths in quotation marks.</p>
<b>szIconPath</b>	<p>Specify the fully qualified path to the file that contains the icon that you want to be displayed for the shortcut.</p> <p>If szItemName is a subfolder, this parameter is not applicable.</p> <p></p> <p><b>Caution •</b> Do not call <b>LongPathToQuote</b> to enclose this path in quotation marks. InstallShield automatically encloses these paths in quotation marks.</p>
<b>nlIcon</b>	<p>Specify the icon index in the executable file that is specified by szIconPath. An icon index of 0 refers to the first icon in the file, an icon index of 1 refers to the second icon, and so on. If you are not using an icon, specify <b>0</b> in this parameter.</p> <p>If szName is a subfolder, this parameter is not applicable.</p>
<b>szShortCutKey</b>	<p>Specify the shortcut key (in the form of a string) that you want to be assigned to your shortcut. You can set szShortCutKey for the shortcut so that end users can press the appropriate hot keys to launch the shortcut.</p> <p>For example, if you want end users to be able to launch the product by pressing the CTRL key, the ALT key, and then the 1 key on the numeric keyboard, pass "Ctrl + Alt + 1" in this parameter.</p> <p>If szName is a subfolder, this parameter is not applicable.</p>



**Table 88 • CreateShortcut Parameters (cont.)**

Parameter	Description
<b>nFlag</b>	<p>Pass one or more of the following predefined constants in this parameter. To pass two or more predefined constants in this parameter, combine those constants with the bitwise OR operator (   ).</p> <ul style="list-style-type: none"> <li>• <b>CS_OPTION_FLAG_REPLACE_EXISTING</b>—Replace an existing shortcut.</li> <li>• <b>CS_OPTION_FLAG_RUN_MAXIMIZED</b>—The target of the shortcut is maximized when launched.</li> <li>• <b>CS_OPTION_FLAG_RUN_MINIMIZED</b>—The target of the shortcut is minimized when launched.</li> <li>• <b>CS_OPTION_FLAG_PREVENT_PINNING</b>—Do not allow the shortcut to be pinned to the Start menu or taskbar on Windows 7 or later systems. This option hides the context menu commands that enable end users to pin the shortcut to the taskbar and to the Start menu.  You may want to prevent pinning for shortcuts that are for tools and secondary products that are part of your installation.</li> <li>• <b>CS_OPTION_FLAG_NO_NEW_INSTALL_HIGHLIGHT</b>—Do not highlight the shortcut as newly installed after end users install your product on Windows 7 or later systems. This has the same effect as clearing the <b>Highlight newly installed programs</b> check box in the Customize Start Menu dialog box for an individual item on a target system.  You may want to use this option for shortcuts that are for tools and secondary products that are part of your installation.</li> <li>• <b>CS_OPTION_FLAG_NO_STARTSCREEN_PIN</b>—Do not pin the shortcut to the Start screen by default on Windows 8 target systems. If you pass this constant, the installation sets a Windows Shell property that was introduced in Windows 8.  You may want to prevent pinning for shortcuts that are for tools and secondary products that are part of your installation.</li> <li>• <b>NULL</b>—Indicates no options.</li> </ul> <p>For more information on CS_OPTION_FLAG_PREVENT_PINNING and CS_OPTION_FLAG_NO_STARTSCREEN_PIN, see the <a href="#">Additional Information section</a>.</p>

## Return Values

**Table 89 • CreateShortcut Return Values**

Return Value	Description
0	Indicates that the function successfully added or replaced the shortcut in the specified folder and associated the executable file with it.
< 0	<p>Indicates that the function was unable to add or replace the shortcut and associate the executable file with it.</p> <p>You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a>.</p>

## Additional Information

Note the following details about two of the nFlag constants.

### CS\_OPTION\_FLAG\_PREVENT\_PINNING

If you configure the shortcut to prevent pinning to the taskbar and the Start menu, the target of the shortcut is ineligible for inclusion in the most frequently used list on the Start menu.

Shortcuts that contain certain strings cannot be pinned to the taskbar or the Start menu, and they cannot be displayed in the most frequently used list. Examples are:

- Documentation
- Help
- Install
- Remove
- Setup
- Support

### CS\_OPTION\_FLAG\_NO\_STARTSCREEN\_PIN

Note that Windows 8 maintains information about shortcut pinning to the Start screen after a shortcut is removed by uninstalling the application. Therefore, the CS\_OPTION\_FLAG\_NO\_STARTSCREEN\_PIN constant has no effect on the target system if the shortcut has already been installed on it. Thus, when you are testing this functionality, ensure that you test on a clean machine—one on which this shortcut and its target have never been installed.

## CreateShortcut Examples

The following examples demonstrate how to use **CreateShortcut**:

- [Place a shortcut to an executable file on the Start menu and the Start Programs menu.](#) (**CreateShortcut** Example 1)
- [Create a cascading submenu on the Startup menu and add a shortcut to the menu.](#) (**CreateShortcut** Example 2)

- Place a subfolder on the desktop and a shortcut pointing to an executable file in the new folder.  
(**CreateShortcut** Example 3)

## CreateShortcut Example 1



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the CreateShortcut function.
*
* This example places a shortcut to an executable file on the
* Start menu and the Start Programs menu.
*
* Note: Before running this script, set the preprocessor
*       constants so that they reference the fully qualified
*       names of the Windows Notepad executable file and
*       a valid text file on the target system.
*
\*-----*/

#define PROGRAM "C:\\Windows\\Notepad.exe"
#define PARAM   "C:\\Windows\\Readme.txt"

function OnFirstUIAfter()
    STRING szShortcutFolder, szName, szCommandLine, szWorkingDir;
    STRING szShortCutKey, szProgram, szParam, szIconPath;
    NUMBER nIcon;
begin

    // Set up parameters for call to CreateShortcut.
    szShortcutFolder = FOLDER_STARTMENU;
    szName           = "Notepad Example 1";
    szProgram        = PROGRAM;
    szParam          = PARAM;

    LongPathToQuote (szProgram, TRUE);

    LongPathToShortPath (szParam);

    szCommandLine = szProgram + " " + szParam;
    szWorkingDir  = "";
    szIconPath    = "";
    nIcon         = 0;
    szShortCutKey = "";

    // Add a shortcut to the Start menu.
    if (CreateShortcut (szShortcutFolder, szName, szCommandLine, szWorkingDir, szIconPath,
                       nIcon, szShortCutKey, CS_OPTION_FLAG_REPLACE_EXISTING) < 0) then
        MessageBox ("CreateShortcut failed.", SEVERE);
    else
        sprintfBox (INFORMATION, "CreateShortcut", "%s created successfully.",
```

```

        szName);
endif;

szShortcutFolder = "";
szName           = "Another Notepad Example";

// Add a shortcut to the Programs menu.
if (CreateShortcut (szShortcutFolder, szName, szCommandLine, szWorkingDir, szIconPath,
                    nIcon, szShortCutKey, CS_OPTION_FLAG_REPLACE_EXISTING) < 0) then
    MessageBox ("CreateShortcut failed.", SEVERE);
else
    sprintfBox (INFORMATION, "CreateShortcut", "%s created successfully.",
                szName);
endif;

end;

```

## CreateShortcut Example 2



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the CreateShortcut function.
*
* This example creates a cascading submenu on the Startup menu
* and adds a shortcut for an executable file to it.
*
* Note: Before running this script, set the preprocessor
*       constants so that they reference the fully qualified
*       names of the Windows Notepad executable file and a
*       valid text file on the target system.
*
\*-----*/

#define PROGRAM "C:\\Windows\\Notepad.exe"
#define PARAM   "C:\\Windows\\Readme.txt"

function OnFirstUIAfter()
    STRING  szShortcutFolder, szName, szCommandLine, szWorkingDir;
    STRING  szIconPath, szShortCutKey, szProgram, szParam;
    NUMBER  nIcon, nFlag, nResult;
begin

    // Set the fully qualified name of the Startup submenu.
    szShortcutFolder = FOLDER_STARTUP ^ "SubMenu Example";

    // Construct the shortcut's command-line property.
    szProgram = PROGRAM;
    szParam   = PARAM;

    LongPathToQuote (szProgram, TRUE);

```

```

LongPathToShortPath (szParam);

szCommandLine = szProgram + " " + szParam;

// Set up the shortcut's other properties to pass to CreateShortcut.
szName       = "Notepad Example2";
szWorkingDir = "";
szIconPath   = "";
nIcon        = 0;
szShortCutKey = "";
nFlag        = CS_OPTION_FLAG_REPLACE_EXISTING|CS_OPTION_FLAG_RUN_MAXIMIZED;

// Add the shortcut to the submenu; create the submenu if necessary.
nResult = CreateShortcut (szShortcutFolder, szName, szCommandLine,
                        szWorkingDir, szIconPath, nIcon,
                        szShortCutKey, nFlag);

// Report the results.
if (nResult < 0) then
    MessageBox ("CreateShortcut failed.", SEVERE);
else
    sprintfBox (INFORMATION, "CreateShortcut", "%s created successfully.",
                szName);
endif;

end;

```

## CreateShortcut Example 3



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the CreateShortcut function.
*
* This example places a subfolder on the desktop and a shortcut
* pointing to an executable file in the new folder. The folder is
* a shortcut that points to an actual directory. From this
* folder, the end user can launch a shortcut that runs the program.
*
* Note: Before running this script, set the preprocessor
*       constants so that they reference the fully qualified
*       names of the Windows Notepad executable file and a valid
*       text file on the target system.
*
\*-----*/

#define FOLDER      "C:\\Windows\\"
#define PROGRAM     "C:\\Windows\\Notepad.exe"
#define PARAM       "C:\\Windows\\Readme.txt"

function OnFirstUIAfter()
    STRING  szShortcutFolder, szName, szCommandLine, szWorkingDir;

```

```

    STRING  szIconPath, szShortCutKey;
    STRING  szProgram, szParam, szFolderDir;
    NUMBER  nIcon, nFlag, nResult;
begin

    // szShortcutFolder is the Desktop on the local system.
    szShortcutFolder = FOLDER_DESKTOP;
    szName           = "Example folder 3";

    // Create the folder to which the folder icon will point.
    szFolderDir = FOLDER ^ szName;
    CreateDir(szFolderDir);

    // The command line for the folder icon must be the folder path, and
    // it must be enclosed in quotation marks if the path is longer than
    // eight characters.

    szCommandLine = szFolderDir;
    LongPathToQuote(szCommandLine, TRUE);

    szWorkingDir  = "";
    szIconPath    = "";
    nIcon         = 0;
    szShortCutKey = "";
    nFlag         = CS_OPTION_FLAG_REPLACE_EXISTING|CS_OPTION_FLAG_RUN_MINIMIZED;

    // Create the folder shortcut, and show the folder it points to.
    nResult = CreateShortcut (szShortcutFolder, szName, szCommandLine,
                             szWorkingDir, szIconPath, nIcon, szShortCutKey,
                             nFlag);

    if (nResult < 0) then
        MessageBox ("CreateShortcut failed.", SEVERE);
    else
        sprintfBox (INFORMATION, "CreateShortcut", "%s created successfully.",
                   szName);
    endif;

    // Display the folder just created.
    ShowProgramFolder (szFolderDir, SW_SHOW);

    // Add the example shortcut to the newly created folder.
    szShortcutFolder = szFolderDir;
    szName           = "Notepad Example 3";

    // Make sure the white space is not seen as a delimiter.
    szProgram        = PROGRAM;
    LongPathToQuote (szProgram, TRUE);

    szParam = PARAM;
    LongPathToShortPath (szParam);

    szCommandLine = szProgram + " " + szParam;
    szWorkingDir  = "";
    szIconPath    = "";
    nResult = CreateShortcut (szShortcutFolder, szName, szCommandLine,
                             szWorkingDir, szIconPath, nIcon, szShortCutKey,
                             nFlag);

```

```

if (nResult < 0) then
    MessageBox ("CreateShortcut failed.", SEVERE);
else
    sprintfBox (INFORMATION, "CreateShortcut", "%s created successfully.",
                szName);
endif;

end;

```

## CreateShortcutFolder

The **CreateShortcutFolder** function creates a new folder on the target system. The folder is created on the Start Programs menu. If the folder already exists, it is highlighted.

### Syntax

```
CreateShortcutFolder (szShortcutFolder);
```

### Parameters

**Table 90** • CreateShortcutFolder Parameters

Parameter	Description
<b>szShortcutFolder</b>	Specify the name of the folder to add to the target system.

### Return Values

**Table 91** • CreateShortcutFolder Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully added the folder to the target system or that the folder already existed.
<b>&lt; 0</b>	Indicates that the function was unable to add the specified program folder.

## CreateShortcutFolder Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the CreateShortcutFolder function.
 *
 * This script creates a program folder named ExampleFolder
 * on the target system.

```

```

*
/*-----*/

function OnFirstUIAfter()
    STRING  szShortcutFolder, szTitle, szMsg;
begin

    // Set up parameters for call to CreateShortcutFolder.
    szShortcutFolder = "ExampleFolder";
    szTitle          = "CreateShortcutFolder";
    szMsg            = "%s created successfully.";

    // Create the program folder.
    if (CreateShortcutFolder (szShortcutFolder) < 0) then
        MessageBox ("Unable to create program folder", SEVERE);
    else
        sprintfBox (INFORMATION, szTitle, szMsg, szShortcutFolder);
    endif;

end;

```

## CtrlClear

The **CtrlClear** function clears the contents of various controls; it deletes the contents of a single- or multi-line edit field, static text field, single- or multi-selection list box, or the edit field of a combo box in a custom dialog.

### Syntax

```
CtrlClear ( szDialogName, nControlID );
```



## Parameters

**Table 92 • CtrlClear Parameters**

Parameter	Description
<b>szDialogName</b>	Specifies the name of the dialog that contains the control to be deleted.
<b>nControlID</b>	Specifies the control ID of the dialog identified by szDialogName.

## Return Values

**Table 93 • CtrlClear Return Values**

Return Value	Description
<b>0</b>	CtrlClear successfully deleted the contents of the specified control.
<b>&lt; 0</b>	CtrlClear was unable to delete the contents of the dialog.

## CtrlClear Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the CtrlClear, CtrlGetText, and CtrlSetText
* functions.
*
* This example script displays a custom dialog that contains
* edit boxes for a user name, company name, and serial number.
* It also contains three buttons: Clear All, Done, and Cancel.
* On initialization of the dialog, the script calls
* CtrlSetText to place default text into each edit control.
*
* Button Operations
*
* Clear All      Calls CtrlClear to clear all edit boxes.
*
* Done           Calls CtrlGetText to retrieve edit box values.
*               If all fields contain data, the dialog
*               closes and the values from the edit boxes are
*               displayed in a message box.
*

```

```

* Cancel          Closes the dialog. Edit box values are
*                  not retrieved or displayed.
*
* The "custom" dialog used in this script is actually the
* Sd dialog that is displayed by the built-in function
* SdRegisterUserEx. Because this dialog is stored
* in the file _isres.dll, which is already compressed in
* the installation, it can be used in a script as a custom
* dialog.
*
/*-----*/

// Initial values to display in the edit boxes.
#define USER_NAME    "Your Name"
#define COMPANY_NAME "Your Company"
#define SERIAL_NUM   "123"

// Dialog and control IDs.
#define RES_DIALOG_ID 12002 // ID of the custom dialog
#define RES_EDIT_NAME 301   // ID of the User Name edit box
#define RES_EDIT_COMPANY 302 // ID of the Company Name edit box
#define RES_EDIT_SERIAL 303 // ID of the Serial Number edit box
#define RES_PBUT_DONE 1     // ID of dialog's Next button
#define RES_PBUT_CANCEL 9   // ID of dialog's Cancel button
#define RES_PBUT_CLEAR 12   // ID of dialog's Back button

    STRING szDialogName, svName, svCompany, svSerial;
    NUMBER nResult, nCmdValue;
    BOOL bDone;
    HWND hwndDlg;

#include "ifx.h"

function OnBegin()
begin

    // Specify a name to identify the custom dialog in this installation.
    szDialogName = "CustomDialog";

    // Define the dialog. Pass a null string in the second parameter
    // to get the dialog from _isuser.dll or _isres.dll. Pass a null
    // string in the third parameter because the dialog is identified
    // by its ID in the fourth parameter.
    nResult = EzDefineDialog (szDialogName, "", "", RES_DIALOG_ID);

    if (nResult < 0) then
        // Report an error; then terminate.
        MessageBox ("Error in defining dialog", SEVERE);
        abort;
    endif;

    // Initialize the indicator used to control the loop.
    bDone = FALSE;

    // Loop until done.
    repeat

        // Display the dialog and return the next dialog event.
        nCmdValue = WaitOnDialog (szDialogName);

```

```

// Respond to the event.
switch (nCmdValue)

    case IDCANCEL:
        // The user clicked the window's Close button.
        Do (EXIT);

    case DLG_ERR:
        MessageBox ("Unable to display dialog. Setup canceled.", SEVERE);
        abort;

    case DLG_INIT:
        // Initialize the back, next, and cancel button enable/disable states
        // for this dialog and replace %P, %VS, %VI with
        // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
        // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
        hwndDlg = CmdGetHwndDlg(szDialogName);
        SdGeneralInit(szDialogName, hwndDlg, 0, "");

        // Set the static text of the buttons.
        CtrlSetText (szDialogName, RES_PBUT_CLEAR, "Clear &All");
        CtrlSetText (szDialogName, RES_PBUT_DONE, "&Done");

        // Initialize the edit controls.
        CtrlSetText (szDialogName, RES_EDIT_NAME, USER_NAME);
        CtrlSetText (szDialogName, RES_EDIT_COMPANY, COMPANY_NAME);
        CtrlSetText (szDialogName, RES_EDIT_SERIAL, SERIAL_NUM);

    case RES_PBUT_CLEAR:
        // Clear all edit controls.
        CtrlClear (szDialogName, RES_EDIT_NAME);
        CtrlClear (szDialogName, RES_EDIT_COMPANY);
        CtrlClear (szDialogName, RES_EDIT_SERIAL);

    case RES_PBUT_DONE:
        // Retrieve the text from edit boxes.
        CtrlGetText (szDialogName, RES_EDIT_NAME, svName);
        CtrlGetText (szDialogName, RES_EDIT_COMPANY, svCompany);
        CtrlGetText (szDialogName, RES_EDIT_SERIAL, svSerial);

        // Verify that all three boxes have data.
        if (StrLength (svName) = 0) ||
            (StrLength (svCompany) = 0) ||
            (StrLength (svSerial) = 0) then
            MessageBox ("All fields must be completed.", WARNING);
        else
            bDone = TRUE;
        endif;

    case RES_PBUT_CANCEL:
        // The user clicked the Cancel button.
        Do (EXIT);
endswitch;

until bDone;

// Close the dialog.
EndDialog (szDialogName);

```

```

// Free the dialog from memory.
ReleaseDialog (szDialogName);

// If the dialog was closed with the Done button,
// display the text from the edit controls.
if (nCmdValue = RES_PBUT_DONE) then
    sprintfBox (INFORMATION, "User Info",
                "Name: %s\nCompany: %s\nSerial: %s",
                svName, svCompany,svSerial);
endif;

end;

```

## CtrlDir

The **CtrlDir** function fills a list box or a combo box control with a file listing that matches the specified path or file name in szDir. You can include names of files, subdirectories, and disk drives in the listing. The CtrlDir function works only with custom dialogs.

### Syntax

```
CtrlDir ( szDialogName, nControlID, szDir, nItems );
```

## Parameters

**Table 94 • CtrlDir Parameters**

Parameter	Description
<b>szDialogName</b>	Specifies the name of a dialog.
<b>nControlID</b>	Specifies the resource ID of the list box or combo box control.
<b>szDir</b>	Specifies the fully qualified path or file name, which may include wildcard characters.
<b>nItems</b>	<p>Specifies the type of listing to display in the control. Pass one or more of the following predefined constants in this parameter. To include more than one type of element, combined these constants with the bitwise OR operator ( ):</p> <ul style="list-style-type: none"> <li>• <b>DLG_DIR_FILE</b>—Creates a list of files matching the file specification szDir.</li> <li>• <b>DLG_DIR_DIRECTORY</b>—Creates a list of subdirectories that exist in the path specification szDir.</li> <li>• <b>DLG_DIR_DRIVE</b>—Creates a list of drives.</li> </ul>

## Return Values

**Table 95 • CtrlDir Return Values**

Return Value	Description
<b>0</b>	CtrlDir successfully filled the specified control in a dialog.
<b>&lt; 0</b>	CtrlDir was unable to fill the specified control.

## CtrlDir Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
```

```

* Demonstrates the CtrlDir function.
*
* This example script displays a file listing in a custom
* dialog. The directory listing is created by a call to
* CtrlDir, which builds the list and places it into the
* custom dialog's list box control. The user can then select
* an item from the list with the keyboard or mouse. If the
* dialog is closed with the Next button, the selected item
* is displayed in a message box.
*
* The "custom" dialog used in this script is actually the
* InstallShield Sd dialog that is displayed by the built-in
* function SdSetupTypeEx. Because this dialog is stored
* in the file _isres.dll, which is already compressed in
* the installation, it can be used in a script as a custom
* dialog.
*
\*-----*/

// Dialog and control IDs.
#define RES_DIALOG_ID      12033 // ID of the custom dialog
#define RES_PBUT_NEXT      1    // ID of Next button
#define RES_PBUT_CANCEL    9    // ID of Cancel button
#define RES_PBUT_BACK      12   // ID of Back button
#define RES_DIALOG_LISTBOX 401   // ID of list box
#define RES_STA_MSG_ABOVE  710   // ID of static message above list
#define RES_STA_MSG_BELOW  711   // ID of static message below list

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_CtrlDir(HWND);

function ExFn_CtrlDir(hMSI)
    STRING  szDialogName, svSelection;
    NUMBER  nResult, nCmdValue;
    BOOL     bDone;
    HWND     hwndDlg;
begin

    // Specify a name to identify the custom dialog in this installation.
    szDialogName = "CustomDialog";

    // Define the dialog. Pass a null string in the second parameter
    // to get the dialog from _isuser.dll or _isres.dll. Pass a null
    // string in the third parameter because the dialog is identified
    // by its ID in the fourth parameter.
    nResult = EzDefineDialog (szDialogName, "", "", RES_DIALOG_ID);

    // Report an error; then terminate.
    if (nResult < 0) then
        MessageBox ("Error in defining dialog", SEVERE);
        abort;
    endif;

    // Initialize the indicator used to control the loop.
    bDone = FALSE;

    // Loop until done.

```

```

repeat

    // Display the dialog and return the next dialog event.
    nCmdValue = WaitOnDialog (szDialogName);

    // Respond to the event.
    switch (nCmdValue)
    case DLG_CLOSE:
        // The user clicked the window's Close button.
        Do (EXIT);
    case DLG_ERR:
        MessageBox ("Unable to display dialog. Setup canceled.", SEVERE);
        abort;
    case DLG_INIT:
        // Initialize the back, next, and cancel button enable/disable states
        // for this dialog and replace %P, %VS, %VI with
        // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
        // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
        hwndDlg = CmdGetHwndDlg(szDialogName);
        SdGeneralInit(szDialogName, hwndDlg, 0, "");

        // Set the messages that appear above and below the list box.
        CtrlSetText (szDialogName, RES_STA_MSG_ABOVE,
            "Files found in the root directory of the Windows drive:");
        CtrlSetText (szDialogName, RES_STA_MSG_BELOW,
            "Select a file; then click the Next button");

        // Create a list of all files in the root directory of the
        // drive on which Windows resides and put it into the dialog's
        // list box control.
        if (CtrlDir (szDialogName, RES_DIALOG_LISTBOX, WINDISK^"*.\"",
            DLG_DIR_FILE) < 0) then
            MessageBox ("CtrlDir failed.", SEVERE);
            bDone = TRUE;
        endif;
    case RES_PBUT_CANCEL:
        // The user clicked the Cancel button.
        Do (EXIT);
    case RES_PBUT_NEXT:
        // Get the current selection so it can be displayed.
        CtrlGetCurSel (szDialogName, RES_DIALOG_LISTBOX, svSelection);
        bDone = TRUE;
    case RES_PBUT_BACK:
        bDone = TRUE;
    endswitch;

until bDone;

// Close the dialog.
EndDialog (szDialogName);

// Free the dialog from memory;
ReleaseDialog (szDialogName);

// If the Next button was used to close the dialog,
// display the item that was selected from the list box.
if (nCmdValue = RES_PBUT_NEXT) then
    MessageBox ( "You selected " + svSelection, INFORMATION);
endif;

```

```
end;
```

## CtrlGetCurSel

The **CtrlGetCurSel** function retrieves the currently selected item from a single-selection list box or combo box control in a custom dialog. Call [CtrlGetMultCurSel](#) to retrieve items from multi-selection list boxes.

### Syntax

```
CtrlGetCurSel ( szDialogName, nControlID, svText );
```

### Parameters

**Table 96** • CtrlGetCurSel Parameters

Parameter	Description
<b>szDialogName</b>	Specifies the name of a custom dialog that contains the item to be retrieved.
<b>nControlID</b>	Specifies the resource ID of the single-selection list box or combo box control.
<b>svText</b>	Returns the item currently selected with the control specified by nControlID.

### Return Values

**Table 97** • CtrlGetCurSel Return Values I

Return Value	Description
<b>0</b>	CtrlGetCurSel successfully retrieved the currently selected item from the dialog.
<b>&lt; 0</b>	CtrlGetCurSel was unable to retrieve the selected item.

## CtrlGetCurSel Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the CtrlGetCurSel and CtrlSetCurSel functions.
```



```

*
* This example script displays a custom dialog that has an
* edit box and a list box. After the dialog is initialized,
* the script places the names of folders that reside in the
* root of the Windows disk into the dialog's list box. It
* then calls CtrlSetCurSel to make "Windows" the selected folder.
*
* Each time the user selects a folder name from the list box,
* the script calls CtrlGetCurSel to get the selected item so
* that it can be placed into the edit box. When the dialog
* is closed with the Done button, the currently selected item
* is displayed in a message box.
*
* The "custom" dialog used in this script is actually the
* InstallShield Sd dialog that is displayed by the built-in
* function SdSelectFolder. Because this dialog is stored
* in the file _isres.dll, which is already compressed in
* the installation, it can be used in a script as a custom
* dialog.
*
\*-----*/

// The folder that will be preselected in the list box.
#define PRESELECTED_FOLDER "windows"

// Dialog and control IDs.
#define RES_DIALOG_ID      12008 // ID of the custom dialog
#define RES_PBUT_NEXT      1 // ID of Next button
#define RES_PBUT_CANCEL    9 // ID of Cancel button
#define RES_PBUT_BACK      12 // ID of Back button
#define RES_DIALOG_EDITBOX 301 // ID of the edit box
#define RES_DIALOG_LISTBOX 401 // ID of the list box
#define RES_STA_DESC       710 // ID of the text at top of dialog

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_CtrlGetCurSel(HWND);

function ExFn_CtrlGetCurSel(hMSI)
    STRING  szDialogName, svSelection, szDesc;
    NUMBER  nResult, nCmdValue;
    BOOL    bDone;
    HWND    hwndDlg;
begin

    // Specify a name to identify the custom dialog in this installation.
    szDialogName = "CustomDialog";

    // Define the dialog. Pass a null string in the second parameter
    // to get the dialog from _isuser.dll or _isres.dll. Pass a null
    // string in the third parameter because the dialog is identified
    // by its ID in the fourth parameter.
    nResult = EzDefineDialog (szDialogName, "", "", RES_DIALOG_ID);

    if (nResult < 0) then
        // Report an error; then terminate.
        MessageBox ("Error in defining dialog", SEVERE);
        abort;

```

```

endif;

// Loop until done.
repeat

    // Display the dialog and return the next dialog event.
    nCmdValue = WaitOnDialog (szDialogName);

    // Respond to the event.
    switch (nCmdValue)
    case DLG_CLOSE:
        // The user clicked the window's Close button.
        Do (EXIT);
    case DLG_ERR:
        MessageBox ("Unable to display dialog. Setup canceled.", SEVERE);
        abort;
    case DLG_INIT:
        // Initialize the back, next, and cancel button enable/disable states
        // for this dialog and replace %P, %VS, %VI with
        // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
        // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
        hwndDlg = CmdGetHwndDlg(szDialogName);
        SdGeneralInit(szDialogName, hwndDlg, 0, "");

        // Set the window title.
        SetWindowText (hwndDlg, "Select Folder");

        // Set the message that appears at the top of the dialog.
        szDesc = "Specify an existing folder from the root of drive "
            + WINSYSDISK + "\nThen press Next to continue.";
        CtrlSetText (szDialogName, RES_STA_DESC, szDesc);

        // Fill the dialog's list box with the names of all folders
        // that reside in the root of the Windows drive.
        CtrlDir (szDialogName, RES_DIALOG_LISTBOX,
            WINSYSDISK + "\\*.*", DLG_DIR_DIRECTORY);

        // Select the preselected folder.
        CtrlSetCurSel (szDialogName, RES_DIALOG_LISTBOX,
            PRESELECTED_FOLDER);

        // Put the name of the preselected folder into the edit box.
        CtrlSetText (szDialogName, RES_DIALOG_EDITBOX, PRESELECTED_FOLDER);
    case RES_DIALOG_LISTBOX:
        // Get the current listbox selection.
        CtrlGetCurSel (szDialogName, RES_DIALOG_LISTBOX, svSelection);

        // Strip off the brackets.
        StrSub (svSelection, svSelection, 1, StrLength(svSelection) - 2);

        // Put the current selection in the edit box.
        CtrlSetText (szDialogName, RES_DIALOG_EDITBOX, svSelection);
    case RES_PBTN_BACK:
        bDone = TRUE;
    case RES_PBTN_NEXT:
        // Get the selection from the edit box.
        CtrlGetText (szDialogName, RES_DIALOG_EDITBOX, svSelection);

        // Verify that the edit box contains the name of a

```

```

        // folder that exists in the root of the Windows disk.
        if Is (PATH_EXISTS, WINSYSDISK + "\\\" + svSelection) then
            bDone = TRUE;
        else
            MessageBox ("Folder does not exist.", WARNING);
        endif ;
    case RES_PBUT_CANCEL:
        // The user clicked the Cancel button.
        Do (EXIT);
    endswitch;

until bDone;

// Close the custom dialog.
EndDialog (szDialogName);

// Remove the custom dialog from memory.
ReleaseDialog (szDialogName);

// If the edit box was closed with the Done button,
// display the selected item.
if (nCmdValue = RES_PBUT_NEXT) then
    MessageBox ("You selected " + svSelection + ".", INFORMATION);
endif;

end;

```

## CtrlGetDlgItem

The **CtrlGetDlgItem** function retrieves the window handle of a control in a custom dialog. **CtrlGetDlgItem** is similar to the Windows API **GetDlgItem**, except that with **CtrlGetDlgItem**, you can specify the InstallScript dialog name instead of the dialog's window handle.

### Syntax

```
CtrlGetDlgItem (byval string szDialogName, byval HWND hDialog, byval number nCtrlId);
```

## Parameters

**Table 98 • CtrlGetDlgItem Parameters**

Parameter	Description
<b>szDialogName</b>	Specify the name of the dialog that contains the control whose handle is being retrieved. The dialog must already exist when the function is called.  If hDialog is not a null string (""), you can specify a null string for szDialogName.
<b>hDialog</b>	Specify the window handle of the dialog that contains the control whose handle is being retrieved. The dialog must already exist when the function is called.  If you specify a string (""), <b>CtrlGetDlgItem</b> determines the dialog's handle by calling <b>CmdGetHwndDlg</b> using szDialogName.
<b>nCtrlId</b>	Specify the ID of the control whose window handle you want to retrieve.

## Return Values

**CtrlGetDlgItem** returns the window handle of the control or NULL if the control does not exist or an error occurred. **GetExtendedErrInfo** may return additional error information.

## Additional Information

If you are making multiple calls to **CtrlGetDlgItem** to look up multiple controls, it is recommended that you call **CmdGetHwndDlg** to get the handle of the dialog and pass it to each function call. If you are looking up the window handle of a single control, it is recommended that you specify the dialog name and specify hDialog as a null string ("").

# CtrlGetMLEText

The **CtrlGetMLEText** function retrieves the contents of a multi-line edit field control in a custom dialog. InstallShield places each line of the multi-line edit field into a string list identified by listID. Call [CtrlGetText](#) to retrieve the contents of a single-line edit field control.

## Syntax

```
CtrlGetMLEText ( szDialogName, nControlID, listID );
```

## Parameters

**Table 99 • CtrlGetMLEText Parameters**

Parameter	Description
<b>szDialogName</b>	Specifies the name of a custom dialog that contains the multi-line edit control whose contents are to be retrieved.
<b>nControlID</b>	Specifies the resource ID of the multi-line edit control.
<b>listID</b>	Returns a string list of the lines in the edit field identified by nControlID. The string list identified by listID must already have been initialized by a call to <a href="#">ListCreate</a> .

## Return Values

**Table 100 • CtrlGetMLEText Return Values**

Return Value	Description
<b>0</b>	CtrlGetMLEText successfully retrieved the contents of a multi-line edit field.
<b>&lt; 0</b>	CtrlGetMLEText was unable to retrieve the contents of the control.

## CtrlGetMLEText Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the CtrlSetMLEText and CtrlGetMLEText functions.
*
* This example script displays a custom dialog that contains
* a multi-line edit box. The script creates a list of all
* program folders on the target system and then calls
* CtrlSetMLEText to place that list into the dialog's multi-
* line edit box. The dialog also contains a Save button that
* enables the end user to save the folder names to a text file.
* When that option is selected, the script calls CtrlGetMLEText
* to get the folder names from the multi-line edit box.
*
* The "custom" dialog used in this script is actually the
* InstallShield Sd dialog that is displayed by the built-in
* function SdShowInfoList. Because this dialog is stored in
* the file _isres.dll, which is already compressed in
* the installation, it can be used in a script as a custom
* dialog.
```

```

*
* Notes: The multi-line edit box is defined as read-only
*       in the resource; its contents cannot be edited.
*
*       The script changes the static text of the dialog
*       box's Next button and disables the Back button in order
*       to make the dialog fit the needs of the example.
*
*       The function GetGroupNameList may return an error
*       if the target system is running under a shell other
*       than the Explorer shell.
*
\*-----*/

// Dialog and control IDs.
#define RES_DIALOG_ID      12007 // ID of the custom dialog
#define RES_PBUT_BACK      12    // ID of Next button
#define RES_PBUT_DONE      9     // ID of Cancel button
#define RES_PBUT_SAVE      1     // ID of Back button
#define RES_DIALOG_EDITBOX 301   // ID of edit box
#define RES_TEXT           711   // ID of text above edit box

// Description to display above the multi-line edit box.
#define DESC_TEXT "Click Save to store the list of program folder names in a disk file."

// The program names will be saved in the root of the current
// drive if the end user clicks the Save button.
#define FOLDER_LIST_FILE "\\ISExempl.txt"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_CtrlGetMLEText(HWND);

function ExFn_CtrlGetMLEText(hMSI)
    STRING szDialogName;
    NUMBER nCmdValue, nResult;
    BOOL    bSave, bDone;
    LIST    listFolders;
    HWND    hwndDlg;
begin

    // Specify a name to identify the custom dialog in this installation.
    szDialogName = "CustomDialog";

    // Define the dialog. Pass a null string in the second parameter
    // to get the dialog from _isuser.dll or _isres.dll. Pass a null
    // string in the third parameter because the dialog is identified
    // by its ID in the fourth parameter.
    nResult = EzDefineDialog (szDialogName, "", "", RES_DIALOG_ID);

    if (nResult < 0) then
        // Report an error; then terminate.
        MessageBox ("Error in defining dialog", SEVERE);
        abort;
    endif;

    // Initialize indicator used to control the while loop.
    bDone = FALSE;

```

```

// Loop until done.
repeat

    // Display the dialog and return the next dialog event.
    nCmdValue = WaitOnDialog (szDialogName);

    // Respond to the event.
    switch (nCmdValue)
    case DLG_CLOSE:
        // The user clicked the window's Close button.
        bDone = TRUE;
    case DLG_ERR:
        MessageBox ("Dialog failed", SEVERE);
        bDone = TRUE;
    case DLG_INIT:
        // Initialize the back, next, and cancel button enable/disable states
        // for this dialog and replace %P, %VS, %VI with
        // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
        // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
        hwndDlg = CmdGetHwndDlg(szDialogName);
        SdGeneralInit(szDialogName, hwndDlg, 0, "");

        // Set the window title.
        SetWindowText (hwndDlg, "View Program Folders");

        // Disable the Back button using a call from WinSub.
        _WinSubEnableControl (hwndDlg, RES_PBUT_BACK, 0);

        //Set the dialog's static text.
        CtrlSetText (szDialogName, RES_TEXT, DESC_TEXT);
        CtrlSetText (szDialogName, RES_PBUT_SAVE, "&Save");
        CtrlSetText (szDialogName, RES_PBUT_DONE, "&Done");

        // Create a string list to store the program folder names.
        listFolders = ListCreate (STRINGLIST);

        if (listFolders = LIST_NULL) then
            MessageBox ("Unable to create list.", SEVERE);
            bDone = TRUE;
        else
            // Get the folder names into a list.
            nResult = GetGroupNameList (listFolders);

            if (nResult = 0) then
                // Put the folder names into the
                // dialog's multi-line edit box.
                nResult = CtrlSetMLEText (szDialogName, RES_DIALOG_EDITBOX,
                                         listFolders);
            elseif (nResult != 0) then
                // Handle error from GetGroupNameList or CtrlSetMLEText.
                MessageBox ("Unable to create folder name list.", SEVERE);
                bDone = TRUE;
            endif;

            // Destroy the listID string list.
            ListDestroy (listFolders);
        endif;
    case RES_PBUT_SAVE :

```

```

// Initialize indicator to save program file names.
bSave = FALSE;

if (AskYesNo("Save list as " + FOLDER_LIST_FILE + "?", YES)) then
    // Check for existing file.
    if (Is (FILE_EXISTS, FOLDER_LIST_FILE) = 1) then
        // Query end user to overwrite existing file.
        if (AskYesNo ("Overwrite existing " + FOLDER_LIST_FILE +
            "?", YES)) then
            bSave = TRUE;
        endif;
    else
        bSave = TRUE;
    endif;
endif;

if bSave = TRUE then
    // Create a string list to store list from dialog.
    listFolders = ListCreate (STRINGLIST);

    if (listFolders = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
    else
        // Get the folder names from the
        // dialog's multi-line edit box.
        nResult = CtrlGetMLEText (szDialogName, RES_DIALOG_EDITBOX,
            listFolders);

        // Save the list to a text file.
        ListWriteToFile (listFolders, FOLDER_LIST_FILE);

        // Destroy the listID string list.
        ListDestroy (listFolders);
    endif;
endif;
case RES_PBUT_DONE:
    bDone = TRUE;
endswitch;

until bDone;

// Close the custom dialog.
EndDialog (szDialogName);

// Remove the custom dialog from memory.
ReleaseDialog (szDialogName);

end;

```

## CtrlGetMultCurSel

The **CtrlGetMultCurSel** function retrieves the currently selected lines from a multi-selection list box control, that is, a list box control with the LBS\_MULTIPLESEL style. (This function does not support extended selection list box controls, that is, list box controls with the LBS\_EXTENDEDSEL style.) Each selected line of the multi-selection list box is placed into a string list identified by listID. To retrieve selected text from a single-selection list box control, call [CtrlGetCurSel](#). CtrlGetMultCurSel is for use only with custom dialogs.



## Syntax

```
CtrlGetMultCurSel ( szDialogName, nControlID, listID );
```

## Parameters

**Table 101** • CtrlGetMultCurSel Parameters

Parameter	Description
<b>szDialogName</b>	Specifies the name of a custom dialog that contains the list box control whose contents are to be retrieved.
<b>nControlID</b>	Specifies the resource ID of the multi-line edit control.
<b>listID</b>	Returns the lines of the list box identified by nControlID. The string list identified by listID must already have been initialized by a call to <a href="#">ListCreate</a> .

## Return Values

**Table 102** • CtrlGetMultCurSel Return Values

Return Value	Description
<b>0</b>	CtrlGetMultCurSel successfully retrieved the currently selected items.
<b>&lt; 0</b>	CtrlGetMultCurSel was unable to retrieve the items.

## CtrlGetMultCurSel Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the CtrlSetMultCurSel and CtrlGetMultCurSel
* functions.
*
* This script retrieves the names of all program folders
* on the target system and places them in a list. When the
* dialog is initialized, the CtrlSetList function sets this
* list to be displayed in the list box. The CtrlSetMultCurSel
```

```

* function is then called to highlight the user-selected
* folder.
*
* This list is then destroyed. A new list is created when the
* Next button is clicked. CtrlGetMultCurSel then retrieves the
* elements in the list box and assigns them to this new string
* list. This list is then displayed in an Sd dialog.
*
* Note: In order for this script to run properly, you must set
*       the RES_DIALOG_ID and RES_DIALOG_LISTBOX constants to a
*       dialog and list box created in _isuser.dll.
*
*       The GetGroupNameList function used in this example may
*       return an error if the target system is running under a
*       shell other than Explorer.
*
\*-----*/

// Dialog control IDs.
#define RES_DIALOG_ID           // ID of dialog itself
#define RES_PBUT_NEXT          1 // ID of Next button
#define RES_PBUT_CANCEL        9 // ID of Cancel button
#define RES_PBUT_BACK         12 // ID of Back button
#define RES_DIALOG_LISTBOX     // ID of list box

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_CtrlGetMultCurSel(HWND);

function ExFn_CtrlGetMultCurSel(hMSI)
    STRING szDialogName, szDLL, szTitle, szMsg;
    STRING szText, szDefFolder, svResultFolder;
    NUMBER nCmdValue, nResult, nControlID, nSelectFlag;
    BOOL bDone;
    LIST listID, listFolders;
    HWND hwndDlg;
begin

    Disable(BACKBUTTON);

    szDialogName = "CtrlSetMultCurSel";
    szDLL        = "";

    // Define the dialog. Pass a null string in the second parameter
    // to get the dialog from _isuser.dll or _isres.dll. Pass a null
    // string in the third parameter because the dialog is identified
    // by its ID in the fourth parameter.
    nResult = EzDefineDialog (szDialogName, szDLL, "", RES_DIALOG_ID);

    if (nResult < 0) then
        MessageBox ("Error in defining dialog", SEVERE);
        bDone = TRUE;
    else
        bDone = FALSE;
    endif;

    // Create the listID string list.
    listID = ListCreate (STRINGLIST);

```

```

if (listID = LIST_NULL) then
    MessageBox ("Unable to create list.", SEVERE);
else
    MessageBox ("listID created.", INFORMATION);
endif;

// Retrieve the program folder names into a list.
GetGroupNameList (listID);

// Retrieve a folder name from the user.
szTitle = "CtrlGetMultCurSel & CtrlSetMultCurSel";
SelectFolder (szTitle, szDefFolder, svResultFolder);

// Loop until done.
while (bDone = FALSE)

    // Display the dialog and return the next dialog event.
    nCmdValue = WaitOnDialog (szDialogName);

    // Respond to the event.
    switch (nCmdValue)
        case DLG_ERR:
            MessageBox ("Unable to display dialog. Setup canceled.",SEVERE);
            abort;
        case DLG_INIT:
            // Initialize the back, next, and cancel button enable/disable states
            // for this dialog and replace %P, %VS, %VI with
            // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
            // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
            hwndDlg = CmdGetHwndDlg(szDialogName);
            SdGeneralInit(szDialogName, hwndDlg, 0, "");

            // The following sets the list box to the list of program folders.
            nControlID = RES_DIALOG_LISTBOX;
            CtrlSetList (szDialogName, nControlID, listID);

            szText = svResultFolder;
            nSelectFlag = TRUE;

            //Set the user-selected folder to be highlighted.
            if (CtrlSetMultCurSel (szDialogName, nControlID, szText,
                                   nSelectFlag) < 0) then
                MessageBox ("CtrlSetMultCurSel failed.", SEVERE);
            endif;

            // Destroy the listID string list.
            ListDestroy (listID);
            MessageBox ("listID destroyed.", INFORMATION);
        case DLG_CLOSE:
            // The user clicked the window's Close button.
            Do (EXIT);
        case RES_PBUT_NEXT:
            // Create the listFolders string list.
            listFolders = ListCreate (STRINGLIST);

            if (listFolders = LIST_NULL) then
                MessageBox ("Unable to create list.", SEVERE);
            else

```

```

        MessageBox ("listFolders created.", INFORMATION);
    endif;

    // Retrieve the highlighted elements in the list box, and
    // put them into the listFolders string list.
    if (CtrlGetMultCurSel (szDialogName, nControlID,
        listFolders) < 0) then
        MessageBox ("CtrlGetMultCurSel failed.", SEVERE);
    else
        MessageBox ("CtrlGetMultCurSel successful.", INFORMATION);
    endif;

    bDone = TRUE;
    case RES_PBUT_BACK:
        bDone = TRUE;
    case RES_PBUT_CANCEL:
        // The user clicked the window's Cancel button.
        Do (EXIT);
    endswitch;

endwhile;

szMsg  = "The following are the elements highlighted in the list box:";

// Display the list of elements highlighted.
SdShowInfoList (szTitle, szMsg, listFolders);

// Remove listFolders string list from memory.
ListDestroy (listFolders);
MessageBox ("listFolders destroyed.", INFORMATION);

// Close the dialog.
EndDialog (szDialogName);

// Remove the dialog from memory.
ReleaseDialog (szDialogName);

end;

```

## CtrlGetState

The **CtrlGetState** function gets the current state of a check box or option button control from a custom dialog.

### Syntax

```
CtrlGetState ( szDialogName, nControlID );
```

## Parameters

**Table 103** • CtrlGetState Parameters

Parameter	Description
<b>szDialogName</b>	Specifies the name of the dialog that contains the control.
<b>nControlID</b>	Specifies the resource ID of the check box or option button control whose state is to be retrieved.

## Return Values

**Table 104** • CtrlGetState Return Values

Return Value	Description
<b>BUTTON_CHECKED (-1001)</b>	The check box or option button is selected.
<b>BUTTON_UNCHECKED (-1002)</b>	The check box or option button is not selected.
<b>DLG_ERR (-1)</b>	CtrlGetState was unable to determine the state of the control.

## CtrlGetState Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the CtrlGetState and CtrlSetState functions.
*
* This example script displays a custom dialog that contains
* four check boxes. The script calls CtrlSetState to set the
* first two check boxes to checked. The last two are unchecked
* by default. When the end user clicks the Next button, the
* script calls CtrlGetState to retrieve the state of each
* each check box. The script then displays a message box that
* reports which check boxes were checked.
*
* The "custom" dialog used in this script is actually the
* InstallShield Sd dialog that is displayed by the built-in
* function SdAskOptions. Because this dialog is stored in
* the file _isres.dll, which is already compressed in
* the installation, it can be used in a script as a custom
* dialog.
```

```

*
\*-----*/

// Dialog and control IDs.
#define RES_DIALOG_ID    12020 // ID of the custom dialog
#define RES_PBUT_NEXT    1 // ID of Next button
#define RES_PBUT_CANCEL  9 // ID of Cancel button
#define RES_PBUT_BACK    12 // ID of Back button
#define ID_OP1_CHECK     501 // ID of Option 1 check box
#define ID_OP2_CHECK     502 // ID of Option 2 check box
#define ID_OP3_CHECK     503 // ID of Option 3 check box
#define ID_OP4_CHECK     504 // ID of Option 4 check box
#define ID_STA_DESC      711 // ID of static text description

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_CtrlGetState(HWND);

function ExFn_CtrlGetState(hMSI)
    STRING  szDialogName, szMsg;
    NUMBER  nResult, nCmdValue, hwndDlg;
    BOOL    bDone;
begin
    // Specify a name to identify the custom dialog in this installation.
    szDialogName = "ExDialog";

    // Define the dialog. Pass a null string in the second parameter
    // to get the dialog from _isuser.dll or _isres.dll. Pass a null
    // string in the third parameter because the dialog is identified
    // by its ID in the fourth parameter.
    nResult = EzDefineDialog (szDialogName, "", "", RES_DIALOG_ID);

    if (nResult < 0) then
        // Report an error; then terminate.
        MessageBox ("Error in defining dialog", SEVERE);
        abort;
    endif;

    // Initialize the indicator used to control the loop.
    bDone = FALSE;

    repeat

        // Display the dialog and return the next dialog event.
        nCmdValue = WaitOnDialog (szDialogName);

        // Respond to the event.
        switch (nCmdValue)
            case DLG_CLOSE:
                // The user clicked the window's Close button.
                Do (EXIT);
            case DLG_ERR:
                MessageBox ("Unable to display dialog. Setup canceled.", SEVERE);
                abort;
            case DLG_INIT:
                // Initialize the back, next, and cancel button enable/disable states

```

```

    // for this dialog and replace %P, %VS, %VI with
    // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
    // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
    hwndDlg = CmdGetHwndDlg(szDialogName);
    SdGeneralInit(szDialogName, hwndDlg, 0, "");

    // Set the window title.
    SetWindowText (hwndDlg, "Select Options");

    // Set static text description displayed above check boxes.
    CtrlSetText (szDialogName, ID_STA_DESC,
        "Select and/or clear options. Then click Next.");

    // Options are cleared by default, so select Options 1 and 2.
    if (CtrlSetState (szDialogName, ID_OP1_CHECK, BUTTON_CHECKED) < 0) then
        MessageBox ("First call to CtrlSetState failed.", SEVERE);
        bDone = TRUE;
    elseif (CtrlSetState (szDialogName, ID_OP2_CHECK, BUTTON_CHECKED) < 0) then
        MessageBox ("Second call to CtrlSetState failed.", SEVERE);
        bDone = TRUE;
    endif;

    case RES_PBUT_NEXT:
        bDone = TRUE;
    case RES_PBUT_CANCEL:
        // The user clicked the Cancel button.
        Do (EXIT);
    case RES_PBUT_BACK:
        bDone = TRUE;
    endswitch;

until bDone;

// Build message if end user clicked the Next button.
if (nCmdValue = RES_PBUT_NEXT) then
    // Start building the message to display to the end user.
    szMsg = "You selected the following items:\n\n";

    // If first option is selected, add line to message.
    if (CtrlGetState (szDialogName, ID_OP1_CHECK) = BUTTON_CHECKED) then
        szMsg = szMsg + "Option 1\n";
    endif;

    // If second option is selected, add line to message.
    if (CtrlGetState (szDialogName, ID_OP2_CHECK) = BUTTON_CHECKED) then
        szMsg = szMsg + "Option 2\n";
    endif;

    // If third option is selected, add line to message.
    if (CtrlGetState (szDialogName, ID_OP3_CHECK) = BUTTON_CHECKED) then
        szMsg = szMsg + "Option 3\n";
    endif;

    // If fourth option is selected, add line to message.
    if (CtrlGetState (szDialogName, ID_OP4_CHECK) = BUTTON_CHECKED) then
        szMsg = szMsg + "Option 4\n";
    endif;
endif;

```

```

// Close the custom dialog.
EndDialog (szDialogName);

// Remove the custom dialog from memory.
ReleaseDialog (szDialogName);

// Display message if dialog was closed with Next button.
if (nCmdValue = RES_PBUT_NEXT) then
    MessageBox (szMsg, INFORMATION);
endif;

end;

```

## CtrlGetSubCommand

The **CtrlGetSubCommand** function retrieves the action performed on a control in a custom dialog. For example, CtrlGetSubCommand can tell you if the user single-clicked or double-clicked a list box or combo box control. It can also tell you when the contents of an edit field have changed.

Advanced developers can call [CmdGetHwndDlg](#) to handle additional information.

### Syntax

```
CtrlGetSubCommand (szDialogName);
```

### Parameters

**Table 105** • CtrlGetSubCommand Parameters

Parameter	Description
<b>szDialogName</b>	Specifies the name of a custom dialog.

### Return Values

**Table 106** • CtrlGetSubCommand Return Values

Return Value	Description
<b>EDITBOX_CHANGE (-1007)</b>	The contents of the edit box have changed.
<b>LISTBOX_ENTER (-1008)</b>	The user double-clicked a list box item.
<b>LISTBOX_SELECT (-1009)</b>	The user single-clicked a list box item.

## CtrlGetSubCommand Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.



```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the CtrlGetSubCommand function.
*
* This example script displays a list of program groups in
* the list box of a custom dialog. It then responds to events
* from the edit box and list box as follows:
*
* Single-click in list box: Selected item is put into edit box.
*
* Double-click in list box: Double-clicked item is stored for
*                           later display and dialog is closed.
*
* Change in edit box value: The default system sound is played.
*
* The "custom" dialog used in this script is actually the
* InstallShield Sd dialog that is displayed by the built-in
* function SdSelectFolder. Because this dialog is stored in
* the file _isres.dll, which is already compressed in
* the installation, it can be used in a script as a custom
* dialog.
*
\*-----*/

// Dialog and control IDs.
#define RES_DIALOG_ID      12008    // ID of custom dialog
#define RES_PBUT_NEXT      1        // ID of Next button
#define RES_PBUT_CANCEL    9        // ID of Cancel button
#define RES_PBUT_BACK      12       // ID of Back button
#define RES_DIALOG_EDITBOX 301      // ID of edit box
#define RES_DIALOG_LISTBOX 401      // ID of list box

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_CtrlGetSubCommand(HWND);

function ExFn_CtrlGetSubCommand(hMSI)
    STRING szDialogName, svSelection;
    NUMBER nResult, nCmdValue, nSubCommand;
    BOOL    bDone, bSelected;
    HWND    hwndDlg;
begin

    // Specify a name to identify the custom dialog in this installation.
    szDialogName = "CustomDialog";

    // Define the dialog. Pass a null string in the second parameter
    // to get the dialog from _isuser.dll or _isres.dll. Pass a null
    // string in the third parameter because the dialog is identified
    // by its ID in the fourth parameter.
    nResult = EzDefineDialog (szDialogName, "", "", RES_DIALOG_ID);

    if (nResult < 0) then
        // Report an error; then terminate.
        MessageBox ("Error in defining dialog", SEVERE);
        abort;

```

```

endif;

// Initialize indicators used to control the while loop
// and indicate whether an item was selected.
bDone = FALSE;
bSelected = FALSE;

// Loop until done.
repeat

    // Display the dialog and return the next dialog event.
    nCmdValue = WaitOnDialog (szDialogName);

    // Respond to the event.
    switch (nCmdValue)
    case DLG_CLOSE:
        // The user clicked the window's Close button.
        Do (EXIT);
    case DLG_ERR:
        MessageBox ("Unable to display dialog. Setup canceled.", SEVERE);
        abort;
    case DLG_INIT:
        // Initialize the back, next, and cancel button enable/disable states
        // for this dialog and replace %P, %VS, %VI with
        // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
        // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
        hwndDlg = CmdGetHwndDlg(szDialogName);
        SdGeneralInit(szDialogName, hwndDlg, 0, "");

        // Place a list of folders into the dialog's list box.
        if (CtrlPGroups (szDialogName, RES_DIALOG_LISTBOX) < 0) then
            MessageBox ("CtrlPGroups failed.", SEVERE);
        endif;
    case RES_DIALOG_LISTBOX:
        // Get the event.
        nSubCommand = CtrlGetSubCommand (szDialogName);

        if (nSubCommand = LISTBOX_SELECT) then
            // Single-click: Put the selected item in the edit box.
            CtrlGetCurSel (szDialogName, RES_DIALOG_LISTBOX, svSelection );
            CtrlSetText (szDialogName, RES_DIALOG_EDITBOX, svSelection );
        elseif (nSubCommand = LISTBOX_ENTER) then
            // Double-click: Get the selected item and set
            // indicator to exit.
            CtrlGetCurSel (szDialogName, RES_DIALOG_LISTBOX, svSelection );
            bSelected = TRUE;
            bDone = TRUE;
        endif;
    case RES_DIALOG_EDITBOX:
        // Get the event.
        nSubCommand = CtrlGetSubCommand (szDialogName);

        // Play default system sound if the edit box contents have changed.
        if (nSubCommand = EDITBOX_CHANGE) then
            MessageBeep (0);
        endif;
    case RES_PBUT_CANCEL:
        // The user clicked the Cancel button.
        Do (EXIT);

```

```

        case RES_PBUT_NEXT:
            // Get the current selection from the edit box.
            CtrlGetText (szDialogName, RES_DIALOG_EDITBOX, svSelection );
            bSelected = TRUE;
            bDone = TRUE;
        case RES_PBUT_BACK:
            bDone = TRUE;
        endswitch;

    until bDone;

    // Close the dialog.
    EndDialog (szDialogName);

    // Free the dialog from memory.
    ReleaseDialog (szDialogName);

    if bSelected then
        // Display the name of the selected folder.
        MessageBox ("You selected " + svSelection + ".", INFORMATION);
    endif;

end;

```

## CtrlGetText

The **CtrlGetText** function retrieves the text from an edit field, static text field, or button control of a custom dialog. To retrieve the text from multi-line edit field controls, call [CtrlGetMLEText](#).

### Syntax

```
CtrlGetText ( szDialogName, nControlID, svText );
```

## Parameters

**Table 107 • CtrlGetText Parameters**

Parameter	Description
<b>szDialogName</b>	Specifies the name of a dialog that contains the field or control whose text is to be retrieved.
<b>nControlID</b>	Specifies the resource ID of the edit field, static text field, or button control.
<b>svText</b>	Returns the text from the control or field identified by nControlID.

## Return Values

**Table 108 • CtrlGetText Return Values**

Return Value	Description
<b>0</b>	CtrlGetText successfully retrieved the contents of the control.
<b>&lt; 0</b>	CtrlGetText was unable to retrieve the contents.

## CtrlGetText Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the CtrlSetText, CtrlGetText, and CtrlSelectText
* functions.
*
* This example script displays a custom dialog with two
* edit boxes to obtain a user name and company name. The
* script calls CtrlSetText to place initial values into the
* edit boxes and CtrlSelectText to select the contents of the
* first edit box. When the user clicks the Next button, the
* script calls CtrlGetText to retrieve the contents of the edit
* boxes so that they can be displayed in a message box after
* the custom dialog is closed.
*
* The "custom" dialog used in this script is actually the
* InstallShield Sd dialog that is displayed by the built-in
* function SdRegisterUser. Because this dialog is stored in
```

```

* the file _isres.dll, which is already compressed in
* the installation, it can be used in a script as a custom
* dialog.
*
\*-----*/

// Dialog and control IDs.
#define RES_DIALOG_ID      12001 // ID of custom dialog
#define RES_PBUT_NEXT      1    // ID of Next button
#define RES_PBUT_CANCEL    9    // ID of Cancel button
#define RES_PBUT_BACK      12   // ID of Back button
#define RES_EDITNAME       301   // ID of edit box
#define RES_EDITCOMPANY    302   // ID of edit box

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_CtrlGetText(HWND);

function ExFn_CtrlGetText(hMSI)
    STRING szDialogName, svName, svCompany;
    NUMBER nResult, nCmdValue;
    BOOL    bDone;
    HWND    hwndDlg;
begin

    // Specify a name to identify the custom dialog in this installation.
    szDialogName = "CustomDialog";

    // Define the dialog. Pass a null string in the second parameter
    // to get the dialog from _isuser.dll or _isres.dll. Pass a null
    // string in the third parameter because the dialog is identified
    // by its ID in the fourth parameter.
    nResult = EzDefineDialog (szDialogName, "", "", RES_DIALOG_ID);

    if (nResult < 0) then
        // Report an error; then terminate.
        MessageBox ("Error in defining dialog", SEVERE);
        abort;
    endif;

    // Initialize indicator used to control the loop.
    bDone = FALSE;

    // Loop until done.
    repeat

        // Display the dialog and return the next dialog event.
        nCmdValue = WaitOnDialog (szDialogName);

        // Respond to the event.
        switch (nCmdValue)
            case DLG_CLOSE:
                // The user clicked the window's Close button.
                Do (EXIT);
            case DLG_ERR:
                MessageBox ("Unable to display dialog. Setup canceled.", SEVERE);
                abort;
            case DLG_INIT:

```

```

// Initialize the back, next, and cancel button enable/disable states
// for this dialog and replace %P, %VS, %VI with
// IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
// IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
hwndDlg = CmdGetHwndDlg(szDialogName);
SdGeneralInit(szDialogName, hwndDlg, 0, "");

// Put initial values into the edit boxes.
CtrlSetText (szDialogName, RES_EDITNAME, "Your name");
CtrlSetText (szDialogName, RES_EDITCOMPANY, "Your company");

// Select the Name edit box.
CtrlSelectText (szDialogName, RES_EDITNAME);
case RES_PBUT_NEXT:
    // Get the contents of the edit boxes.
    CtrlGetText (szDialogName, RES_EDITNAME, svName);
    CtrlGetText (szDialogName, RES_EDITCOMPANY, svCompany);

    // Verify that both edit boxes have data.
    if (StrLength(svName) = 0) || (StrLength(svCompany) = 0) then
        MessageBox ("Both fields must be completed.", INFORMATION);
    else
        bDone = TRUE;
    endif;
case RES_PBUT_CANCEL:
    // The user clicked the Cancel button.
    Do (EXIT);
case RES_PBUT_BACK:
    bDone = TRUE;
endswitch;

until bDone;

// Close the dialog.
EndDialog (szDialogName);

// Remove the dialog from memory.
ReleaseDialog (szDialogName);

// If dialog is closed with Next button, display name and company.
if nCmdValue = RES_PBUT_NEXT then
    MessageBox (svName + "\n" + svCompany, INFORMATION);
endif;

end;

```

## CtrlGetUrlForLinkClicked



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **CtrlGetUrlForLinkClicked** function retrieves the URL for a link that an end user clicked.



**Tip** • To learn how to add links to dialogs in InstallScript and InstallScript MSI projects, see *Using an HTML Control on a Dialog*.

## Syntax

```
CtrlGetUrlForLinkClicked (byval string szDialogName, byval number nControlID, byref string svText);
```

## Parameters

**Table 109** • CtrlGetUrlForLinkClicked Parameters

Parameter	Description
<b>szDialogName</b>	Specify the name of the dialog that contains the HTML control.
<b>nControlID</b>	Specify the control ID of the HTML control. This ID is the same as the ID of the static control that was converted into an HTML control.
<b>svText</b>	Specify the string variable to return the link URL text.

## Return Values

**Table 110** • CtrlGetUrlForLinkClicked Return Values

Return Value	Description
<b>0</b>	<b>CtrlGetUrlForLinkClicked</b> was able to determine the link that the end user clicked.
<b>ISERR_GEN_FAILURE</b>	<b>CtrlGetUrlForLinkClicked</b> was unable to determine the link that an end user clicked. This function returns this value if the specified control ID is not for an HTML control.

## CtrlGetUrlForLinkClicked Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
//-----
//
// InstallShield Example Script
//
// Demonstrates how to use an HTML control with the
// CtrlGetUrlForLinkClicked and CtrlSetText functions
//
// To use this sample script:
// 1. Add a custom dialog to your project.
```

```

// 2. Add a static text control to the dialog.
//
//-----

#define MY_HYPERLINK1 1401

function MyCustomDialog(szTitle, szMsg)
    STRING  szDlg, szTemp, szUrl;
    NUMBER  nId, nMessage, nTemp, nSdDialog;
    HWND    hwndDlg;
    BOOL     bDone;

begin

    // Specify a name to identify the custom dialog in this installation.
    szDlg = "CustomName";

    while (!bDone)

        nId = WaitOnDialog( szDlg );

        switch(nId)
        case DLG_INIT:
            // Initialize the back, next, and cancel button enable/disable states
            // for this dialog and replace %P, %VS, %VI with
            // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
            // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
            hwndDlg = CmdGetHwndDlg(szDlg);
            SdGeneralInit(szDlg, hwndDlg, 0, "");

            // Put the corresponding Info in the List Field
            if( szMsg != "" ) then
                SdSetStatic(szDlg, SD_STA_MSG, szMsg);
            endif;

            SdSetDlgTitle(szDlg, hwndDlg, szTitle);

            CtrlSetText(szDlg, MY_HYPERLINK1,
                "[html]<style type=\"text/css\">html,body {padding:0; margin:0;} *
                {font-size: 8pt; font-family: \"MS Sans Serif\";}</style>
                <a href=\"http://www.MyWebSite.com\">
                Visit my Web site</a>");

        case MY_HYPERLINK1:
            CtrlGetUrlForLinkClicked(szDlg, MY_HYPERLINK1, szUrl);
            MessageBox("Hyperlink clicked: " + szUrl, 0);

            // TO DO: Add additional case statements as needed.

        default:
            // check standard handling
            if (SdIsStdButton( nId ) && SdDoStdButton( nId )) then
                bDone = TRUE;
            endif;
        endswitch;

    endwhile;
end;

```



**See Also**

Creating New Custom Dialogs in InstallScript and InstallScript MSI Projects  
Using an HTML Control on a Dialog

## CtrlPGroups

The **CtrlPGroups** function places a list of existing program folders in a list box or combo box control. This function is for use only with custom dialogs.

### Syntax

```
CtrlPGroups ( szDialogName, nControlID );
```

### Parameters

**Table 111** • CtrlPGroups Parameters

Parameter	Description
<b>szDialogName</b>	Specifies the name of a custom dialog that contains the control to use.
<b>nControlID</b>	Specifies the resource ID of a list box or combo box control.

### Return Values

**Table 112** • CtrlPGroups Return Values

Return Value	Description
<b>0</b>	CtrlPGroups successfully placed the specified list of program folders in the control.
<b>&lt; 0</b>	CtrlPGroups was unable to place the specified list of program folders in the control.

## CtrlPGroups Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the CtrlPGroups function.
 *
 * This example script displays a custom dialog that has an
```

```

* edit box and a list box. After the dialog is initialized,
* the script calls CtrlPGroups to create a list of program
* folder names and place that list into the dialog's list box.
*
* The "custom" dialog used in this script is actually the
* InstallShield Sd dialog that is displayed by the built-in
* function SdSelectFolder. Because this dialog is stored in
* the file _isres.dll, which is already compressed in
* the installation, it can be used in a script as a custom
* dialog.
*
\*-----*/

// Dialog and control IDs.
#define RES_DIALOG_ID      12008    // ID of custom dialog
#define RES_PBUT_NEXT      1       // ID of Next button
#define RES_PBUT_CANCEL    9       // ID of Cancel button
#define RES_PBUT_BACK      12      // ID of Back button
#define RES_DIALOG_EDITBOX 301     // ID of edit box
#define RES_DIALOG_LISTBOX 401     // ID of list box

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_CtrlPGroups(HWND);

function ExFn_CtrlPGroups(hMSI)
    STRING szDialogName, svSelection;
    NUMBER nResult, nCmdValue, nControlID;
    BOOL    bDone;
    HWND    hwndDlg;
begin

    // Specify a name to identify the custom dialog in this installation.
    szDialogName = "CustomDialog";

    // Define the dialog. Pass a null string in the second parameter
    // to get the dialog from _isuser.dll or _isres.dll. Pass a null
    // string in the third parameter because the dialog is identified
    // by its ID in the fourth parameter.
    nResult = EzDefineDialog (szDialogName, "", "", RES_DIALOG_ID);

    if (nResult < 0) then
        // Report an error; then terminate.
        MessageBox ("Error in defining dialog", SEVERE);
        abort;
    endif;

    // Initialize indicator used to control the while loop.
    bDone = FALSE;

    // Loop until done.
    repeat

        // Display the dialog and return the next dialog event.
        nCmdValue = WaitOnDialog (szDialogName);

        // Respond to the event.
        switch (nCmdValue)

```

```

case DLG_CLOSE:
    // The user clicked the window's Close button.
    Do (EXIT);
case DLG_ERR:
    MessageBox ("Unable to display dialog. Setup canceled.", SEVERE);
    abort;
case DLG_INIT:
    // Initialize the back, next, and cancel button enable/disable states
    // for this dialog and replace %P, %VS, %VI with
    // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
    // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
    hwndDlg = CmdGetHwndDlg(szDialogName);
    SdGeneralInit(szDialogName, hwndDlg, 0, "");

    // Place a list of folders into the dialog's list box.
    if (CtrlPGroups (szDialogName, RES_DIALOG_LISTBOX) < 0) then
        MessageBox ("CtrlPGroups failed.", SEVERE);
        bDone = TRUE;
    endif;
case RES_DIALOG_LISTBOX:
    // Get the current list box selection.
    CtrlGetCurSel (szDialogName, RES_DIALOG_LISTBOX, svSelection);

    // Put the current selection in the edit box.
    CtrlSetText (szDialogName, RES_DIALOG_EDITBOX, svSelection);
case RES_PBUT_CANCEL:
    // The user clicked the Cancel button.
    Do (EXIT);
case RES_PBUT_NEXT:
    // Get the current value of the edit box.
    CtrlGetText (szDialogName, RES_DIALOG_EDITBOX, svSelection);

    // Verify that the edit box contains the name of an
    // existing program folder.
    if CtrlSetCurSel (szDialogName, RES_DIALOG_LISTBOX, svSelection) =
0 then
        bDone = TRUE;
    else
        MessageBox ("Program folder does not exist.", WARNING);
    endif;
case RES_PBUT_BACK:
    bDone = TRUE;
endswitch;

until bDone;

// Close the dialog.
EndDialog (szDialogName);

// Free the dialog from memory.
ReleaseDialog (szDialogName);

// If the edit box was closed with the Done button,
// display the selected item.
if (nCmdValue = RES_PBUT_NEXT) then
    MessageBox ( "You selected " + svSelection + ".", INFORMATION);
endif;

end;

```

# CtrlSelectText

The **CtrlSelectText** function selects all the text in an edit field or the edit field of a combo box. If the control is a multi-line edit field, this function selects all the text on all lines. This function is for use only with custom dialogs.

## Syntax

```
CtrlSelectText ( szDialogName, nControlID );
```

## Parameters

**Table 113** • CtrlSelectText Parameters

Parameter	Description
<b>szDialogName</b>	Specifies the name of a valid dialog that contains the edit field to be selected.
<b>nControlID</b>	Specifies the resource ID of the edit field or combo box control to be selected.

## Return Values

**Table 114** • CtrlSelectText Return Values

Return Value	Description
<b>0</b>	CtrlSelectText successfully selected all the text in the field.
<b>&lt; 0</b>	CtrlSelectText was unable to select the text.

## CtrlSelectText Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the CtrlSetText, CtrlGetText, and CtrlSelectText
* functions.
*
* This example script displays a custom dialog with two
* edit boxes to obtain a user name and company name. The
* script calls CtrlSetText to place initial values into the
* edit boxes and CtrlSelectText to select the contents of the
* first edit box. When the user clicks the Next button, the
* script calls CtrlGetText to retrieve the contents of the edit
```

```

* boxes so that they can be displayed in a message box after
* the custom dialog is closed.
*
* The "custom" dialog used in this script is actually the
* InstallShield Sd dialog that is displayed by the built-in
* function SdRegisterUser. Because this dialog is stored in
* the file _isres.dll, which is already compressed in
* the installation, it can be used in a script as a custom
* dialog.
*
\*-----*/

// Dialog and control IDs.
#define RES_DIALOG_ID      12001  // ID of custom dialog
#define RES_PBUT_NEXT      1      // ID of Next button
#define RES_PBUT_CANCEL    9      // ID of Cancel button
#define RES_PBUT_BACK      12     // ID of Back button
#define RES_EDITNAME       301     // ID of edit box
#define RES_EDITCOMPANY    302     // ID of edit box

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_CtrlSelectText(HWND);

function ExFn_CtrlSelectText(hMSI)
    STRING szDialogName, svName, svCompany;
    NUMBER nResult, nCmdValue;
    BOOL    bDone;
    HWND    hwndDlg;
begin

    // Specify a name to identify the custom dialog in this installation.
    szDialogName = "CustomDialog";

    // Define the dialog. Pass a null string in the second parameter
    // to get the dialog from _isuser.dll or _isres.dll. Pass a null
    // string in the third parameter because the dialog is identified
    // by its ID in the fourth parameter.
    nResult = EzDefineDialog (szDialogName, "", "", RES_DIALOG_ID);

    if (nResult < 0) then
        // Report an error; then terminate.
        MessageBox ("Error in defining dialog", SEVERE);
        abort;
    endif;

    // Initialize indicator used to control the loop.
    bDone = FALSE;

    // Loop until done.
    repeat

        // Display the dialog and return the next dialog event.
        nCmdValue = WaitOnDialog (szDialogName);

        // Respond to the event.
        switch (nCmdValue)
            case DLG_CLOSE:

```

```

        // The user clicked the window's Close button.
        Do (EXIT);
    case DLG_ERR:
        MessageBox ("Unable to display dialog. Setup canceled.", SEVERE);
        abort;
    case DLG_INIT:
        // Initialize the back, next, and cancel button enable/disable states
        // for this dialog and replace %P, %VS, %VI with
        // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
        // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
        hwndDlg = CmdGetHwndDlg(szDialogName);
        SdGeneralInit(szDialogName, hwndDlg, 0, "");

        // Put initial values into the edit boxes.
        CtrlSetText (szDialogName, RES_EDITNAME, "Your name");
        CtrlSetText (szDialogName, RES_EDITCOMPANY, "Your company");

        // Select the Name edit box.
        CtrlSelectText (szDialogName, RES_EDITNAME);
    case RES_PBUT_NEXT:
        // Get the contents of the edit boxes.
        CtrlGetText (szDialogName, RES_EDITNAME, svName);
        CtrlGetText (szDialogName, RES_EDITCOMPANY, svCompany);

        // Verify that both edit boxes have data.
        if (StrLength(svName) = 0) || (StrLength(svCompany) = 0) then
            MessageBox ("Both fields must be completed.", INFORMATION);
        else
            bDone = TRUE;
        endif;
    case RES_PBUT_CANCEL:
        // The user clicked the Cancel button.
        Do (EXIT);
    case RES_PBUT_BACK:
        bDone = TRUE;
    endswitch;

until bDone;

// Close the dialog.
EndDialog (szDialogName);

// Remove the dialog from memory.
ReleaseDialog (szDialogName);

// If dialog is closed with Next button, display name and company.
if nCmdValue = RES_PBUT_NEXT then
    MessageBox (svName + "\n" + svCompany, INFORMATION);
endif;

end;

```

## CtrlSetCurSel

The **CtrlSetCurSel** function searches the specified list or combo box control for a string. If the string is found, CtrlSetCurSel selects (highlights) the item. Call [CtrlSetMultCurSel](#) for multi-selection list box and combo box controls. The CtrlSetCurSel function is for use only with custom dialogs.

## Syntax

```
CtrlSetCurSel ( szDialogName, nControlID, szText );
```

## Parameters

**Table 115 • CtrlSetCurSel Parameters**

Parameter	Description
<b>szDialogName</b>	Specifies the name of a valid custom dialog that contains the control to be searched.
<b>nControlID</b>	Specifies the resource ID of the control that contains the search string.
<b>szText</b>	Specifies the search string. If the string is found, it is selected (highlighted).

## Return Values

**Table 116 • CtrlSetCurSel Return Values**

Return Value	Description
<b>0</b>	CtrlSetCurSel successfully found and selected the specified string.
<b>&lt; 0</b>	CtrlSetCurSel was unable to find and select the specified string.

## CtrlSetCurSel Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the CtrlGetCurSel and CtrlSetCurSel functions.
*
* This example script displays a custom dialog that has an
* edit box and a list box. After the dialog is initialized,
* the script places the names of folders that reside in the
* root of the Windows disk into the dialog's list box. It
* then calls CtrlSetCurSel to make "Windows" the selected folder.
*
* Each time the user selects a folder name from the list box,
* the script calls CtrlGetCurSel to get the selected item so
* that it can be placed into the edit box. When the dialog
```

```

* is closed with the Done button, the currently selected item
* is displayed in a message box.
*
* The "custom" dialog used in this script is actually the
* InstallShield Sd dialog that is displayed by the built-in
* function SdSelectFolder. Because this dialog is stored
* in the file _isres.dll, which is already compressed in
* the installation, it can be used in a script as a custom
* dialog.
*
\*-----*/

// The folder that will be preselected in the list box.
#define PRESELECTED_FOLDER "windows"

// Dialog and control IDs.
#define RES_DIALOG_ID      12008 // ID of the custom dialog
#define RES_PBUT_NEXT      1 // ID of Next button
#define RES_PBUT_CANCEL    9 // ID of Cancel button
#define RES_PBUT_BACK      12 // ID of Back button
#define RES_DIALOG_EDITBOX 301 // ID of the edit box
#define RES_DIALOG_LISTBOX 401 // ID of the list box
#define RES_STA_DESC        710 // ID of the text at top of dialog

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_CtrlSetCurSel(HWND);

function ExFn_CtrlSetCurSel(hMSI)
    STRING  szDialogName, svSelection, szDesc;
    NUMBER  nResult, nCmdValue;
    BOOL     bDone;
    HWND     hwndDlg;
begin

    // Specify a name to identify the custom dialog in this installation.
    szDialogName = "CustomDialog";

    // Define the dialog. Pass a null string in the second parameter
    // to get the dialog from _isuser.dll or _isres.dll. Pass a null
    // string in the third parameter because the dialog is identified
    // by its ID in the fourth parameter.
    nResult = EzDefineDialog (szDialogName, "", "", RES_DIALOG_ID);

    if (nResult < 0) then
        // Report an error; then terminate.
        MessageBox ("Error in defining dialog", SEVERE);
        abort;
    endif;

    // Loop until done.
    repeat

        // Display the dialog and return the next dialog event.
        nCmdValue = WaitOnDialog (szDialogName);

        // Respond to the event.
        switch (nCmdValue)

```



```

case DLG_CLOSE:
    // The user clicked the window's Close button.
    Do (EXIT);
case DLG_ERR:
    MessageBox ("Unable to display dialog. Setup canceled.", SEVERE);
    abort;
case DLG_INIT:
    // Initialize the back, next, and cancel button enable/disable states
    // for this dialog and replace %P, %VS, %VI with
    // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
    // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
    hwndDlg = CmdGetHwndDlg(szDialogName);
    SdGeneralInit(szDialogName, hwndDlg, 0, "");

    // Set the window title.
    SetWindowText (hwndDlg, "Select Folder");

    // Set the message that appears at the top of the dialog.
    szDesc = "Specify an existing folder from the root of drive "
        + WINSYSDISK + "\nThen press Next to continue.";
    CtrlSetText (szDialogName, RES_STA_DESC, szDesc);

    // Fill the dialog's list box with the names of all folders
    // that reside in the root of the Windows drive.
    CtrlDir (szDialogName, RES_DIALOG_LISTBOX,
        WINSYSDISK + "\\*.*", DLG_DIR_DIRECTORY);

    // Select the preselected folder.
    CtrlSetCurSel (szDialogName, RES_DIALOG_LISTBOX,
        PRESELECTED_FOLDER);

    // Put the name of the preselected folder into the edit box.
    CtrlSetText (szDialogName, RES_DIALOG_EDITBOX, PRESELECTED_FOLDER);
case RES_DIALOG_LISTBOX:
    // Get the current list box selection.
    CtrlGetCurSel (szDialogName, RES_DIALOG_LISTBOX, svSelection);

    // Strip off the brackets.
    StrSub (svSelection, svSelection, 1, StrLength(svSelection) - 2);

    // Put the current selection in the edit box.
    CtrlSetText (szDialogName, RES_DIALOG_EDITBOX, svSelection);
case RES_PBUT_BACK:
    bDone = TRUE;
case RES_PBUT_NEXT:
    // Get the selection from the Edit box.
    CtrlGetText (szDialogName, RES_DIALOG_EDITBOX, svSelection);

    // Verify that the edit box contains the name of a
    // folder that exists in the root of the Windows disk.
    if Is (PATH_EXISTS, WINSYSDISK + "\\\" + svSelection) then
        bDone = TRUE;
    else
        MessageBox ("Folder does not exist.", WARNING);
    endif ;
case RES_PBUT_CANCEL:
    // The user clicked the Cancel button.
    Do (EXIT);
endswitch;

```

```

        until bDone;

        // Close the custom dialog.
        EndDialog (szDialogName);

        // Remove the custom dialog from memory.
        ReleaseDialog (szDialogName);

        // If the edit box was closed with the Done button,
        // display the selected item.
        if (nCmdValue = RES_PBUT_NEXT) then
            MessageBox ("You selected " + svSelection + ".", INFORMATION);
        endif;
    end;

```

## CtrlSetFont

The **CtrlSetFont** function specifies a font for a control in a custom dialog. Call this function from within the DLG\_INIT routine of the dialog message processing loop.

### Syntax

```
CtrlSetFont ( szDialogName, hFont, nControlID );
```

## Parameters

**Table 117** • CtrlSetFont Parameters

Parameter	Description
<b>szDialogName</b>	Specifies the name of a valid dialog.
<b>hFont</b>	Specifies the handle of a font that has been created by a call to GetFont
<b>nControlID</b>	Specifies the resource ID of the control whose font is to be set. To set the font for all the controls in the dialog, pass the predefined constant ALLCONTROLS in this parameter.

## Return Values

**Table 118** • CtrlSetFont Return Values

Return Value	Description
<b>0</b>	CtrlSetFont successfully set the requested font in a dialog.
<b>&lt; 0</b>	CtrlSetFont was unable to set the font in the requested dialog.

## CtrlSetFont Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the GetFont and CtrlSetFont functions.
*
* This example script calls GetFont to retrieve the handles
* of four fonts. These handles are then passed to CtrlSetFont
* to set the font of the static text fields in a custom dialog
* box.
*
* The "custom" dialog used in this script is actually the
* InstallShield dialog that is displayed by the built-in
* function SetupType. Because this dialog is stored in
* the file _isres.dll, which is already compressed in
* the installation, it can be used in a script as a custom
* dialog.
*
```

```

\*-----*/

// Dialog and control IDs.
#define RES_DIALOG_ID    10203 // ID of the custom dialog
#define RES_PBUT_NEXT    1     // ID of Next button
#define RES_PBUT_CANCEL  9     // ID of Cancel button
#define RES_TEXT_1       202   // ID of first static text box

#define RES_TEXT_2       210   // ID of second static text box
#define RES_TEXT_3       220   // ID of third static text box
#define RES_TEXT_4       230   // ID of fourth static text box

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_CtrlSetFont(HWND);

function ExFn_CtrlSetFont(hMSI)
    STRING szDialogName;
    NUMBER nResult, nCmdValue;
    HWND   hFont1, hFont2, hFont3, hFont4, hwndDlg;
    BOOL    bDone;
begin
    // Get the handle of the fonts to use for the static text
    // that is displayed by the custom dialog.
    hFont1 = GetFont("Arial", 14, STYLE_BOLD);
    hFont2 = GetFont("Times New Roman", 11, STYLE_ITALIC);
    hFont3 = GetFont("Arial", 10, STYLE_BOLD);
    hFont4 = GetFont("Courier New", 9, STYLE_NORMAL);

    if (hFont1 = 0 || hFont2 = 0 || hFont3 = 0 || hFont4 = 0) then
        // Report an error; then terminate.
        MessageBox ("Unable to get all fonts. ", SEVERE);
        abort;
    endif;

    // Specify a name to identify the custom dialog in this installation.
    szDialogName = "CustomDialog";

    // Define the dialog. Pass a null string in the second parameter
    // to get the dialog from _isuser.dll or _isres.dll. Pass a null
    // string in the third parameter because the dialog is identified
    // by its ID in the fourth parameter.

    nResult = EzDefineDialog (szDialogName, "", "", RES_DIALOG_ID);

    if (nResult < 0) then
        // Report an error; then terminate.
        MessageBox ("Error in defining dialog.", SEVERE);
        abort;
    endif;

    // Initialize indicator used to control the while loop.
    bDone = FALSE;

    // Loop until done.
    repeat

```

```

// Display the dialog and return the next dialog event.
nCmdValue = WaitOnDialog (szDialogName);

// Respond to the event.
switch (nCmdValue)
case DLG_CLOSE:
    // The user clicked the window's Close button.
    Do (EXIT);
case DLG_ERR:
    MessageBox ("Unable to display dialog. Setup canceled.", SEVERE);
    abort;
case DLG_INIT:
    // Initialize the back, next, and cancel button enable/disable states
    // for this dialog and replace %P, %VS, %VI with
    // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
    // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
    hwndDlg = CmdGetHwndDlg(szDialogName);
    SdGeneralInit(szDialogName, hwndDlg, 0, "");

    // Set the font and text for static text box 1.
    if (CtrlSetFont (szDialogName, hFont1, RES_TEXT_1) = 0) then
        CtrlSetText (szDialogName, RES_TEXT_1,
            "This text is set in 14-point Arial bold.");
    else
        CtrlSetText (szDialogName, RES_TEXT_1,
            "Unable to set font for first static text box.");
    endif;

    // Set font and text for static text box 2.
    if (CtrlSetFont (szDialogName, hFont2, RES_TEXT_2) = 0) then
        CtrlSetText (szDialogName, RES_TEXT_2,
            "This text is set in 11-point Times New Roman italic.");
    else
        CtrlSetText (szDialogName, RES_TEXT_2,
            "Unable to set font for second static text box.");
    endif;

    // Set font and text for static text box 3.
    if (CtrlSetFont (szDialogName, hFont3, RES_TEXT_3) = 0) then
        CtrlSetText (szDialogName, RES_TEXT_3,
            "This text is set in 10-point Arial bold.");
    else
        CtrlSetText (szDialogName, RES_TEXT_3,
            "Unable to set font for third static text box.");
    endif;

    // Set font and text for static text box 4.
    if (CtrlSetFont (szDialogName, hFont4, RES_TEXT_4) = 0) then
        CtrlSetText (szDialogName, RES_TEXT_4,
            "This text is set in 9-point Courier New.");
    else
        CtrlSetText (szDialogName, RES_TEXT_4,
            "Unable to set font for fourth static text box.");
    endif;
case RES_PBUT_NEXT:
    bDone = TRUE;
case RES_PBUT_CANCEL:
    // The user clicked the Cancel button.
    Do (EXIT);

```

```
        endswitch;  
  
    until bDone;  
  
    // Close the dialog  
    EndDialog (szDialogName);  
  
    // Free the dialog from memory  
    ReleaseDialog (szDialogName);  
  
end;
```

## CtrlSetList

The **CtrlSetList** function places the contents of a string list into the specified single- or multi-selection list box or combo box control. Any pre-existing contents are replaced with the items contained in listID. InstallShield places each element of the string list into each element of the list box or combo box control.

### Syntax

```
CtrlSetList (szDialogName, nControlID, listID);
```

## Parameters

**Table 119** • CtrlSetList Parameters

Parameter	Description
<b>szDialogName</b>	Specifies the name of a dialog that contains the list box or combo box.
<b>nControlID</b>	Specifies the resource ID of the list box or combo box.
<b>listID</b>	Specifies the name of a string list that contains the elements to be copied into the list box or combo box control.

## Return Values

**Table 120** • CtrlSetList Return Values

Return Value	Description
<b>0</b>	CtrlSetList successfully placed the contents of the string list into the control.
<b>&lt; 0</b>	CtrlSetList was unable to place the contents of the string list into the control.

## CtrlSetList Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the CtrlSetList function.
*
* This example script displays a custom dialog that
* contains a list box. After the dialog is initialized,
* the script calls CtrlSetList to place a list of
* InstallShield background color constants into the custom dialog's
* list box.
*
* The user can view the background that corresponds to a
* color constant either by double-clicking the constant or by
* selecting it and clicking the Set button.
*
* The "custom" dialog used in this script is actually the
* InstallShield Sd dialog that is displayed by the built-in
```

```

* function SdSetupTypeEx. Because this dialog is stored in
* the file _isres.dll, which is already compressed in
* the installation, it can be used in a script as a custom
* dialog. Note that the script changes the dialog's
* static text and disables the Back button to make the dialog
* meet the requirements of the example.
*
\*-----*/

// Dialog controls.
#define RES_DIALOG_ID      12033 // ID of the custom dialog
#define RES_PBUTTON_SET    1     // ID of Next button
#define RES_PBUTTON_DONE   9     // ID of Cancel button
#define RES_PBUTTON_BACK   12    // ID of Back button
#define RES_DIALOG_LISTBOX 401   // ID of edit box.
#define RES_TEXT_ABOVE     710   // ID of text above edit box
#define RES_TEXT_BELOW     711   // ID of text below edit box

// Description to display above and below the multi-line edit box.
#define DESC_TEXT_ABOVE "View the background colors that can be produced with InstallShield's predefined constants."
#define DESC_TEXT_BELOW "To change the background color, select a color; then click the Set button. Or double-click the color name."

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

// Script-defined function to create color list.
prototype CreateColorList ();

// Script-defined functions to change background color.
prototype SetBackgroundColor (STRING);

export prototype ExFn_CtrlSetList(HWND);

function ExFn_CtrlSetList(hMSI)
    STRING szDialogName, svCurSel;
    NUMBER nCmdValue, nResult;
    BOOL    bDone;
    LIST    listBackgroundColors;
    HWND    hwndDlg;
begin

    Enable ( BACKGROUND );

    // Specify a name to identify the custom dialog in this installation.
    szDialogName = "CustomDialog";

    // Define the dialog. Pass a null string in the second parameter
    // to get the dialog from _isuser.dll or _isres.dll. Pass a null
    // string in the third parameter because the dialog is identified
    // by its ID in the fourth parameter.
    nResult = EzDefineDialog (szDialogName, "", "", RES_DIALOG_ID);

    if (nResult < 0) then
        // Report an error; then terminate.
        MessageBox ("Error in defining dialog", SEVERE);
        abort;
    endif;

```



```

// Call script-defined function to create color list.
listBackgroundColors = CreateColorList ();

if (listBackgroundColors = LIST_NULL) then
    MessageBox ("Unable to create list of background colors", SEVERE);
    abort;
endif;

// Initialize indicator used to control the loop.
bDone = FALSE;

repeat

    // Display the dialog and return the next dialog event.
    nCmdValue = WaitOnDialog (szDialogName);

    // Respond to the event.
    switch (nCmdValue)
    case DLG_CLOSE:
        // The user clicked the window's Close button.
        bDone = TRUE;
    case DLG_ERR:
        MessageBox ("Dialog failed", SEVERE);
        bDone = TRUE;
    case DLG_INIT:
        // Initialize the back, next, and cancel button enable/disable states
        // for this dialog and replace %P, %VS, %VI with
        // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
        // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
        hwndDlg = CmdGetHwndDlg(szDialogName);
        SdGeneralInit(szDialogName, hwndDlg, 0, "");

        // Set the window title.
        SetWindowText (hwndDlg, "View Program Folders");

        //Set the dialog's static text.
        CtrlSetText (szDialogName, RES_TEXT_ABOVE, DESC_TEXT_ABOVE);
        CtrlSetText (szDialogName, RES_TEXT_BELOW, DESC_TEXT_BELOW);
        CtrlSetText (szDialogName, RES_PBUTTON_SET, "&Set");
        CtrlSetText (szDialogName, RES_PBUTTON_DONE, "&Done");

        // Disable the Back button using a call from WinSub.
        _WinSubEnableControl (hwndDlg, RES_PBUTTON_BACK, 0);

        // Place the list of colors into the dialog's list box.
        nResult = CtrlSetList (szDialogName, RES_DIALOG_LISTBOX,
                               listBackgroundColors);
        if (nResult != 0) then
            // Handle error from CtrlSetList.
            MessageBox ("Unable to create folder name list.", SEVERE);
            bDone = TRUE;
        endif;

        // Destroy the color list.
        ListDestroy (listBackgroundColors);
    case RES_DIALOG_LISTBOX:
        // If the end user double-clicked a color, display it.
        if (CtrlGetSubCommand (szDialogName) = LISTBOX_ENTER) then

```

```

        CtrlGetCurSel (szDialogName, RES_DIALOG_LISTBOX, svCurSel);
        SetBackgroundColor (svCurSel);
    endif;
case RES_PBUTTON_DONE:
    bDone = TRUE;
case RES_PBUTTON_SET :
    // Display the selected color.
    CtrlGetCurSel (szDialogName, RES_DIALOG_LISTBOX, svCurSel);
    SetBackgroundColor (svCurSel);
endswitch;

until bDone;

end;

/*-----*\
*
* Script-defined functions begin here.
*
*-----*/

// CreateColorList returns a list of background color constants.
function CreateColorList ()
    LIST listBkColors;
begin
    // Create a list to hold the colors constants;
    listBkColors = ListCreate (STRINGLIST);

    // Build the list of color constants.
    if (listBkColors != LIST_NULL) then
        ListAddString (listBkColors, "BK_BLUE", AFTER);
        ListAddString (listBkColors, "BK_GREEN", AFTER);
        ListAddString (listBkColors, "BK_MAGENTA", AFTER);
        ListAddString (listBkColors, "BK_ORANGE", AFTER);
        ListAddString (listBkColors, "BK_RED", AFTER);
        ListAddString (listBkColors, "BK_YELLOW", AFTER);
        ListAddString (listBkColors, "BK_SOLIDBLACK", AFTER);
        ListAddString (listBkColors, "BK_SOLIDBLUE", AFTER);
        ListAddString (listBkColors, "BK_SOLIDGREEN", AFTER);
        ListAddString (listBkColors, "BK_SOLIDMAGENTA", AFTER);
        ListAddString (listBkColors, "BK_SOLIDORANGE", AFTER);
        ListAddString (listBkColors, "BK_SOLIDPINK", AFTER);
        ListAddString (listBkColors, "BK_SOLIDRED", AFTER);
        ListAddString (listBkColors, "BK_SOLIDWHITE", AFTER);
        ListAddString (listBkColors, "BK_SOLIDYELLOW", AFTER);
    endif;

    // Return the pointer to the list.
    return listBkColors;
end;

// SetBackgroundColor sets the background to the color
// specified by szColor.
function SetBackgroundColor (szColor)
    NUMBER nColor;
begin
    // Determine which color the end user selected.
    if szColor = "BK_BLUE" then
        nColor = BK_BLUE;

```

```

elseif szColor = "BK_GREEN" then
    nColor = BK_GREEN;
elseif szColor = "BK_MAGENTA" then
    nColor = BK_MAGENTA;
elseif szColor = "BK_ORANGE" then
    nColor = BK_ORANGE;
elseif szColor = "BK_RED" then
    nColor = BK_RED;
elseif szColor = "BK_YELLOW" then
    nColor = BK_YELLOW;
elseif szColor = "BK_SOLIDBLACK" then
    nColor = BK_SOLIDBLACK;
elseif szColor = "BK_SOLIDBLUE" then
    nColor = BK_SOLIDBLUE;
elseif szColor = "BK_SOLIDGREEN" then
    nColor = BK_SOLIDGREEN;
elseif szColor = "BK_SOLIDMAGENTA" then
    nColor = BK_SOLIDMAGENTA;
elseif szColor = "BK_SOLIDORANGE" then
    nColor = BK_SOLIDORANGE;
elseif szColor = "BK_SOLIDPINK" then
    nColor = BK_SOLIDPINK;
elseif szColor = "BK_SOLIDRED" then
    nColor = BK_SOLIDRED;
elseif szColor = "BK_SOLIDWHITE" then
    nColor = BK_SOLIDWHITE;
elseif szColor = "BK_SOLIDYELLOW" then
    nColor = BK_SOLIDYELLOW;
endif;

// Set the background to the selected color.
SetColor (BACKGROUND, nColor);
end;

```

## CtrlSetMLEText

The **CtrlSetMLEText** function sets the text of a multi-line edit box control. InstallShield separately places each string in listID into the multi-line edit box control. This function is for use only with custom dialogs.

### Syntax

```
CtrlSetMLEText ( szDialogName, nControlID, listID );
```

## Parameters

**Table 121** • CtrlSetMLEText Parameters

Parameter	Description
<b>szDialogName</b>	Specifies the name of a dialog.
<b>nControlID</b>	Specifies the resource ID of the multi-line edit box control in the dialog.
<b>listID</b>	Specifies the name of a valid string list that contains the elements to be copied into the multi-line edit control.

## Return Values

**Table 122** • CtrlSetMLEText Return Values

Return Value	Description
<b>0</b>	CtrlSetMLEText set the text into the control.
<b>&lt; 0</b>	CtrlSetMLEText was unable to set the text in the control.

## CtrlSetMLEText Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the CtrlSetMLEText and CtrlGetMLEText functions.
*
* This example script displays a custom dialog that contains
* a multi-line edit box. The script creates a list of all
* program folders on the target system and then calls
* CtrlSetMLEText to place that list into the dialog's multi-
* line edit box. The dialog also has a Save button that
* enables the end user to save the folder names to a text file.
* When that option is selected, the script calls CtrlGetMLEText
* to get the folder names from the multi-line edit box.
*
* The "custom" dialog used in this script is actually the
* InstallShield Sd dialog that is displayed by the built-in
* function SdShowInfoList. Because this dialog is stored
* in the file _isres.dll, which is already compressed in
* the installation, it can be used in a script as a custom

```

```

* dialog.
*
* Notes: The multi-line edit box is defined as read-only
*       in the resource; its contents cannot be edited.
*
*       The script changes the static text of the dialog
*       box's Next button and disables the Back button to
*       make the dialog fit the needs of the example.
*
*       The function GetGroupNameList may return an error
*       if the target system is running under a shell other
*       than the Explorer shell.
*
\*-----*/

// Dialog and control IDs.
#define RES_DIALOG_ID      12007 // ID of the custom dialog
#define RES_PBUT_BACK      12    // ID of Next button
#define RES_PBUT_DONE      9     // ID of Cancel button
#define RES_PBUT_SAVE      1     // ID of Back button
#define RES_DIALOG_EDITBOX 301   // ID of edit box
#define RES_TEXT            711  // ID of text above edit box

// Description to display above the multi-line edit box.
#define DESC_TEXT "Click Save to store the list of program folder names in a disk file."

// The program names will be saved in the root of the current
// drive if the end user clicks the Save button.
#define FOLDER_LIST_FILE "\\ISExampl.txt"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_CtrlSetMLEText(HWND);

function ExFn_CtrlSetMLEText(hMSI)
    STRING szDialogName;
    NUMBER nCmdValue, nResult;
    BOOL    bSave, bDone;
    LIST    listFolders;
    HWND    hwndDlg;
begin

    // Specify a name to identify the custom dialog in this installation.
    szDialogName = "CustomDialog";

    // Define the dialog. Pass a null string in the second parameter
    // to get the dialog from _isuser.dll or _isres.dll. Pass a null
    // string in the third parameter because the dialog is identified
    // by its ID in the fourth parameter.
    nResult = EzDefineDialog (szDialogName, "", "", RES_DIALOG_ID);

    if (nResult < 0) then
        // Report an error; then terminate.
        MessageBox ("Error in defining dialog", SEVERE);
        abort;
    endif;

    // Initialize indicator used to control the while loop.

```

```

bDone = FALSE;

// Loop until done.
repeat

    // Display the dialog and return the next dialog event.
    nCmdValue = WaitOnDialog (szDialogName);

    // Respond to the event.
    switch (nCmdValue)
    case DLG_CLOSE:
        // The user clicked the window's Close button.
        bDone = TRUE;
    case DLG_ERR:
        MessageBox ("Dialog failed", SEVERE);
        bDone = TRUE;
    case DLG_INIT:
        // Initialize the back, next, and cancel button enable/disable states
        // for this dialog and replace %P, %VS, %VI with
        // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
        // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
        hwndDlg = CmdGetHwndDlg(szDialogName);
        SdGeneralInit(szDialogName, hwndDlg, 0, "");

        // Set the window title.
        SetWindowText (hwndDlg, "View Program Folders");

        // Disable the Back button using a call from WinSub.
        _WinSubEnableControl (hwndDlg, RES_PBUT_BACK, 0);

        //Set the dialog's static text.
        CtrlSetText (szDialogName, RES_TEXT, DESC_TEXT);
        CtrlSetText (szDialogName, RES_PBUT_SAVE, "&Save");
        CtrlSetText (szDialogName, RES_PBUT_DONE, "&Done");

        // Create a string list to store the program folder names.
        listFolders = ListCreate (STRINGLIST);

        if (listFolders = LIST_NULL) then
            MessageBox ("Unable to create list.", SEVERE);
            bDone = TRUE;
        else
            // Get the folder names into a list.
            nResult = GetGroupNameList (listFolders);

            if (nResult = 0) then
                // Put the folder names into the
                // dialog's multi-line edit box.
                nResult = CtrlSetMLEText (szDialogName, RES_DIALOG_EDITBOX,
                                         listFolders);
            elseif (nResult != 0) then
                // Handle error from GetGroupNameList or CtrlSetMLEText.
                MessageBox ("Unable to create folder name list.", SEVERE);
                bDone = TRUE;
            endif;

            // Destroy the listID string list.
            ListDestroy (listFolders);
        endif;
    endif;

```

```

case RES_PBUT_SAVE :
    // Initialize indicator to save program file names.
    bSave = FALSE;

    if (AskYesNo("Save list as " + FOLDER_LIST_FILE + "?", YES)) then
        // Check for existing file.
        if (Is (FILE_EXISTS, FOLDER_LIST_FILE) = 1) then
            // Query end user to overwrite existing file.
            if (AskYesNo ("Overwrite existing " + FOLDER_LIST_FILE +
                "?", YES)) then
                bSave = TRUE;
            endif;
        else
            bSave = TRUE;
        endif;
    endif;

    if bSave = TRUE then
        // Create a string list to store list from dialog.
        listFolders = ListCreate (STRINGLIST);

        if (listFolders = LIST_NULL) then
            MessageBox ("Unable to create list.", SEVERE);
        else
            // Get the folder names from the
            // dialog's multi-line edit box.
            nResult = CtrlGetMLEText (szDialogName, RES_DIALOG_EDITBOX,
                listFolders);

            // Save the list to a text file.
            ListWriteToFile (listFolders, FOLDER_LIST_FILE);

            // Destroy the listID string list.
            ListDestroy (listFolders);
        endif;
    endif;
case RES_PBUT_DONE:
    bDone = TRUE;
endswitch;

until bDone;

// Close the custom dialog.
EndDialog (szDialogName);

// Remove the custom dialog from memory.
ReleaseDialog (szDialogName);

end;

```

## CtrlSetMultCurSel

The **CtrlSetMultCurSel** function searches the specified multi-selection list or combo box control. If `nSelectFlag` is set to `TRUE`, `CtrlSetMultCurSel` selects (highlights) the item when it is found. This function is for use only with custom dialogs.

## Syntax

```
CtrlSetMultCurSel (szDialogName, nControlID, szText, nSelectFlag);
```

## Parameters

**Table 123** • CtrlSetMultCurSel Parameters

Parameter	Description
<b>szDialogName</b>	Specifies the name of a custom dialog.
<b>nControlID</b>	Specifies the resource ID of the multi-selection list box control in the dialog.
<b>szText</b>	Specifies the search string.
<b>nSelectFlag</b>	<p>Indicates whether or not to select an item when CtrlSetMultCurSel finds it. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li><b>TRUE</b>—Indicates that the item is to be selected.</li> <li><b>FALSE</b>—Indicates that the item is not to be selected.</li> </ul>

## Return Values

**Table 124** • CtrlSetMultCurSel Return Values

Return Value	Description
<b>0</b>	CtrlSetMultCurSel successfully found the text in the control and either selected it or did not select it, as indicated in nSelectFlag.
<b>&lt; 0</b>	CtrlSetMultCurSel was unable to find the text in the control.

## CtrlSetMultCurSel Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
```



```

* Demonstrates the CtrlSetMultCurSel and CtrlGetMultCurSel
* functions.
*
* This script retrieves the names of all program folders
* on the target system and places them in a list. When the
* dialog is initialized, the CtrlSetList function sets this
* list to be displayed in the list box. The CtrlSetMultCurSel
* function is then called to select the user-selected
* folder.
*
* This list is then destroyed. A new list is created when the
* Next button is clicked. CtrlGetMultCurSel then retrieves the
* elements in the list box and assigns them to this new string
* list. This list is then displayed in an Sd dialog.
*
* Note: In order for this script to run properly, you must set
*       the RES_DIALOG_ID and RES_DIALOG_LISTBOX constants to a
*       dialog and list box created in _isuser.dll.
*
*       The GetGroupNameList function used in this example may
*       return an error if the target system is running under a
*       shell other than the Explorer shell.
*
\*-----*/

// Dialog controls.
#define RES_DIALOG_ID           // ID of dialog
#define RES_PBUT_NEXT          1 // ID of Next button
#define RES_PBUT_CANCEL        9 // ID of Cancel button
#define RES_PBUT_BACK          12 // ID of Back button
#define RES_DIALOG_LISTBOX     // ID of list box

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_CtrlSetMultCurSel(HWND);

function ExFn_CtrlSetMultCurSel(hMSI)
    STRING szDialogName, szDLL, szTitle, szMsg;
    STRING szText, szDefFolder, svResultFolder;
    NUMBER nCmdValue, nResult, nControlID, nSelectFlag;
    BOOL bDone;
    LIST listID, listFolders;
    HWND hwndDlg;
begin

    Disable(BACKBUTTON);

    szDialogName = "CtrlSetMultCurSel";
    szDLL        = "";

    // Define the dialog. Pass a null string in the second parameter
    // to get the dialog from _isuser.dll or _isres.dll. Pass a null
    // string in the third parameter because the dialog is identified
    // by its ID in the fourth parameter.
    nResult = EzDefineDialog (szDialogName, szDLL, "", RES_DIALOG_ID);

    if (nResult < 0) then
        MessageBox ("Error in defining dialog", SEVERE);

```

```

        bDone = TRUE;
    else
        bDone = FALSE;
    endif;

    // Create the listID string list.
    listID = ListCreate (STRINGLIST);

    if (listID = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
    else
        MessageBox ("listID created.", INFORMATION);
    endif;

    // Retrieve the program folder names into a list.
    GetGroupNameList (listID);

    // Retrieve a folder name from the user.
    szTitle = "CtrlGetMultCurSel & CtrlSetMultCurSel";
    SelectFolder (szTitle, szDefFolder, svResultFolder);

    // Loop until done.
    while (bDone = FALSE)

        // Display the dialog and return the next dialog event.
        nCmdValue = WaitOnDialog (szDialogName);

        // Respond to the event.
        switch (nCmdValue)
        case DLG_ERR:
            MessageBox ("Unable to display dialog. Setup canceled.",SEVERE);
            abort;
        case DLG_INIT:
            // Initialize the back, next, and cancel button enable/disable states
            // for this dialog and replace %P, %VS, %VI with
            // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
            // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
            hwndDlg = CmdGetHwndDlg(szDialogName);
            SdGeneralInit(szDialogName, hwndDlg, 0, "");

            // The following sets the list box to the list of program folders.
            nControlID = RES_DIALOG_LISTBOX;
            CtrlSetList (szDialogName, nControlID, listID);

            szText = svResultFolder;
            nSelectFlag = TRUE;

            //Set the user-selected folder to be highlighted.
            if (CtrlSetMultCurSel (szDialogName, nControlID, szText,
                                   nSelectFlag) < 0) then
                MessageBox ("CtrlSetMultCurSel failed.", SEVERE);
            endif;

            // Destroy the listID string list.
            ListDestroy (listID);
            MessageBox ("listID destroyed.", INFORMATION);
        case DLG_CLOSE:
            // The user clicked the window's Close button.
            Do (EXIT);

```

```

case RES_PBUT_NEXT:
    // Create the listFolders string list.
    listFolders = ListCreate (STRINGLIST);

    if (listFolders = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
    else
        MessageBox ("listFolders created.", INFORMATION);
    endif;

    // Retrieve the highlighted elements in the list box and
    // put them into the listFolders string list.
    if (CtrlGetMultCurSel (szDialogName, nControlID,
        listFolders) < 0) then
        MessageBox ("CtrlGetMultCurSel failed.", SEVERE);
    else
        MessageBox ("CtrlGetMultCurSel successful.", INFORMATION);
    endif;

    bDone = TRUE;
case RES_PBUT_BACK:
    bDone = TRUE;
case RES_PBUT_CANCEL:
    // The user clicked the window's Cancel button.
    Do (EXIT);
endswitch;

endwhile;

szMsg  = "The following are the elements highlighted in the list box:";

// Display the list of elements highlighted.
SdShowInfoList (szTitle, szMsg, listFolders);

// Remove listFolders string list from memory.
ListDestroy (listFolders);
MessageBox ("listFolders destroyed.", INFORMATION);

// Close the dialog.
EndDialog (szDialogName);

// Remove dialog from memory.
ReleaseDialog (szDialogName);

end;

```

## CtrlSetState

The **CtrlSetState** function sets the current state of a check box or option button control in a custom dialog. You can set certain characteristics of option buttons and check boxes when you create them using a resource or dialog editor. If you experience difficulties with the behavior of a button control, check the characteristics of the control in the editor.

### Syntax

```
CtrlSetState ( szDialogName, nControlID, nState );
```

## Parameters

**Table 125** • CtrlSetState Parameters

Parameter	Description
<b>szDialogName</b>	Specifies the name of a dialog that contains the check box or option button control.
<b>nControlID</b>	Specifies the resource ID of the check box or option button control.
<b>nState</b>	Specifies the new state of the button control. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>BUTTON_CHECKED</b>—Sets the button's state to CHECKED.</li> <li>• <b>BUTTON_UNCHECKED</b>—Sets the button's state to UNCHECKED.</li> </ul>

## Return Values

**Table 126** • CtrlSetState Return Values

Return Value	Description
<b>0</b>	CtrlSetState successfully set the state of the check box or option button control.
<b>&lt; 0</b>	CtrlSetState was unable to set the state of the control.

## CtrlSetState Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the CtrlGetState and CtrlSetState functions.
*
* This example script displays a custom dialog that contains
* four check boxes. The script calls CtrlSetState to set the
* first two check boxes to checked. The last two are unchecked
* by default. When the end user clicks the Next button, the
* script calls CtrlGetState to retrieve the state of each
* each check box. The script then displays a message box that
```

```

* reports which check boxes were selected.
*
* The "custom" dialog used in this script is actually the
* InstallShield Sd dialog that is displayed by the built-in
* function SdAskOptions. Because this dialog is stored in
* the file _isres.dll, which is already compressed in
* the installation, it can be used in a script as a custom
* dialog.
*
\*-----*/

// Dialog and control IDs.
#define RES_DIALOG_ID      12020  // ID of the custom dialog
#define RES_PBUT_NEXT      1      // ID of Next button
#define RES_PBUT_CANCEL    9      // ID of Cancel button
#define RES_PBUT_BACK      12     // ID of Back button
#define ID_OP1_CHECK       501    // ID of Option 1 check box
#define ID_OP2_CHECK       502    // ID of Option 2 check box
#define ID_OP3_CHECK       503    // ID of Option 3 check box
#define ID_OP4_CHECK       504    // ID of Option 4 check box
#define ID_STA_DESC        711    // ID of static text description

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_CtrlSetState(HWND);

function ExFn_CtrlSetState(hMSI)
    STRING  szDialogName, szMsg;
    NUMBER  nResult, nCmdValue, hwndDlg;
    BOOL    bDone;
begin

    // Specify a name to identify the custom dialog in this installation.
    szDialogName = "ExDialog";

    // Define the dialog. Pass a null string in the second parameter
    // to get the dialog from _isuser.dll or _isres.dll. Pass a null
    // string in the third parameter because the dialog is identified
    // by its ID in the fourth parameter.
    nResult = EzDefineDialog (szDialogName, "", "", RES_DIALOG_ID);

    if (nResult < 0) then
        // Report an error; then terminate.
        MessageBox ("Error in defining dialog", SEVERE);
        abort;
    endif;

    // Initialize the indicator used to control the loop.
    bDone = FALSE;

    repeat

        // Display the dialog and return the next dialog event.
        nCmdValue = WaitOnDialog (szDialogName);

        // Respond to the event.
        switch (nCmdValue)

```

```

case DLG_CLOSE:
    // The user clicked the window's Close button.
    Do (EXIT);
case DLG_ERR:
    MessageBox ("Unable to display dialog. Setup canceled.", SEVERE);
    abort;
case DLG_INIT:
    // Initialize the back, next, and cancel button enable/disable states
    // for this dialog and replace %P, %VS, %VI with
    // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
    // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
    hwndDlg = CmdGetHwndDlg(szDialogName);
    SdGeneralInit(szDialogName, hwndDlg, 0, "");

    // Set the window title.
    SetWindowText (hwndDlg, "Select Options");

    // Set static text description displayed above check boxes.
    CtrlSetText (szDialogName, ID_STA_DESC,
        "Select and/or clear options. Then click Next.");

    // Options are cleared by default, so select Options 1 and 2.
    if (CtrlSetState (szDialogName, ID_OP1_CHECK, BUTTON_CHECKED) < 0) then
        MessageBox ("First call to CtrlSetState failed.", SEVERE);
        bDone = TRUE;
    elseif (CtrlSetState (szDialogName, ID_OP2_CHECK, BUTTON_CHECKED) < 0) then
        MessageBox ("Second call to CtrlSetState failed.", SEVERE);
        bDone = TRUE;
    endif;

case RES_PBUT_NEXT:
    bDone = TRUE;
case RES_PBUT_CANCEL:
    // The user clicked the Cancel button.
    Do (EXIT);
case RES_PBUT_BACK:
    bDone = TRUE;
endswitch;

until bDone;

// Build message if end user clicked Next button.
if (nCmdValue = RES_PBUT_NEXT) then
    // Start building the message to display to the end user.
    szMsg = "You selected the following items:\n\n";

    // If first option is selected, add line to message.
    if (CtrlGetState (szDialogName, ID_OP1_CHECK) = BUTTON_CHECKED) then
        szMsg = szMsg + "Option 1\n";
    endif;

    // If second option is selected, add line to message.
    if (CtrlGetState (szDialogName, ID_OP2_CHECK) = BUTTON_CHECKED) then
        szMsg = szMsg + "Option 2\n";
    endif;

    // If third option is selected, add line to message.
    if (CtrlGetState (szDialogName, ID_OP3_CHECK) = BUTTON_CHECKED) then
        szMsg = szMsg + "Option 3\n";

```

```

endif;

// If fourth option is selected, add line to message.
if (CtrlGetState (szDialogName, ID_OP4_CHECK) = BUTTON_CHECKED) then
    szMsg = szMsg + "Option 4\n";
endif;
endif;

// Close the custom dialog.
EndDialog (szDialogName);

// Remove the custom dialog from memory.
ReleaseDialog (szDialogName);

// Display message if dialog was closed with Next button.
if (nCmdValue = RES_PBUT_NEXT) then
    MessageBox (szMsg, INFORMATION);
endif;

end;

```

## CtrlSetText


The **CtrlSetText** function sets the text of a single-line edit field, static text field, or button control in a custom dialog. To set the text in multi-line edit fields, call [CtrlSetMLEText](#).

### Syntax

```
CtrlSetText ( szDialogName, nControlID, szText );
```

## Parameters

**Table 127 •** CtrlSetText Parameters

Parameter	Description
<b>szDialogName</b>	Specify the name of the dialog that you want to modify.
<b>nControlID</b>	Specify the resource ID of the single-line edit field, static text field, or button control in which text is to be set.
<b>szText</b>	Specify the text to place in the control.
	 <p><b>Tip •</b> If you want the control to be an HTML control, you can specify <b>[html]</b> at the beginning of the szText value. To learn more, see <i>Using an HTML Control on a Dialog</i>.</p>

## Return Values

**Table 128 •** CtrlSetText Return Values

Return Value	Description
<b>0</b>	CtrlSetText successfully set the text in the control.
<b>ISERR_GEN_FAILURE</b>	CtrlSetText was unable to set the text in the control.

## CtrlSetText Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the CtrlSetText, CtrlGetText, and CtrlSelectText
* functions.
*
* This example script displays a custom dialog with two
* edit boxes to obtain a user name and company name. The
* script calls CtrlSetText to place initial values into the
* edit boxes and CtrlSelectText to select the contents of the
* first edit box. When the user clicks the Next button, the
* script calls CtrlGetText to retrieve the contents of the edit
* boxes so that they can be displayed in a message box after
* the custom dialog is closed.
*
* The "custom" dialog used in this script is actually the
* InstallShield Sd dialog that is displayed by the built-in
* function SdRegisterUser. Because this dialog is stored in
```



```

* the file _isres.dll, which is already compressed in
* the installation, it can be used in a script as a custom
* dialog.
*
\*-----*/

// Dialog and control IDs.
#define RES_DIALOG_ID      12001 // ID of custom dialog
#define RES_PBUT_NEXT      1 // ID of Next button
#define RES_PBUT_CANCEL    9 // ID of Cancel button
#define RES_PBUT_BACK      12 // ID of Back button
#define RES_EDITNAME       301 // ID of edit box
#define RES_EDITCOMPANY    302 // ID of edit box

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_CtrlSetText(HWND);

function ExFn_CtrlSetText(hMSI)
    STRING szDialogName, svName, svCompany;
    NUMBER nResult, nCmdValue;
    BOOL    bDone;
    HWND    hwndDlg;
begin

    // Specify a name to identify the custom dialog in this installation.
    szDialogName = "CustomDialog";

    // Define the dialog. Pass a null string in the second parameter
    // to get the dialog from _isuser.dll or _isres.dll. Pass a null
    // string in the third parameter because the dialog is identified
    // by its ID in the fourth parameter.
    nResult = EzDefineDialog (szDialogName, "", "", RES_DIALOG_ID);

    if (nResult < 0) then
        // Report an error; then terminate.
        MessageBox ("Error in defining dialog", SEVERE);
        abort;
    endif;

    // Initialize indicator used to control the loop.
    bDone = FALSE;

    // Loop until done.
    repeat

        // Display the dialog and return the next dialog event.
        nCmdValue = WaitOnDialog (szDialogName);

        // Respond to the event.
        switch (nCmdValue)
            case DLG_CLOSE:
                // The user clicked the window's Close button.
                Do (EXIT);
            case DLG_ERR:
                MessageBox ("Unable to display dialog. Setup canceled.", SEVERE);
                abort;
            case DLG_INIT:

```

```

// Initialize the back, next, and cancel button enable/disable states
// for this dialog and replace %P, %VS, %VI with
// IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
// IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
hwndDlg = CmdGetHwndDlg(szDialogName);
SdGeneralInit(szDialogName, hwndDlg, 0, "");

// Put initial values into the edit boxes.
CtrlSetText (szDialogName, RES_EDITNAME, "Your name");
CtrlSetText (szDialogName, RES_EDITCOMPANY, "Your company");

// Select the Name edit box.
CtrlSelectText (szDialogName, RES_EDITNAME);
case RES_PBUT_NEXT:
    // Get the contents of the edit boxes.
    CtrlGetText (szDialogName, RES_EDITNAME, svName);
    CtrlGetText (szDialogName, RES_EDITCOMPANY, svCompany);

    // Verify that both edit boxes have data.
    if (StrLength(svName) = 0) || (StrLength(svCompany) = 0) then
        MessageBox ("Both fields must be completed.", INFORMATION);
    else
        bDone = TRUE;
    endif;
case RES_PBUT_CANCEL:
    // The user clicked the Cancel button.
    Do (EXIT);
case RES_PBUT_BACK:
    bDone = TRUE;
endswitch;

until bDone;

// Close the dialog.
EndDialog (szDialogName);

// Remove the dialog from memory.
ReleaseDialog (szDialogName);

// If dialog is closed with Next button, display name and company.
if nCmdValue = RES_PBUT_NEXT then
    MessageBox (svName + "\n" + svCompany, INFORMATION);
endif;

end;

```

## DefineDialog

The **DefineDialog** function defines a custom dialog. Call this function instead of [EzDefineDialog](#) when you need to specify a dialog attribute that cannot be specified with **EzDefineDialog**.




**Note • DefineDialog** does not display the custom dialog. To display a custom dialog, you must call [WaitOnDialog](#).

## Syntax


```
DefineDialog ( szDialogName, hInstance, szDLLName, nDialogID, szDialogID, nReserved, hwndOwner,  
              lMsgLevel );
```

## Parameters

**Table 129 • DefineDialog Parameters**

Parameter	Description
<b>szDialogName</b>	<p>Specifies the name to associate with the dialog identified by szDialogID or nDialogID. To process this dialog, use this name in subsequent calls to custom dialog functions.</p> <p>Note that the dialog's name is case-sensitive; you must use it exactly as you have specified it in this parameter.</p>
<b>hInstance</b>	<p>Specifies the instance handle of the .dll file in which the dialog resides. If you specify the fully qualified name of the .dll file in szDLLName, you can specify 0 in this parameter. To obtain the instance handle of a .dll file, call the Microsoft Windows API <b>LoadLibrary</b>.</p>
<b>szDLLName</b>	<p>Specifies the name of the .dll file that contains the dialog resource. If this name is not qualified (that is, if you do not specify the drive and path with the file name), the InstallScript engine searches for the .dll file in the Windows folder. If it is not found there, the InstallScript engine searches the folders that are specified in the search path. When the dialog is located in <b>_isuser.dll</b>, you can specify a null string (""), the InstallScript engine automatically checks <b>_isuser.dll</b> if this parameter is specified as a null string ("").</p> <div>  <p><b>Note •</b> When you use a .dll file other than <b>_isres.dll</b> or <b>_isuser.dll</b>, you must call <a href="#">UseDLL</a> to load the .dll file before calling <b>DefineDialog</b>. To unload the .dll file from memory, call <a href="#">UnUseDLL</a> after calling <a href="#">ReleaseDialog</a>.</p> </div>
<b>nDialogID</b>	<p>Specifies the dialog's resource ID if you use a number rather than a string to identify the resource. This parameter is used only when szDialogID is a null string (""). It is recommended that you use nDialogID rather than szDialogID to identify the dialog resource.</p>

**Table 129** • DefineDialog Parameters (cont.)

Parameter	Description
<b>szDialogID</b>	<p>Specifies the dialog's resource ID if you use a string rather than a number to identify the resource. If this parameter is a null string (""), nDialogID is used to identify the dialog resources. It is recommended that you use nDialogID rather than szDialogID to identify the dialog resource.</p> <p></p> <p><b>Note</b> • If you created the dialog in the Dialogs view, the dialog is created with a string ID; therefore, you must specify this name as the szDialogID parameter to define the dialog.</p> <p>If you override an existing dialog in the Dialogs view, the dialog has a numeric ID, and you must specify the ID in the nDialogID parameter.</p> <p>If you want to use numeric IDs in all cases—which is what is recommended—you must edit the dialog in the Dialogs view to change the ISResourceID property of the created dialog to the desired Dialog ID instead of 0. Note that if you do this, the dialog name is no longer used when the dialog is created; it is used only in the Dialogs view in InstallShield to identify the dialog. The dialog is built with the correct numeric ID.</p>
<b>nReserved</b>	Pass zero in this parameter. No other value is allowed.
<b>hwndOwner</b>	Specifies the window handle of the owner window. Specify HWND_INSTALL in this parameter to make the main installation window the owner of the dialog.
<b>IMsgLevel</b>	<p>This parameter specifies which windows messages will be sent to the dialog. You must use the OR operator ( ) to combine one of the following constants with the constant DLG_CENTERED.</p> <ul style="list-style-type: none"> <li>• <b>DLG_MSG_STANDARD</b>—Filters out most Windows messages; only those directly related to the dialog's controls are passed to the dialog.</li> <li>• <b>DLG_MSG_ALL</b>—Passes most Windows messages.</li> </ul>

## Return Values

**Table 130** • DefineDialog Return Values

Return Value	Description
<b>0</b>	<b>DefineDialog</b> successfully defined the dialog.
<b>DLG_ERR_ALREADY_EXISTS (-3)</b>	Indicates that you are trying to define a dialog that has already been defined in the installation script. You cannot define two dialogs with the same name.
<b>DLG_ERR (-1)</b>	Indicates an unspecified error condition.

## DefineDialog Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the DefineDialog, EndDialog, and ReleaseDialog
* functions.
*
* This script opens a simple custom dialog that displays
* a bitmap. The dialog can be closed with any of three
* buttons: Back, Next, or Cancel.
*
* The "custom" dialog used in this script is actually the
* InstallShield Sd dialog that is displayed by the built-in
* function SdBitmap. Because this dialog is stored in
* the file _isres.dll, which is already compressed in
* the installation, it can be used in a script as a custom
* dialog.
*
* In order to use this dialog as a custom dialog, the
* script first defines it by calling DefineDialog. It then
* displays the dialog by calling WaitOnDialog. When an event
* ends dialog processing, EndDialog is called to close the
* dialog. Then the dialog is released from memory by
* a call to ReleaseDialog.
*
\*-----*/

// Dialog and control IDs.
#define RES_DIALOG_ID      12027 // ID of dialog itself
#define RES_PBUT_NEXT      1    // ID of Next button
#define RES_PBUT_CANCEL    9    // ID of Cancel button
#define RES_PBUT_BACK      12   // ID of Back button

// Include Ifx.h for built-in InstallScript function prototypes.
```

```

#include "Ifx.h"

export prototype ExFn_DefineDialog(HWND);

function ExFn_DefineDialog(hMSI)
    STRING  szDialogName, szDLLName, szDialog;
    NUMBER  nDialog, nResult, nCmdValue;
    BOOL    bDone;
    HWND    hInstance, hwndParent, hwndDlg;
begin

    // Define the name of a dialog to pass as first
    // parameter to DefineDialog.
    szDialogName = "ExampleDialog";

    // DefineDialog's second parameter will be 0 because the
    // .dll file is in _isres.dll.
    hInstance = 0;

    // DefineDialog's third parameter will be null; installation will
    // search for the dialog in _isuser.dll and _isres.dll.
    szDLLName = "";

    // DefineDialog's fifth parameter will be null because the
    // dialog is identified by its ID in the fourth parameter.
    szDialog = "";

    // This value is reserved and must be 0.
    hwndParent = 0;

    // Define the dialog. The installation's main window will own the
    // dialog (indicated by HWND_INSTALL in parameter 7).
    nResult = DefineDialog (szDialogName, hInstance, szDLLName,
                           RES_DIALOG_ID, szDialog, hwndParent,
                           HWND_INSTALL, DLG_MSG_STANDARD|DLG_CENTERED);

    // Check for an error.
    if (nResult < 0) then
        MessageBox ("An error occurred while defining the dialog.", SEVERE);
        bDone = TRUE;
        abort;
    endif;

    // Initialize the indicator used to control the while loop.
    bDone = FALSE;

    // Loop until done.
    repeat

        // Display the dialog and return the next dialog event.
        nCmdValue = WaitOnDialog(szDialogName);

        // Respond to the event.
        switch (nCmdValue)
            case DLG_CLOSE:
                // The user clicked the window's Close button.
                Do (EXIT);
            case DLG_ERR:
                MessageBox ("Unable to display dialog. Setup canceled.", SEVERE);

```

```

        abort;
    case DLG_INIT:
        // Initialize the back, next, and cancel button enable/disable states
        // for this dialog and replace %P, %VS, %VI with
        // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
        // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
        hwndDlg = CmdGetHwndDlg(szDialogName);
        SdGeneralInit(szDialogName, hwndDlg, 0, "");

    case RES_PBUT_CANCEL:
        // The user clicked the Cancel button.
        Do (EXIT);
    case RES_PBUT_NEXT:
        bDone = TRUE;
    case RES_PBUT_BACK:
        bDone = TRUE;
        // check standard handling
        if (SdIsStdButton( nCmdValue ) && SdDoStdButton( nCmdValue )) then
            bDone = TRUE;
        endif;
    endswitch;

until bDone;

// Close the dialog.
EndDialog (szDialogName);

// Free the dialog from memory.
ReleaseDialog (szDialogName);

end;

```

## DeinstallSetReference

The **DeinstallSetReference** function is obsolete.

If you want to check whether a file is locked during uninstallation, write script code that calls the **Is** function with the `FILE_LOCKED` constant for the `nlsFlag` parameter and that responds as needed.

### Syntax

```
DeinstallSetReference (szReferenceFile);
```

## DeinstallStart

The **DeinstallStart** function is obsolete. Installations create the required registry keys to enable uninstallation.

### Syntax

```
DeinstallStart (szObsolete, svObsolete, szObsolete, lReserved);
```



# Delay

The **Delay** function delays the execution of a script by a specified number of seconds. Other tasks running simultaneously with InstallShield proceed normally while InstallShield is delayed.

## Syntax

```
Delay ( nSeconds );
```

## Delay Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the Delay function.
*
* First, SdShowMsg is called to display a message box and Delay
* is called to pause the script for three seconds. Then the
* message box is removed and Delay is called again to pause for
* two seconds. Finally, another message is displayed for three
* seconds.
*
\*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_Delay(HWND);

function ExFn_Delay(hMSI)
begin

    SdShowMsg ("This message will be displayed for three seconds.", TRUE);

    Delay (3);

    SdShowMsg ("", FALSE);

    Delay (2);

    SdShowMsg ("This is another message that will be displayed for a mere " +
        "three seconds.", TRUE);

    Delay (3);

end;
```

# DeleteCharArray

## Description

The **DeleteCharArray** function deletes the array of pointers to which pCharArray points.

## Syntax

```
DeleteCharArray ( pCharArray );
```

## Parameters

**Table 131** • DeleteCharArray Parameters

Parameter	Description
<b>pCharArray</b>	Specifies a pointer to an array of pointers to ANSI character strings. Typically, this pointer is returned by a previous call to <a href="#">GetCharArrayFromISStringArray</a> .

## Return Values

**Table 132** • DeleteCharArray Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	This function always returns ISERR_SUCCESS.

# DeleteDir

The **DeleteDir** function deletes a subdirectory. Depending on the value you use in the parameter nFlag, you can delete a subdirectory only if it is empty, delete a subdirectory even if it contains files, or delete an entire root directory. Set nFlag with extreme caution.



**Note** • Note the following restrictions:

- You cannot use DeleteDir to delete the current directory.
- You cannot delete files on a network system where you lack the appropriate rights.
- DeleteDir cannot delete read-only, hidden, or system files.
- If DeleteDir encounters a read-only file, the function can fail after having deleted only some of the files in the subdirectory.

## Syntax

```
DeleteDir( szDir, nFlag );
```

## Parameters

**Table 133** • DeleteDir Parameters

Parameter	Description
<b>szDir</b>	Specifies the fully qualified name of the directory to delete.
<b>nFlag</b>	<p>Specifies deletion options. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>ALLCONTENTS</b>—Deletes the directory in szDir, including all subdirectories and files beneath it. The directory you are deleting must be a subdirectory and cannot be a root directory of the drive.</li> <li>• <b>ONLYDIR</b>—Deletes the directory in szDir only if it is empty. Otherwise, the function fails.</li> <li>• <b>ROOT</b>—Deletes the directory in szDir even if it is the root directory. If szDir is a root directory, DeleteDir will delete everything on the disk.</li> </ul>

## Return Values

**Table 134** • DeleteDir Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully deleted the subdirectory.
<b>&lt; 0</b>	Indicates that the function was unable to delete the subdirectory.

## DeleteDir Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the DeleteDir function.
```

```

*
* First, CreateDir is called to create a directory. Then,
* DeleteDir is called to delete it.
*
\*-----*/

#define EXAMPLE_DIR "C:\\Newdir"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_DeleteDir(HWND);

function ExFn_DeleteDir(hMSI)
begin
    // Create a directory.
    if (CreateDir (EXAMPLE_DIR) != 0) then
        // Report the error; then terminate.
        MessageBox ("Unable to create directory.", SEVERE);
    else
        // Report success.
        MessageBox (EXAMPLE_DIR + " was created.", INFORMATION);

        // Delete the directory. If the directory is not
        // empty, it is not deleted.
        if (DeleteDir (EXAMPLE_DIR, ONLYDIR) = 0) then
            // Report success.
            MessageBox (EXAMPLE_DIR + " was deleted.", INFORMATION);
        else
            MessageBox ("Unable to delete directory.", SEVERE);
        endif;
    endif;
end;

```

## DeleteFile

The **DeleteFile** function deletes one or more files.



**Project** • In a Basic MSI or InstallScript MSI project, this functionality might be better achieved by using the native Windows Installer RemoveFiles action. For more information, see the Windows Installer Help.



**Note** • Note the following:

- You cannot use **DeleteFile** to delete files on a network system where you lack the appropriate rights.
- **DeleteFile** cannot delete read-only, hidden, or system files.
- You can use wild-card characters with [FindFile](#) to locate files and then delete them with **DeleteFile**.

## Syntax

```
DeleteFile ( szFile );
```

## Parameters

**Table 135 • DeleteFile Parameters**

Parameter	Description
<b>szFile</b>	Specifies the names of the files to delete. If szFile specifies a fully qualified file name, that is, if it includes a path, <b>DeleteFile</b> will delete the file from specified directory. If szFile contains an unqualified file name, that is, without path information, <b>DeleteFile</b> deletes the file from the directory that is specified by the system variable TARGETDIR (in InstallScript installations) or INSTALLDIR (in Basic MSI and InstallScript MSI installations). You can include wildcard characters in szFile to delete more than one file.

## Return Values

**Table 136 • DeleteFile Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully deleted the specified file or files.
<b>ISERR_PATH_NOT_FOUND (0x80070003)</b>	The specified path was not found.
<b>ISERR_FILE_NOT_FOUND (0x80070004)</b>	The specified file was not found or no files matched the specified wildcard.
<b>All other negative values</b>	<p>Indicates that the function was unable to delete one or more of the specified files. In the case of multiple files, only the error for the last file that could not be deleted is returned. The system variable <a href="#">ERRORFILENAME</a> contains a semicolon delimited list of the files that could not be deleted.</p> <p>You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a>.</p>

## DeleteFile Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
```

```

*
* Demonstrates the DeleteFile function.
*
* First, DeleteFile is called to delete a specified file from a
* directory. Then it is called again to delete all files with
* the extension "sys" from the same directory.
*
* Note: Before running this script, create a directory named
*       ISExaml in the root of drive C. Then create a file
*       named ISExaml.txt in that directory. Finally, create
*       two or more files with the extension "sys" in that
*       directory. These files will be deleted by the script.
*
\*-----*/

#define DEL_DIR      "C:\\ISExaml"
#define DEL_FILE     "ISExaml.txt"
#define DEL_SYS_FILES "*.sys"
#define TITLE_TEXT   "DeleteFile example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_DeleteFile(HWND);

function ExFn_DeleteFile(hMSI)
    STRING szMsg;
begin

    // Delete the file specified by DEL_FILE from the target directory.
    if (DeleteFile (DEL_DIR ^ DEL_FILE) < 0) then
        MessageBox ("First call to DeleteFile failed.", SEVERE);
    else
        sprintfBox (INFORMATION, TITLE_TEXT, "%s was delete from %s.",
                    DEL_FILE, DEL_DIR);
    endif;

    // Delete the files specified by DEL_SYS_FILES
    // from the target directory.
    if (DeleteFile (DEL_SYS_FILES) < 0) then
        MessageBox ("Second call to DeleteFile failed.", SEVERE);
    else
        sprintfBox (INFORMATION, TITLE_TEXT,
                    "All files matching %s were delete from %s.",
                    DEL_SYS_FILES, DEL_DIR);
    endif;

end;

```

## DeleteFolderIcon

The [DeleteShortcut](#) function supersedes the **DeleteFolderIcon** function.

The **DeleteFolderIcon** function removes a shortcut from a folder.



**Note** • *DeleteFolderIcon cannot be used for Internet shortcuts.*

## Syntax

```
DeleteFolderIcon ( szProgramFolder, szItemName );
```

## Parameters

**Table 137** • DeleteFolderIcon Parameters

Parameter	Description
<b>szProgramFolder</b>	Specifies the name of the folder that contains the shortcut to remove.
<b>szItemName</b>	Specifies the name of the shortcut to delete.

## Return Values

**Table 138** • DeleteFolderIcon Parameters

Return Value	Description
<b>0</b>	Indicates that the function successfully deleted the specified shortcut.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .
<b>&lt; 0</b>	Indicates that the function was unable to delete the shortcut.

## DeleteFolderIcon Example



**Note** • *To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.*

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the DeleteFolderIcon and DeleteProgramFolder
* functions.
*
* This script deletes the 'Notepad Example' icon from the
* 'Example folder' folder. DeleteProgramFolder is then
* called again to delete this folder.
*
* Note: In order for this script to run properly, you must set
*       the preprocessor constants to a valid folder and icon
*       on the target system. To easily create this example
*       folder and icon, run the AddFolderIcon example #3.
```

```

*
\*-----*/

#define FOLDER "C:\\Windows\\Example folder"
#define ICON   "Notepad Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_DeleteFolderIcon(HWND);

function ExFn_DeleteFolderIcon(hMSI)
begin

    // Display the folder.
    ShowProgramFolder (FOLDER, SW_SHOW);
    Delay (3);

    // Delete the 'Notepad Example' icon.
    if (DeleteFolderIcon (FOLDER, ICON) < 0) then
        MessageBox ("DeleteFolderIcon failed.", SEVERE);
    endif;

    // Delete the 'Example folder' icon.
    if (DeleteProgramFolder (FOLDER) < 0) then
        MessageBox ("DeleteProgramFolder failed.", SEVERE);
    endif;

end;

```

## DeleteProgramFolder

The [DeleteShortcutFolder](#) function supersedes the **DeleteProgramFolder** function.

The **DeleteProgramFolder** function deletes a program folder (that is, a subfolder of the Start menu's Programs folder) and its contents, including all shortcuts and all of the program folder's subfolders and their contents.

DeleteProgramFolder cannot delete the Programs folder.



**Tip** • In a Windows Installer based or InstallScript MSI project, this functionality is possibly better achieved by using the native Windows Installer RemoveFolders action. Or, if you are simply uninstalling, the MSI engine natively handles the removal of all files and folders created during the setup. For more information on the RemoveFolders action, see the Windows Installer Help.

### Syntax

```
DeleteProgramFolder ( szFolderName );
```



## Parameters

**Table 139** • DeleteProgramFolder Parameters

Parameter	Description
<b>szFolderName</b>	Specifies the name of the folder to remove.

## Return Values

**Table 140** • DeleteProgramFolder Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully removed the specified folder.
<b>&lt; 0</b>	Indicates that the function was unable to remove the folder.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

## DeleteProgramFolder Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the DeleteFolderIcon and DeleteProgramFolder
* functions.
*
* This script deletes the 'Notepad Example' icon from the
* 'Example folder' folder. DeleteProgramFolder is then
* called again to delete this folder.
*
* Note: In order for this script to run properly, you must set
*       the preprocessor constants to a valid folder and icon
*       on the target system. To easily create this example
*       folder and icon, run the AddFolderIcon example #3.
*
\*-----*/

#define FOLDER "C:\\Windows\\Example folder"
#define ICON   "Notepad Example"

// Include Ifx.h for built-in InstallScript function prototypes.
```

```

#include "Ifx.h"

export prototype ExFn_DeleteProgramFolder(HWND);

function ExFn_DeleteProgramFolder(hMSI)
begin

    // Display the folder.
    ShowProgramFolder (FOLDER, SW_SHOW);
    Delay (3);

    // Delete the 'Notepad Example' icon.
    if (DeleteFolderIcon (FOLDER, ICON) < 0) then
        MessageBox ("DeleteFolderIcon failed.", SEVERE);
    endif;

    // Delete the 'Example folder' icon.
    if (DeleteProgramFolder (FOLDER) < 0) then
        MessageBox ("DeleteProgramFolder failed.", SEVERE);
    endif;

end;

```

## DeleteShortcut

The **DeleteShortcut** function removes a shortcut from a folder.




---

**Note • DeleteShortcut** cannot be used for Internet shortcuts.

### Syntax

```
DeleteShortcut (szShortcutFolder, szName);
```

## Parameters

**Table 141** • DeleteShortcut Parameters

Parameter	Description
<b>szShortcutFolder</b>	Specify the name of the folder that contains the shortcut that you want to remove.
<b>szName</b>	Specify the name of the shortcut that you want to remove.

## Return Values

**Table 142** • DeleteShortcut Parameters

Return Value	Description
<b>0</b>	Indicates that the function successfully deleted the specified shortcut.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .
<b>&lt; 0</b>	Indicates that the function was unable to delete the shortcut.

## DeleteShortcut Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the DeleteShortcut and DeleteShortcutFolder
 * functions.
 *
 * This script deletes the Notepad Example 3 shortcut from the
 * Example folder 3 folder. DeleteShortcutFolder is then
 * called again to delete this folder.
 *
 * Note: In order for this script to run properly, you must set
 *       the preprocessor constants to a valid folder and shortcut
 *       on the target system. To easily create this example
 *       folder and shortcut, run the CreateShortcut example 3.
 *
\*-----*/

#define FOLDER "C:\\Windows\\Example folder 3"
#define SHORTCUT "Notepad Example 3"

function OnFirstUIAfter()
begin

```

```

// Display the folder.
ShowProgramFolder (FOLDER, SW_SHOW);
Delay (3);

// Delete the Notepad Example 3 shortcut.
if (DeleteShortcut (FOLDER, SHORTCUT) < 0) then
    MessageBox ("DeleteShortcut failed.", SEVERE);
endif;

// Delete the Example folder 3 shortcut.
if (DeleteShortcutFolder (FOLDER) < 0) then
    MessageBox ("DeleteShortcutFolder failed.", SEVERE);
endif;

end;

```

## DeleteShortcutFolder

The **DeleteShortcutFolder** function removes a shortcut folder (that is, a subfolder of the Start menu's Programs folder) and its contents, including all shortcuts and all of the shortcut folder's subfolders and their contents.

**DeleteShortcutFolder** cannot remove the Programs folder.




---

**Tip** • In a Basic MSI or InstallScript MSI project, this functionality is possibly better achieved by using the native Windows Installer RemoveFolders action. Or, if you are simply uninstalling, the Windows Installer engine natively handles the removal of all files and folders that were created during the installation. For more information on the RemoveFolders action, see the Windows Installer Help.

### Syntax

```
DeleteShortcutFolder ( szFolderName );
```

## Parameters

**Table 143** • DeleteShortcutFolder Parameters

Parameter	Description
<b>szFolderName</b>	Specify the name of the folder to remove.

## Return Values

**Table 144** • DeleteShortcutFolder Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully removed the specified folder.
<b>&lt; 0</b>	Indicates that the function was unable to remove the folder.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

## DeleteShortcutFolder Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the DeleteShortcut and DeleteShortcutFolder
 * functions.
 *
 * This script deletes the Notepad Example 3 shortcut from the
 * Example folder 3 folder. DeleteShortcutFolder is then
 * called again to delete this folder.
 *
 * Note: In order for this script to run properly, you must set
 *       the preprocessor constants to a valid folder and shortcut
 *       on the target system. To easily create this example
 *       folder and shortcut, run the CreateShortcut example 3.
 *
\*-----*/

#define FOLDER "C:\\Windows\\Example folder 3"
#define SHORTCUT "Notepad Example 3"

function OnFirstUIAfter()
begin

```

```

// Display the folder.
ShowProgramFolder (FOLDER, SW_SHOW);
Delay (3);

// Delete the Notepad Example 3 shortcut.
if (DeleteShortcut (FOLDER, SHORTCUT) < 0) then
    MessageBox ("DeleteShortcut failed.", SEVERE);
endif;

// Delete the Example folder 3 shortcut.
if (DeleteShortcutFolder (FOLDER) < 0) then
    MessageBox ("DeleteShortcutFolder failed.", SEVERE);
endif;

end;

```

## DeleteWCharArray

### Description

The **DeleteWCharArray** function deletes the array of pointers to which pCharArray points.

### Syntax

```
DeleteWCharArray ( pCharArray );
```

### Parameters

**Table 145** • DeleteWCharArray Parameters

Parameter	Description
<b>pCharArray</b>	Specifies a pointer to an array of pointers to Unicode character strings. Typically, this pointer is returned by a previous call to <a href="#">GetWCharArrayFromISStringArray</a> .

### Return Values

**Table 146** • DeleteWCharArray Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	This function always returns ISERR_SUCCESS.

## DialogSetFont

The **DialogSetFont** function sets the font for InstallScript dialogs that are displayed at run time. This function affects built-in InstallScript dialogs and custom InstallScript dialogs (that is, dialogs that are defined through [EzDefineDialog](#) or [DefineDialog](#)). This function does not affect dialogs that are displayed by calling the Windows API

function `MessageBox`, which are displayed in the font specified by the user for message boxes (specified using the Windows Control panel). This function does not affect the text in the title bar of any dialog; the font of dialog title bars is set by Windows.

## Syntax

```
DialogSetFont (szFontName, nFontSize, nReserved);
```

## Parameters

**Table 147** • DialogSetFont Parameters

Parameter	Description
<b>szFontName</b>	Specifies the font to be used—for example, “Times New Roman”.
<b>nFontSize</b>	Specifies the font size—for example, 10.
<b>nReserved</b>	Pass 0 (zero) in this parameter. No other value is allowed.

## Return Values

`DialogSetFont` always returns 0 (zero). If the function cannot change the font, dialog text is displayed in the system font.

## Additional Information

When changing the font of InstallShield dialogs, use a font that you know is available on any system that the setup is running on. Also, be sure to test the setup on a variety of screen resolutions to ensure that the font works correctly.

# DialogSetInfo



**Edition** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI (in event-driven InstallScript—not in InstallScript custom actions)*

The **DialogSetInfo** function changes the following display elements in run-time dialogs:

- The image to be displayed
- The style of the check boxes used to obtain end-user selections
- The precision of the values that indicate available hard drive space

You must call **DialogSetInfo** each time that you want to change a particular aspect of a single dialog. Changes that are made by a call to **DialogSetInfo** remain in effect for the remainder of the installation or until they are changed again by a subsequent call to **DialogSetInfo**.



---

**Note** • If your script calls **DialogSetInfo** before calling any of the *Sd* dialog functions, the call to **DialogSetInfo** must be preceded by a call to [SdInit](#). If it is not, the call to **DialogSetInfo** has no effect.

## Syntax

```
DialogSetInfo ( nInfoType, szInfoString, nParameter );
```



## Parameters

**Table 148 • DialogSetInfo Parameters**

Parameter	Description
<b>nInfoType</b>	<p>Specifies the display feature to be modified. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li> <b>DLG_INFO_USEDECIMAL</b>—By default, the values displayed to indicate feature sizes, available disk space, and required disk space are rounded to the nearest kilobyte or megabyte. Pass this constant when you want these values displayed to the nearest tenth of a kilobyte or megabyte. The following dialogs are affected by this parameter: <b>FeatureDialog</b>, <b>SdFeatureDialog</b>, <b>SdFeatureDialog2</b>, <b>SdFeatureDialogAdv</b>, and <b>SdFeatureMult</b>. </li> <li> <b>DLG_INFO_KUNITS</b>—By default, the values displayed to indicate feature sizes, available disk space, and required disk space are displayed as a measurement in megabytes. Pass this constant to display these measurements in kilobytes. The following dialogs are affected by this parameter: <b>FeatureDialog</b>, <b>SdFeatureDialog</b>, <b>SdFeatureDialog2</b>, <b>SdFeatureDialogAdv</b>, and <b>SdFeatureMult</b>. </li> <li> <b>DLG_INFO_ALTIMAGE</b>—Specifies an alternate bitmap to be displayed in the dialog. If nParameter is set to DLG_INFO_ALTIMAGE_VERIFY_BMP or TRUE, szInfoString should specify the image to be displayed in the dialog. This parameter applies to all dialogs that display the standard installation image on the left side of the dialog. For more information, see the <a href="#">nParameter</a> description. <p>Display effects that have been set with <a href="#">SetDisplayEffect</a> do not apply to alternate images, which are always displayed without any special effects.</p> </li> <li> <b>DLG_INFO_ALTIMAGE_HIDPI</b>—Specifies a high DPI image to be displayed in the dialog. High DPI image types supported include BMP, GIF, JPEG, PNG, and TIFF. For best results, in order to ensure the image is centered and shifted to the right of the header, use images with an aspect ratio of 2.5:1, similar to the images shipped with InstallShield (where the actual logo portion of the image is approximately half of the image width and the other half is transparent), and note the following image sizing recommendations: <ul style="list-style-type: none"> <li>For a 100% scaled image, your image should be 180x75</li> <li>For a 200% scaled image, your image should be 360x150</li> <li>For a 250% scaled image, your image should be 450x187, etc.</li> </ul> <p>If transparency is required, image types that support transparency (such as PNG) should be used and szInfoString should specify the name of the image to be displayed (optionally including the path) in the dialog. This parameter applies to all dialogs that display the standard installation image on the left side of the dialog. When DLG_INFO_ALTIMAGE_HIDPI is passed in nInfoType, the following parameter values are expected: szInfoString and nParameter. For more information, see the <a href="#">szInfoString</a> and <a href="#">nParameter</a> descriptions.</p> <p>Display effects that have been set with <a href="#">SetDisplayEffect</a> do not apply to alternate images, which are always displayed without any special effects.</p> </li> </ul>

**Table 148 •** DialogSetInfo Parameters (cont.)

Parameter	Description
<b>szInfoString</b>	<p>When DLG_INFO_ALTIMAGE is passed in nInfoType, this parameter specifies the file name of an alternate bitmap to display and, optionally, a set of bitmap attributes. If bitmap attributes are included, the string passed in this parameter should be formatted as follows:</p> <pre>"bitmap file name;transparent flag;3-D flag;&lt;unused&gt;;background color"</pre> <ul style="list-style-type: none"> <li>• <b>Bitmap file name</b>—Specifies the name of the bitmap file. If the file name is unqualified (that is, if it does not include a drive designation and path), the installation searches for the bitmap in <a href="#">SUPPORTDIR</a>.</li> <li>• <b>Transparent flag</b>—Indicates whether to display the bitmap transparently. When this flag is 1 (true), all portions of the bitmap that are magenta (RGB Value: 255,0,255) will be displayed transparently. The default for this parameter is 0 (non-transparent).</li> <li>• <b>3-D flag</b>—Indicates whether to add a 3-D border around the edges of the static field that contains the bitmap. The default for this parameter is 0 (no 3D border).</li> <li>• <b>&lt;unused&gt;</b>—This portion of the formatted string is ignored, but it must be included. That is, the formatted string must contain four semicolons, with two semicolons between the 3-D flag and the background color.</li> <li>• <b>Background color</b>—Indicates the color to use for the background of the static text field. Note that this color will be visible only if the bitmap is smaller than the static text field in which it appears or if the transparent flag is set to 1 and the bitmap has transparent areas. The background color must be expressed as an RGB value, that is, as three numeric values separated by commas.</li> </ul> <p>The following example will display the bitmap from the file <b>MyBitmap.bmp</b>, which is located in the SUPPORTDIR folder. The bitmap is placed on a black background and has a three-dimensional border. Any parts of the bitmap that are magenta are displayed in the background color of black.</p> <pre>SUPPORTDIR ^ "MyBitmap.bmp" + ";1;1;;0,0,0"</pre> <p>Note that the standard bitmap measures 57 x 53. An alternate bitmap should be about this size as well. If the bitmap is larger than this size, it will be centered vertically within the title area, and the right side of the bitmap will be aligned with the right side of the dialog. (In the Welcome, SdWelcome, and SdFinish dialogs, the right side of the bitmap will be aligned with the right side of the larger image within which the bitmap appears.) The left side of the bitmap will extend as far to the left of the dialog as necessary. Any part of the bitmap that extends below the title area of the dialog will be clipped. If the bitmap is smaller than 57 x 53 it will be displayed correctly, but it will not be resized or extended.</p> <p>When DLG_INFO_ALTIMAGE_HIDPI is passed in nInfoType, this parameter specifies the file name of a high DPI image to display, optionally including the path. If no path to the file is specific, the file is assumed to be in SUPPORTDIR. If this file does not exist, DialogSetInfo returns ISERR_FILE_NOT_FOUND.</p> <p>This parameter is ignored when the default image is being restored or when nInfoType is not DLG_INFO_ALTIMAGE or DLG_INFO_ALTIMAGE_HIDPI.</p>

**Table 148 •** DialogSetInfo Parameters (cont.)

Parameter	Description
<b>nParameter</b>	<p>Operates in conjunction with nInfoType to specify dialog features.</p> <p>When nInfoType is DLG_INFO_ALTIMAGE, pass one of the following predefined constants to specify which bitmap to display:</p> <ul style="list-style-type: none"> <li>• <b>DLG_INFO_ALTIMAGE_VERIFY_BMP</b>—Specifies that the bitmap that is indicated by szInfoString should be used in subsequent dialogs. Before this bitmap is used, the installation checks for the existence of the bitmap.</li> <li>• <b>DLG_INFO_ALTIMAGE_REVERT_IMAGE (-1)</b>—Specifies that subsequent dialogs should display the default bitmap. The installation does not check for the existence of a bitmap.</li> <li>• <b>TRUE</b>—Specifies that the bitmap that is indicated by szInfoString should be used in subsequent dialogs. The installation does not check for the existence of the bitmap.</li> </ul> <p>When nInfoType is DLG_INFO_ALTIMAGE_HIDPI, nParameter specifies the DPI scaling percentage. For example, pass 200 for a 200% image scale, 150 for a 150% scale, etc. The minimum supported scaling value is 25. If 0 is passed for this value, no image is displayed. If DLG_INFO_ALTIMAGE_REVERT_IMAGE is passed, the previous image used is displayed.</p> <p>When nInfoType is either DLG_INFO_KUNITS or DLG_INFO_USEDECIMAL, pass one of the following predefined constants to specify how sizes should be displayed:</p> <ul style="list-style-type: none"> <li>• <b>TRUE</b>—Specifies that sizes should be displayed as indicated by nInfoType.</li> <li>• <b>FALSE</b>—Specifies that sizes should be displayed in the default style.</li> </ul>

## Return Values

**Table 149 •** DialogSetInfo Return Values

Return Value	Description
<b>ISERR_SUCCESS (0)</b>	The function successfully set the specified style. If DLG_INFO_ALTIMAGE_VERIFY_BMP was passed in nParameter, this return value also indicates that the bitmap was found.
<b>&lt; ISERR_SUCCESS (&lt; 0)</b>	An unspecified error occurred when the function attempted to set the dialog information.
<b>ISERR_FILE_NOT_FOUND (0x80070004)</b>	The image that was indicated by szInfoString was not found.

## Additional Information

To preview the effects of a call to **DialogSetInfo** in an InstallScript installation, run the Dialog Sampler (which is available from the Tools menu's InstallScript submenu), change the attributes of the dialogs (by clicking the Attributes button), then examine the changes in dialogs such as **SdFeatureMult**.

## DialogSetInfo Example



**Edition** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI (in event-driven InstallScript—not in InstallScript custom actions)*

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the DialogSetInfo function.
 *
 * This script calls AskText twice. On the first call the
 * AskText dialog displays the default bitmap. Then
 * DialogSetInfo is called to specify an alternate bitmap;
 * that bitmap is then displayed on the second call to AskText.
 *
 * Note: Before running this script, set the defined constant
 *       FULL_BMP_PATH so that it references a bitmap file
 *       included in the Support Files/Billboards view.
 *
 \*-----*/

#define FULL_BMP_PATH SUPPORTDIR ^ "MyBitmap.bmp"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

function OnBegin()
    STRING  szText, szMsg, szBmpPath;
    STRING  svReturnText;
    NUMBER  nReturn;
begin

start:

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Display the AskText dialog with its default bitmap.
    szText = "Default Bitmap.";
    szMsg  = "The bitmap on the left is the default.";
    nReturn = AskText (szMsg, szText, svReturnText);

    // Enable the Back button.
    Enable (BACKBUTTON);

    szBmpPath = FULL_BMP_PATH;

    // Set the alternate bitmap for the AskText dialog.
    DialogSetInfo (DLG_INFO_ALTIMAGE, szBmpPath, TRUE);

    // Set the text for display in the AskText dialog.
    szText = "Alternate Bitmap.";
    szMsg  = "The bitmap on the left is a custom bitmap. This alternate " +
```

```

        "bitmap was displayed using the DLG_INFO_ALTBITMAP option in " +
        "DialogSetInfo.";

// Display the AskText dialog with the alternate
// bitmap.
nReturn = AskText (szMsg, szText, svReturnText);

// Handle Back button.
if (nReturn = BACK) then
    // Restore the default bitmap setting.
    DialogSetInfo (DLG_INFO_ALTIMAGE, "", DLG_INFO_ALTIMAGE_REVERT_IMAGE);
    goto start;
endif;

end;

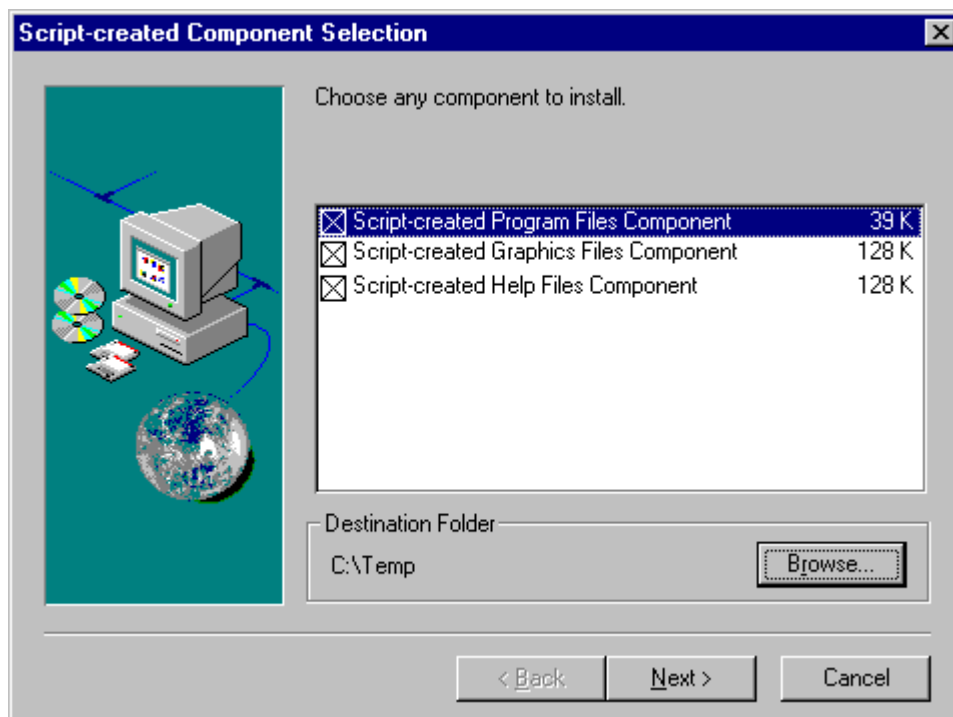
```

## Dialog Styles

Four different dialog styles are available:

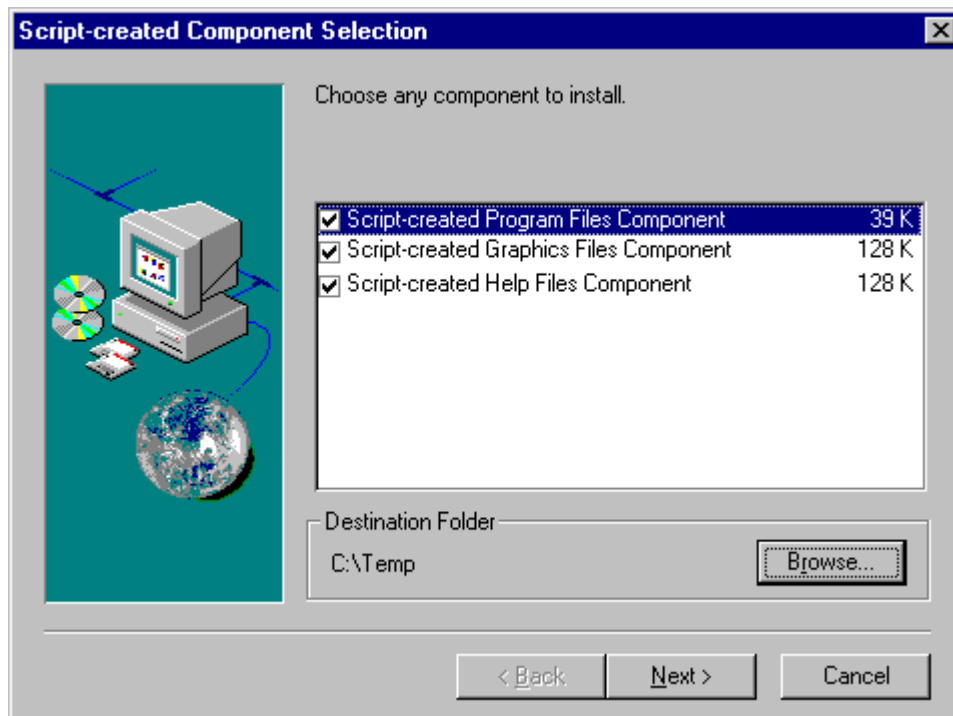
- [CHECKBOX Dialog Style](#)
- [CHECKBOX95 Dialog Style](#)
- [CHECKMARK Dialog Style](#)
- [CHECKLINE Dialog Style](#)

### CHECKBOX Dialog Style

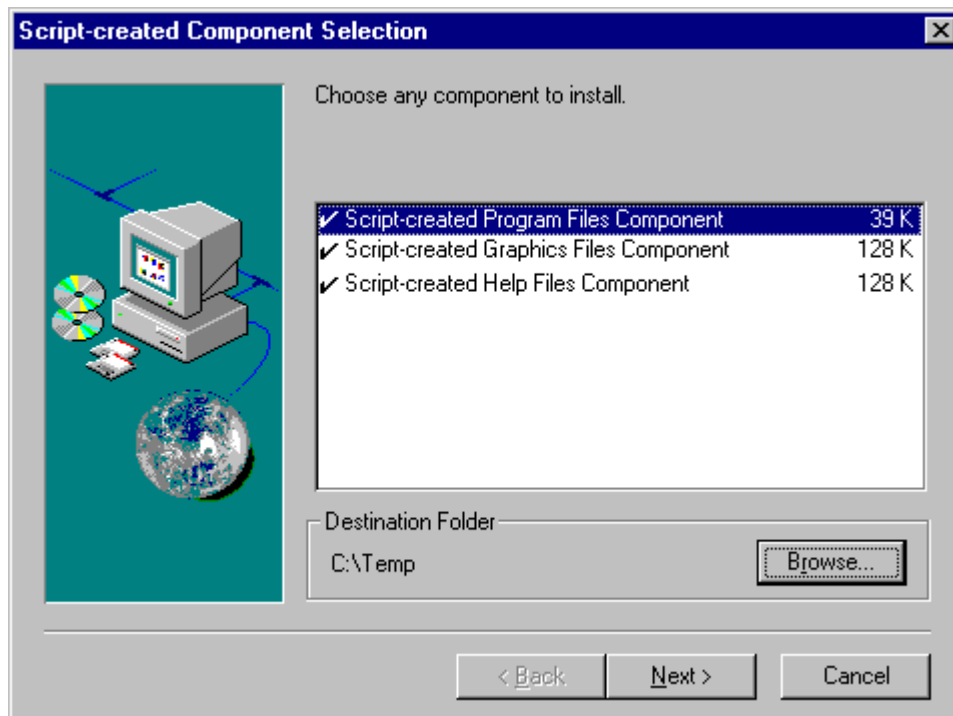


**Figure 1:** CHECKBOX Dialog Style

## CHECKBOX95 Dialog Style

**Figure 2:** CHECKBOX95 Dialog Style

## CHECKMARK Dialog Style

**Figure 3:** CHECKMARK Dialog Style

## CHECKLINE Dialog Style

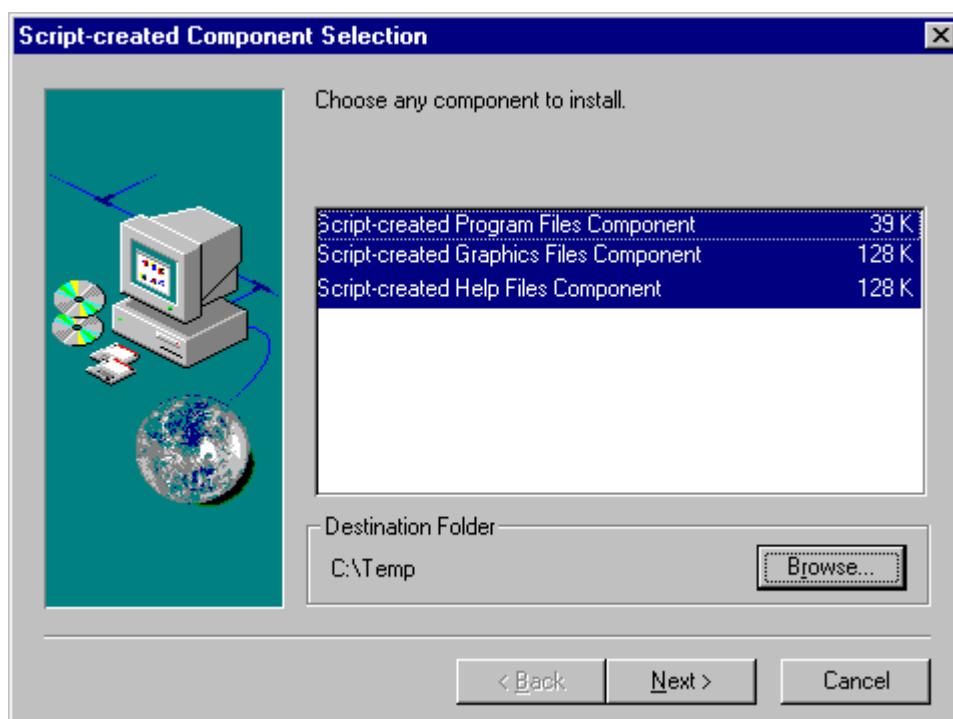


Figure 4: CHECKLINE Dialog Style

## DIFxDriverPackageGetPath



**Project** • This function applies to InstallScript projects only. This function is not required in InstallScript MSI projects since DIFx can be called by the Windows Installer in those projects.

The **DIFxDriverPackageGetPath** function retrieves the fully qualified path to a driver store .inf file.



**Note** • This function calls the DIFxAPI function *DriverPackageGetPath*. See the DIFxAPI documentation for additional details regarding this function and its parameters and return values.

## Syntax

```
DIFxDriverPackageGetPath( byval string szDriverPackageInfPath, byref string svDestInfPath, byval number nISFlags );
```



## Parameters

**Table 150** • DIFxDriverPackageGetPath Parameters

Parameter	Description
<b>szDriverPackageInfPath</b>	Specifies the fully qualified path to a driver package .inf file for which to retrieve the corresponding driver store .inf file.
<b>svDestInfPath</b>	Specifies the fully qualified path of the driver store .inf file that corresponds to the driver package .inf file supplied by DriverPackageInfPath.
<b>nISFlags</b>	Specifies InstallScript-specific flags. The following flags are available: <ul style="list-style-type: none"> <li>• <b>0</b>—Default behavior</li> <li>• <b>ISDIFX_OPTION_DONT_RESOLVE_TEXTSUBS</b>—By default, the function resolves any text substitutions found in szDriverPackageInfPath. However, if this flag is specified, text substitutions are not resolved.</li> </ul>

## Return Values

**Table 151** • DIFxDriverPackageGetPath Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	The function was successful.
<b>&lt; ISERR_SUCCESS</b>	<p>The function failed.</p> <p>If the return value from <b>DriverPackageGetPath</b> is a Win32 error (a positive return value), ISERR_WIN_BASE is added to the error to ensure that it is &lt; ISERR_SUCCESS.</p> <p>You can use the following code to get the original Win32 error, if desired:</p> <pre>if( nResult &amp; ISERR_WIN_BASE ) then nResult = nResult - ISERR_WIN_BASE; endif;</pre> <p>For a list of specific errors, see DIFxAPI Errors (InstallScript Projects).</p>

## Additional Information

For more information on DIFx and DIFxAPI, see the [MSDN Library](#).

# DIFxDriverPackageInstall



**Project** • This function applies to InstallScript projects only. This function is not required in InstallScript MSI projects since DIFx can be called by the Windows Installer in those projects.

The **DIFxDriverPackageInstall** function installs a driver package in the driver store and then installs the driver in the system. If all of the files associated with the driver are in a single component or you can guarantee that all appropriate driver files will be installed when this component's Installed event is called, then the recommended place for calling this function is in the component containing the DIFx driver's Installed event. Otherwise, call this function in the installation's OnMoved event.

If uninstall logging is enabled when this function is called, the driver installed by this function is logged for uninstallation and is automatically removed by the OnUninstallingDIFxDriverFile event when the application is removed.



---

**Note** • This function calls the DIFxAPI function *DriverPackageInstall*. See the DIFxAPI documentation for additional details regarding this function and its parameters and return values.

## Syntax

```
DIFxDriverPackageInstall( byval string szDriverPackageInfPath, byval number nFlags, byval number  
    nISFlags );
```

## Parameters

**Table 152 • DIFxDriverPackageInstall Parameters**

Parameter	Description
<b>szDriverPackageInfPath</b>	String that supplies the fully qualified path to the driver package .inf file of the driver package to install.
<b>nFlags</b>	<p>One or more flags that control the installation operation. In most cases you can specify 0 because the installer will automatically add the appropriate flags.</p> <p>The following additional flags can be specified manually:</p> <div data-bbox="589 623 630 667" data-label="Image"> </div> <p><b>Note •</b> These flags are defined by and passed directly to the <i>Flags</i> parameter of the DIFxAPI function <i>DriverPackageInstall</i>. See the DIFxAPI documentation for additional information regarding these flags.</p> <ul style="list-style-type: none"> <li>• <b>DRIVER_PACKAGE_REPAIR</b>—Re-installs the specified driver package in the driver store even if the driver package is already installed. This flag is specified automatically in repair mode.</li> <li>• <b>DRIVER_PACKAGE_ONLY_IF_DEVICE_PRESENT</b>—(Applies only to PnP function drivers) Preinstalls and installs the driver only if the driver package is a better match to a device in the device tree.</li> <li>• <b>DRIVER_PACKAGE_FORCE</b>—(Applies only to PnP function drivers) By default, installs a new driver for device only if the new driver is a better match for the device than the driver currently installed for the device. If you specify this flag, the function preinstalls and installs the specified driver package even if the driver package currently installed for a device is a better match for the device than the specified driver package.</li> <li>• <b>DRIVER_PACKAGE_SILENT</b>—Suppresses the display of user dialogs. If a user interaction is required to continue the installation, for example, in response to a driver signing dialog, the installation operation fails without displaying a user message. The function returns an error code that indicates the cause of the failure.</li> <li>• <b>DRIVER_PACKAGE_LEGACY_MODE</b>—Preinstalls and installs unsigned driver packages and driver packages that cannot be completely preinstalled because there are files that cannot be found.</li> </ul>

**Table 152** • DIFxDriverPackageInstall Parameters (cont.)

Parameter	Description
<b>nISFlags</b>	<p>Specifies InstallScript-specific flags. The following flags are available:</p> <ul style="list-style-type: none"> <li>● <b>0</b>—Default behavior</li> <li>● <b>ISDIFX_OPTION_DONT_ASSOCIATE</b>—By default, associates the installed driver with the application being installed. If this flag is specified the driver is not associated with any application.</li> <li>● <b>ISDIFX_OPTION_NO_REPAIR</b>—By default, the function automatically adds the DRIVER_PACKAGE_REPAIR flag when calling the <b>DriverPackagePreinstall</b> function if REINSTALLMODE is TRUE. If this flag is specified, DRIVER_PACKAGE_REPAIR is not added automatically. However, it is passed if it is specified in nFlags.</li> <li>● <b>ISDIFX_OPTION_LOG_IN_DRIVER_PACKAGE_PATH</b>—By default, the function logs the installed driver for uninstallation in the installed driver cache. If this flag is specified, the driver is logged in the package path instead.</li> <li>● <b>ISDIFX_OPTION_DONT_RESOLVE_TEXTSUBS</b>—By default, the function resolves any text substitutions found in szDriverPackageInfPath. However, if this flag is specified, text substitutions are not resolved.</li> </ul>

## Return Values

**Table 153** • DIFxDriverPackageInstall Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	The function was successful.
<b>&lt; ISERR_SUCCESS</b>	<p>The function failed.</p> <p>If the return value from <b>DriverPackageInstall</b> is a Win32 error (a positive return value), ISERR_WIN_BASE is added to the error to ensure that it is &lt; ISERR_SUCCESS.</p> <p>You can use the following code to get the original Win32 error, if desired:</p> <pre>if( nResult &amp; ISERR_WIN_BASE ) then nResult = nResult - ISERR_WIN_BASE; endif;</pre> <p>For a list of specific errors, see DIFxAPI Errors (InstallScript Projects).</p>

## Additional Information

- For more information on DIFx and DIFxAPI, see the [MSDN Library](#).
- When a driver is installed by **DIFxDriverPackageInstall** or uninstalled by **DIFxDriverPackageUninstall**, the driver is associated with the application being installed by the installation by default. This association can be disabled by specifying ISDIFX\_OPTION\_DONT\_ASSOCIATE. These functions use the following script variables to determine the application to associate:
  - **ISDIFXAPPID**

- [IFX\\_PRODUCT\\_DISPLAY\\_NAME](#)
- [IFX\\_PRODUCT\\_NAME](#)
- [IFX\\_COMPANY\\_NAME](#)

## DIFxDriverPackagePreinstall



**Project** • This function applies to InstallScript projects only. This function is not required in InstallScript MSI projects since DIFx can be called by the Windows Installer in those projects.

The **DIFxDriverPackagePreinstall** function preinstalls a driver package for a Plug and Play (PnP) function driver in the driver store and installs the .inf file for the driver package in the system .inf file directory. If all of the files associated with the driver are in a single component or you can guarantee that all appropriate driver files will be installed when this component's Installed event is called, then the recommended place for calling this function is in the component containing the DIFx driver's Installed event. Otherwise, call this function in the installation's OnMoved event.

If uninstall logging is enabled when this function is called, the driver pre-installed by this function is logged for uninstallation and is automatically removed by the OnUninstallingDIFxDriverFile event when the application is removed.



**Note** • This function calls the DIFxAPI function *DriverPackagePreinstall*. See the DIFxAPI documentation for additional details regarding this function and its parameters and return values.

### Syntax

```
DIFxDriverPackagePreinstall( byval string szDriverPackageInfPath, byval number nFlags, byval number nISFlags );
```

## Parameters

**Table 154 • DIFxDriverPackagePreinstall Parameters**

Parameter	Description
<b>szDriverPackageInfPath</b>	String that supplies the fully qualified path to the driver package .inf file of the driver package to pre-install.
<b>nFlags</b>	<p>One or more flags that control the installation operation. In most cases, specify 0 to have the installer automatically add the appropriate flags.</p> <p>The following additional flags can be specified manually:</p> <div data-bbox="625 617 665 661" data-label="Image"> </div> <p><b>Note •</b> These flags are defined by and passed directly to the <i>Flags</i> parameter of the DIFxAPI function <i>DriverPackagePreinstall</i>. See the DIFxAPI documentation for additional information regarding these flags.</p> <ul style="list-style-type: none"> <li>• <b>DRIVER_PACKAGE_REPAIR</b>—Re-installs the specified driver package in the driver store even if the driver package is already installed. This flag is specified automatically in repair mode.</li> <li>• <b>DRIVER_PACKAGE_ONLY_IF_DEVICE_PRESENT</b>—(Applies only to PnP function drivers) Preinstalls and installs the driver only if the driver package is a better match to a device in the device tree.</li> <li>• <b>DRIVER_PACKAGE_FORCE</b>—(Applies only to PnP function drivers) By default, installs a new driver for device only if the new driver is a better match for the device than the driver currently installed for the device. If you specify this flag, the function preinstalls and installs the specified driver package even if the driver package currently installed for a device is a better match for the device than the specified driver package.</li> <li>• <b>DRIVER_PACKAGE_SILENT</b>—Suppresses the display of user dialogs. If a user interaction is required to continue the installation, for example, in response to a driver signing dialog, the installation operation fails without displaying a user message. The function returns an error code that indicates the cause of the failure.</li> <li>• <b>DRIVER_PACKAGE_LEGACY_MODE</b>—Preinstalls and installs unsigned driver packages and driver packages that cannot be completely preinstalled because there are files that cannot be found.</li> </ul>

**Table 154** • DIFxDriverPackagePreinstall Parameters (cont.)

Parameter	Description
<b>nISFlags</b>	<p>Specifies InstallScript-specific flags. The following flags are available:</p> <ul style="list-style-type: none"> <li>• <b>0</b>—Default behavior</li> <li>• <b>ISDIFX_OPTION_NO_REPAIR</b>—By default, the function automatically adds the DRIVER_PACKAGE_REPAIR flag when calling <b>DriverPackagePreinstall</b> if REINSTALLMODE is TRUE. If this flag is specified, DRIVER_PACKAGE_REPAIR is not added automatically. However, it is passed if it is specified in nFlags.</li> <li>• <b>ISDIFX_OPTION_LOG_IN_DRIVER_PACKAGE_PATH</b>—By default, the function logs the installed driver for uninstallation in the installed driver cache. If this flag is specified, the driver is logged in the package page instead.</li> <li>• <b>ISDIFX_OPTION_DONT_RESOLVE_TEXTSUBS</b>—By default, the function resolves any text substitutions found in <b>szDriverPackageInfPath</b>. If this flag is specified, text substitutions are not resolved.</li> </ul>

## Return Values

**Table 155** • DIFxDriverPackagePreinstall Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	The function was successful.
<b>&lt; ISERR_SUCCESS</b>	<p>The function failed.</p> <p>If the return value from <b>DriverPackagePreinstall</b> is a Win32 error (a positive return value), ISERR_WIN_BASE is added to the error to ensure that it is &lt; ISERR_SUCCESS.</p> <p>You can use the following code to get the original Win32 error, if desired:</p> <pre>if( nResult &amp; ISERR_WIN_BASE ) then nResult = nResult - ISERR_WIN_BASE; endif;</pre> <p>For a list of specific errors, see DIFxAPI Errors (InstallScript Projects).</p>

## Additional Information

For more information on DIFx and DIFxAPI, see the [MSDN Library](#).

# DIFxDriverPackageUninstall



**Project** • This function applies to InstallScript projects only. This function is not required in InstallScript MSI projects since DIFx can be called by the Windows Installer in those projects.

The **DIFxDriverPackageUninstall** function uninstalls the specified driver package from the system and removes the driver package from the driver store.

It is not necessary to call this function explicitly for drivers installed with **DIFxDriverPackageInstall** or **DIFxDriverPackagePreinstall** while uninstall logging is enabled as these drivers are automatically removed by the OnUninstallingDIFxDriverFile event.



---

**Note** • This function calls the DIFxAPI function *DriverPackageUninstall*. See the DIFxAPI documentation for additional details regarding this function and its parameters and return values.

## Syntax

```
DIFxDriverPackageUninstall( byval string szDriverPackageInfPath, byval number nFlags, byval number  
    nISFlags );
```



## Parameters

**Table 156 • DIFxDriverPackageUninstall Parameters**

Parameter	Description
<b>szDriverPackageInfPath</b>	String that supplies the fully qualified path to the driver package .inf file of the driver package to pre-install.
<b>nFlags</b>	<p>One or more flags that control the installation operation. In most cases, specify 0 to have the installer automatically add the appropriate flags.</p> <p>The following additional flags can be specified manually:</p> <div data-bbox="625 623 664 667" data-label="Image"> </div> <p><b>Note</b> • These flags are defined by and passed directly to the <i>Flags</i> parameter of the DIFxAPI function <i>DriverPackageUninstall</i>. See the DIFxAPI documentation for additional information regarding these flags.</p> <ul style="list-style-type: none"> <li>• <b>DRIVER_PACKAGE_FORCE</b>—(Applies only to PnP function drivers) By default, installs a new driver for device only if the new driver is a better match for the device than the driver currently installed for the device. If you specify this flag, the function preinstalls and installs the specified driver package even if the driver package currently installed for a device is a better match for the device than the specified driver package.</li> <li>• <b>DRIVER_PACKAGE_SILENT</b>—Suppresses the display of user dialogs. If a user interaction is required to continue the installation, for example, in response to a driver signing dialog, the installation operation fails without displaying a user message. The function returns an error code that indicates the cause of the failure.</li> <li>• <b>DRIVER_PACKAGE_DELETE_FILES</b>—Removes the binary files from a system that were copied to the system when the driver package was installed. The function removes a binary file from the system only if the binary file is identical to the corresponding binary file in the driver store.</li> </ul> <div data-bbox="625 1373 670 1415" data-label="Image"> </div> <p><b>Caution</b> • Use this flag with caution. Only use this flag if you can verify that a binary file in the system is not required by any other driver package or application.</p>
<b>nISFlags</b>	<p>Specifies InstallScript-specific flags. The following flags are available:</p> <ul style="list-style-type: none"> <li>• <b>0</b>—Default behavior</li> <li>• <b>ISDIFX_OPTION_DONT_ASSOCIATE</b>—By default, the function associates the installed driver with the application being installed. If this flag is specified, the driver is not associated with any application.</li> <li>• <b>ISDIFX_OPTION_DONT_RESOLVE_TEXTSUBS</b>—By default, the function resolves any text substitutions found in <i>szDriverPackageInfPath</i>. If this flag is specified, text substitutions are not resolved.</li> </ul>

## Return Values

**Table 157** • DIFxDriverPackageUninstall Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	The function was successful.
<b>&lt; ISERR_SUCCESS</b>	<p>The function failed.</p> <p>If the return value from <b>DriverPackageUninstall</b> is a Win32 error (a positive return value), ISERR_WIN_BASE is added to the error to ensure that it is &lt; ISERR_SUCCESS.</p> <p>You can use the following code to get the original Win32 error, if desired:</p> <pre>if( nResult &amp; ISERR_WIN_BASE ) then nResult = nResult - ISERR_WIN_BASE; endif;</pre> <p>For a list of specific errors, see DIFxAPI Errors (InstallScript Projects).</p>

## Additional Information

- For more information on DIFx and DIFxAPI, see the [MSDN Library](#).
- When a driver is installed by **DIFxDriverPackageInstall** or uninstalled by **DIFxDriverPackageUninstall**, the driver is associated with the application being installed by the installation by default. This association can be disabled by specifying ISDIFX\_OPTION\_DONT\_ASSOCIATE. These functions use the following script variables to determine the application to associate:
  - [ISDIFXAPPID](#)
  - [IFX\\_PRODUCT\\_DISPLAY\\_NAME](#)
  - [IFX\\_PRODUCT\\_NAME](#)
  - [IFX\\_COMPANY\\_NAME](#)

## Disable

The **Disable** function deactivates the user interface object or setup feature specified by the parameter nConstant.


If your script calls the **Disable** function to disable the Next or Back button, that button is disabled in all dialogs displayed after that function call. To re-enable the Next or Back button, you must call [Enable](#) with the corresponding constant.

## Syntax



```
Disable ( nConstant );
```

## Parameters

**Table 158 •** Disable Parameters

Parameter	Description
<b>nConstant</b>	<p>Specifies the user interface object or operational feature to disable.</p> <p>Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>BACKBUTTON</b>—Disables (grays out) the Back button that is displayed in some built-in dialogs. The Back button is enabled by default.</li> <li>• <b>BACKGROUND</b>—Disables and hides the installation's main background window. Note that this parameter has no effect when the installation is in full-screen mode.</li> <li>• <b>BILLBOARD</b>—Suppresses the display of billboards during an installation.</li> <li>• <b>CANCELBUTTON</b>—Disables (grays out) the Cancel button that is displayed in some built-in dialogs.</li> <li>• <b>DIALOGCACHE</b>—Disables the dialog cache mechanism. For more information about dialog caching, see <a href="#">Enable</a>.</li> </ul> <hr/> <p> <b>Note</b> • <i>DIALOGCACHE has no effect on dialogs that do not have a Next or Back button.</i></p> <ul style="list-style-type: none"> <li>• <b>HOURLGLASS</b>—Causes the mouse cursor to change from the "Busy" cursor (an hourglass by default), to a normal cursor (a pointer by default).</li> </ul>

**Table 158 •** Disable Parameters (cont.)

Parameter	Description
<b>nConstant (cont.)</b>	<ul style="list-style-type: none"> <li> <b>LOGGING</b>—Disables the logging of uninstallation information so that no information is recorded in the uninstallation log file. <p>Note that logging is enabled automatically by default. If you need to disable logging, it is recommended that you do so by calling <b>Disable</b> with the <b>LOGGING</b> constant immediately before performing operations that should not be logged for uninstallation. Then you can re-enable logging by calling <b>Enable</b> with the <b>LOGGING</b> constant.</p> <p>For more information about logging, see <i>InstallScript Functions that Are Logged for Uninstallation</i>.</p> </li> </ul> <hr/> <p> <b>Tip</b> • To prevent changes that are made by a component from being undone during uninstallation, select <i>No</i> for the <i>Uninstall</i> setting of a component in an <i>InstallScript</i> project, or select <i>Yes</i> for the <i>Permanent</i> setting of a component in an <i>InstallScript MSI</i> project.</p> <ul style="list-style-type: none"> <li> <b>NEXTBUTTON</b>—Disables (grays out) the Next button that is displayed in some built-in <i>InstallScript</i> dialogs. On most dialogs, the Next button is enabled by default. </li> </ul> <hr/> <p> <b>Note</b> • Calling <b>Disable</b> with the <b>NEXTBUTTON</b> constant has no effect on the <b>SdCustomerInformation</b>, <b>SdCustomerInformationEx</b>, <b>SdRegisterUser</b>, or <b>SdRegisterUserEx</b> dialogs since they enable and disable the Next button internally.</p> <ul style="list-style-type: none"> <li> <b>PCRESTORE</b>—Disables System Restore compatibility, which is enabled by default. </li> <li> <b>REGISTRYFUNCTIONS_USETEXTSUBS</b>—Disables text substitutions in strings that are passed to registry functions; this is enabled by default. Use this option when working with registry function strings that contain opening angle brackets (&lt;) and closing angle brackets (&gt;) but that should not be interpreted as text substitutions. </li> <li> <b>SELFREGISTERBATCH</b>—Disables the “batch method” for registering files copied with <b>XCopyFile</b> and the <b>SELFREGISTER</b> constant. The batch method is enabled by default. <p>When the batch method is disabled, files are registered immediately when copied with <b>XCopyFile</b> and the <b>SELFREGISTER</b> constant. If the batch method is enabled, registration is postponed until data transfer takes place.</p> </li> </ul>

**Table 158 •** Disable Parameters (cont.)

Parameter	Description
<b>nConstant (cont.)</b>	<ul style="list-style-type: none"> <li>• <b>SERVICE_DIFX_32</b>—Disables DIFx support for 32-bit platforms.</li> <li>• <b>SERVICE_DIFX_AMD64</b>—Disables DIFx support for AMD 64-bit platforms.</li> <li>• <b>SERVICE_DIFX_IA64</b>—Disables DIFx support for Itanium 64-bit platforms.</li> <li>• <b>SERVICE_ISFONTREG</b>—Disables global font registration. For details, see <a href="#">Installing Fonts Through InstallScript and InstallScript Object Projects</a>. Global font registration is enabled by default; once it is disabled, it cannot be re-enabled from the script.</li> <li>• <b>STATUS</b>—Disables and hides the progress indicator (status bar).</li> <li>• <b>STATUSBBRD</b>—Disables and hides the progress dialog that includes a billboard.</li> <li>• <b>STATUSDLG</b>—Disables and hides the dialog style progress indicator (status bar).</li> <li>• <b>STATUSEX</b>—Disables the display of the default Setup Status dialog.</li> <li>• <b>STATUSOLD</b>—Disables and hides the old style progress indicator (status bar).</li> <li>• <b>UPDATE_SERVICE_INSTALL</b>—This constant is obsolete.</li> <li>• <b>USE_LOADED_SKIN</b>—Intended for use with custom dialogs that will not work with dialog skins (for example, dialogs that are not the standard size); not recommended for use with built-in dialogs. Disables the use of the skin that you specified in the Specify Skin setting on the Build tab for the release in the Releases view. (If you selected the <b>&lt;Do not use any skin&gt; option</b> in that setting, this constant has no effect.) Disabling of the skin applies only to dialogs that are displayed after you call <code>Disable(USE_LOADED_SKIN)</code>; in your InstallScript code. To disable skin use for a custom dialog, you must call <code>Disable(USE_LOADED_SKIN)</code>; before calling <b>WaitOnDialog</b>. To re-enable skin use, call <code>Enable(USE_LOADED_SKIN)</code>; in your script.</li> </ul>



**Project •** The `USE_LOADED_SKIN` constant is applicable to InstallScript projects.

- **WOW64FSREDIRECTION**—Disables 64-bit Windows file system redirection. You may need to do this before installing files to the [WINSYSDIR64](#) destination. To learn more, see [Targeting 64-Bit Operating Systems with InstallScript Installations](#).

## Return Values

**Table 159** • Disable Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully disabled the user interface object or setup feature specified by the parameter nConstant.
<b>&lt; 0</b>	Indicates that the function was unable to disable the user interface object or setup feature specified by the parameter nConstant.

## Disable Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the Disable and Enable functions.
 *
 * This script displays two dialogs. In the first box, the
 * Back button is disabled. In the second box, the Next button
 * is disabled and the Back button enabled.
 *
 \*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_Disable(HWND);

function ExFn_Disable(hMSI)
begin
start:

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // The following displays a dialog with the Back button disabled.
    SetupType ("", "", "", TYPICAL, 0);

    // Enable the Back button,
    Enable (BACKBUTTON);

    // Next button is disabled.
    Disable (NEXTBUTTON);

    // The following displays a dialog with only the Back button enabled.
    if (SetupType ("", "", "", TYPICAL, 0) = BACK) then
        // If the Back button is pressed, the Next button is enabled.
        Enable (NEXTBUTTON);
        goto start;
    endif;

end;

```

# Do

The **Do** function executes the currently defined EXIT and HELP handlers, giving you greater control over these handlers, which are normally executed only when the user presses the F1 key (HELP) or the Cancel button (EXIT). Using the **Do** function, you can execute EXIT or HELP handlers in response to custom dialog events or to any user input from built-in dialogs. You can also use the **Do** function during script development to test your EXIT and HELP handlers.

The **Do** function can also register queued self-registering files. Files are queued for registration using the “batch method” for installing self-registering files. When you call `Do(SELFREGISTRATIONPROCESS)`, the installation carries out self-registration of all queued files, even if one of them fails. (Note that when you call [FeatureTransferData](#), `Do(SELFREGISTRATIONPROCESS)` is called automatically after the files are installed but before the **FeatureTransferData** call returns. If you use an event-based script, the **FeatureTransferData** function is called by the default code for the **OnMoveData** event handler function.)

## Syntax

```
Do (nOperation);
```

## Parameters

**Table 160 • Do Parameters**

Parameter	Description
<b>nOperation</b>	Specifies the type of operation to perform. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li><b>EXIT</b>—Initiates the Exit operation. If no EXIT handler is defined, the default Exit dialog is displayed.</li> <li><b>HELP</b>—Initiates the Help operation. If no HELP handler is defined, the function takes no action.</li> <li><b>SELFREGISTRATIONPROCESS</b>—Registers all self-registering files that have been queued for registration.</li> </ul>

## Return Values

**Table 161 • Do Return Values**

Return Value	Description
<b>0</b>	The <b>Do</b> function successfully initiated the specified operation.
<b>&lt; 0</b>	The <b>Do</b> function was unable to initiate the specified operation.

## Additional Information

- The **Do** function allows the currently defined HELP and EXIT handlers to execute without the end user pressing F1. The **Do** function also provides more versatility than the goto statement, which can be used to call HELP and EXIT handler labels. The goto statement cannot be used in all circumstances, but the **Do** function can be called virtually anytime. For more information on default and custom HELP and EXIT handlers, see [HandlerEx](#).

- If **Do** fails for any reason, it returns -1. The names of the files that failed to self-register are stored in the InstallScript system variable ERRORFILENAME, and each is separated by a semicolon (;).

## Do Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the Do function.
*
* This script calls Do to test the HELP and EXIT handler.
*
* Note: Before running this script, set the preprocessor
*       constant so that it references a valid
*       help file on the target system.
*
\*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

#define HELPFILE WINDIR^"Help\\Windows.chm"

export prototype ExFn_Do(HWND);

function ExFn_Do(hMSI)
begin
    // Install the exit handler.
    HandlerEx (EXIT, Exit_Handler);

    // Install the help handler.
    HandlerEx (HELP, Help_Handler);

    // Execute loop forever -- or until user aborts.
    while (TRUE)
        if (AskYesNo ("View the help?", NO) = YES) then
            // Execute the help handler.
            Do (HELP);
        endif;

        // Execute the exit handler.
        Do (EXIT);
    endwhile;

    // The exit handler.
    Exit_Handler:

        // Ask for confirmation to abort.
        if (AskYesNo ("Do you really want to exit?", NO) = YES) then
            abort;
        endif;
    end Exit_Handler;
end ExFn_Do;
```



```

    else
        // Continue if not sure.
        return;
    endif;

// The help handler.
Help_Handler:

    // Display the help.
    LaunchApplication (HELPPFILE, "", "", SW_SHOW, INFINITE, LAAW_OPTION_WAIT);
    return;

end;

```

## DoInstall



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **DoInstall** function launches another InstallShield installation that has a valid setup executable file (.exe). The second installation is executed immediately when this function is called. The third parameter, *nOptions*, specifies various options, including whether the installation should wait until the launched application terminates before continuing.



**Note** • **DoInstall** cannot be used directly to launch a .msi file. In addition, because **DoInstall** adds command-line parameters that are specific to InstallShield installations, **DoInstall** should not be used to launch a non-InstallShield installation. To launch a non-InstallShield installation, use the **LaunchApplication** function.

### Syntax


```
DoInstall ( byval string szSetupExe, byval string szCmdLine, byval number nOptions );
```

Parameters

**Table 162** • DoInstall Parameters

Parameter	Description
<b>szSetupExe</b>	<p>Specifies the fully qualified name—including a drive designation and complete path—of the setup executable file to be launched.</p> <p>Alternatively, you can specify the compiled script file of the installation to be launched. For InstallScript MSI installations, you can specify the installation's .msi package. Note that in order to be able to use <b>DoInstall</b>, your installation must include a valid setup executable file named <b>Setup.exe</b>, or an alternative name, as described in <a href="#">Additional Information</a>.</p>

**Table 162 • DoInstall Parameters (cont.)**

Parameter	Description
<b>szCmdLine</b>	<p>Specifies the command line for the launched installation. You can specify any valid startup InstallShield command line for this parameter. Note that <b>DoInstall</b> automatically adds some command-line switches, depending on the nOptions specified.</p> <p></p> <p><b>Note</b> • Unlike with previous versions of InstallShield, if you are passing command-line parameters to a child InstallScript MSI installation, you must pass these options through the /z parameter (as when launching the installation directly).</p>
<b>nOptions</b>	<p>Specifies options for launching the installation. You can specify any valid <a href="#">LaunchApplication</a> option, including:</p> <ul style="list-style-type: none"> <li>• LAAW_OPTION_NOWAIT</li> <li>• LAAW_OPTION_WAIT</li> </ul> <p>You can also specify some additional options that apply only to <b>DoInstall</b>:</p> <ul style="list-style-type: none"> <li>• <b>DOINSTALL_OPTION_NOHIDEPROGRESS</b>—Specifies that the entire initialization user interface, including the initial progress dialog and splash screen (if any), should be displayed for the child installation. <p>If this option is not specified, <b>DoInstall</b> automatically hides the initialization user interface of the child installation by using the /hide_progress option.</p> </li> <li>• <b>DOINSTALL_OPTION_NOHIDESPLASH</b>—Specifies that the splash screen (if any) should be displayed for the child installation that is being launched.</li> <li>• <b>DOINSTALL_OPTION_NOLANGSWITCH</b>—Specifies that the /1 switch will not be specified. <p>By default, <b>DoInstall</b> adds the /1 switch to szCmdLine so that the launched installation runs in the same language as the launching installation. Note that if the child installation does not support the language in which the parent installation is running (as determined by calling <b>Is(LANGUAGE_SUPPORTED)</b>), the /1 switch is not added.</p> </li> <li>• <b>DOINSTALL_OPTION_NOSETBATCHINSTALL</b>—Specifies that <b>DoInstall</b> should not use the LAAW_OPTION_SET_BATCH_INSTALL option to determine whether the child installation performed actions that require a reboot. Thus, BATCH_INSTALL for the parent installation will not be changed by <b>DoInstall</b>, regardless of the actions of the child installation. <p>By default, DoInstall automatically uses the LAAW_OPTION_SET_BATCH_INSTALL option.</p> </li> </ul> <p>You can combine these constants by using the bitwise OR operator (   ). Note that combining DOINSTALL_OPTION_NOSETBATCHINSTALL with LAAW_OPTION_SET_BATCH_INSTALL may lead to unexpected results.</p>

## Return Values

**Table 163 • DoInstall Return Values**

Return Value	Description
0	If <b>DoInstall</b> was called with LAAW_OPTION_WAIT as the third argument, the installation launched by <b>DoInstall</b> terminated successfully. 0 is always returned when <b>DoInstall</b> is called with LAAW_OPTION_NOWAIT as the third argument. Control resumes in the calling installation with the statement that follows the <b>DoInstall</b> function.
<b>ISERR_SETUP_CANCELED (0x80042000)</b>	The InstallScript installation launched by <b>DoInstall</b> with LAAW_OPTION_WAIT as the third argument exited with the abort keyword. This usually indicates that the user canceled the installation.
-3	The InstallScript MSI installation launched by <b>DoInstall</b> with LAAW_OPTION_WAIT as the third argument exited with the abort keyword. This usually indicates that the user canceled the installation.
<b>All other negative values</b>	An unspecified error has occurred.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

## Additional Information

- By default, **DoInstall** automatically attempts to launch the specified setup executable file. If a non-executable file name is specified in szSetupExe, the function attempts to launch **Setup.exe** in the specified folder. If the installation's executable file has a name other than *Setup*, the launched installation's **Setup.ini** file must have the new name of **Setup.exe** in the [Startup] section's LauncherName key.



**Project •** If you specify a value for the **Setup File Name** setting for a release in the Releases view of an InstallScript MSI project and then build the release, InstallShield automatically adds file name value to **Setup.ini**. If you rename the setup launcher executable file in an InstallScript project, you must add the name to the **Setup.ini** file manually.

- When the installation is run from any removable media, such as a CD or a DVD, the **Setup.exe** file on Disk1 may not be available during the entire installation. (If **Setup.exe** becomes unavailable while it is running, the operating system sometimes displays a prompt to request that the end user insert the correct disk, and this may cause the installation to fail.) Therefore, to avoid this problem, the **Setup.exe** file is copied to a Temp folder, and the installation is relaunched from there. The original **Setup.exe** then terminates. However, when this happens, **DoInstall** behaves as if the installation has completed, and it does not wait.

To avoid this issue, you may want to use the /clone\_wait parameter when you are launching the child installation; when this occurs, the launched installation keeps the original launched process running, and the parent installation then waits. Note, however, that this may cause problems if the original CD containing **Setup.exe** is not available throughout the entire installation. This includes multiple-CD installations, where the first CD is not available during some parts of the installation.

The only other way to avoid this problem is to add code that determines the ID of the child processes of the launched process and wait for the child process to complete.

## DoInstall Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the DoInstall function.
*
* This example script runs the MessageBox example script
* using the DoInstall function.
*
* Note: To make this example work correctly you must also do
* the following:
*
* 1. Create a second setup project. This setup will be
* launched by the DoInstall function. This project
* should include an up-to-date built release.
*
* 2. Create a new folder named 'Second' in the disk1
* folder of this setup.
*
* 3. Copy the disk# folder(s) from the second setup
* into the newly created 'Second' folder.
*
* The second setup should then be launched successfully.
*
\*-----*/

#define SECOND_INSTALL_PATH SRCDIR ^ "Second\\Disk1"
#define SECOND_INSTALL_FILENAME "Setup.exe"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_DoInstall(HWND);

function ExFn_DoInstall(hMSI)
    NUMBER nReturn;
    STRING szTemp;
begin

    MessageBox ("About to launch the second setup.", INFORMATION);

    // Launch the second setup.
    nReturn = DoInstall (SECOND_INSTALL_PATH ^ SECOND_INSTALL_FILENAME, "",
        LAAW_OPTION_WAIT);

    if ( nReturn = 0) then
        // Report successful second setup launch.
```

```

        MessageBox("The second setup was launched successfully.", INFORMATION);
    else
        // Report failure to launch second setup.
        sprintfBox(SEVERE, "", "DoInstall failed with a return code of %d.", nReturn);
    endif;

end;

```

## DotNetCoCreateObject



**Project** • The following project types support the **DotNetCoCreateObject** function:

- *InstallScript*
- *InstallScript MSI*
- *Basic MSI with InstallScript custom actions*

The **DotNetCoCreateObject** function calls functions in .NET assemblies without the assembly being registered for COM interoperability. This function lets you specify the .NET application domain in which the .NET assemblies should be loaded and run. Each object created with this function is associated with a single class in a single .NET assembly. If you want to access multiple classes in the same assembly, you must create a separate object for each class.




**Important** • The assembly does not have to be registered as a COM component, but the assembly must be built to be compatible with COM interoperability. Assemblies built with Visual Studio .NET 2003 and earlier are automatically built with this compatibility. However, projects created with Visual Studio 2005 must manually specify [assembly: ComVisible(true)] in the appropriate file.

### Syntax

```
DotNetCoCreateObject ( byval string szAssemblyPathFile, byval string szAssemblyAndClassName, byval
    string szAppDomain );
```

## Parameters

**Table 164 • DotNetCoCreateObject Parameters**

Parameter	Description
<b>szAssemblyPathFile</b>	Specifies the full path and file name of the .NET assembly that contains the appropriate class.
<b>szAssemblyAndClassName</b>	Specifies the assembly and class name.   <b>Note</b> • This value is the same as the <i>szProgId</i> parameter if the assembly is registered for COM interoperability using the <b>CoCreateObject</b> function.
<b>szAppDomain</b>	Specifies the .NET application domain to load and run the assembly in. If the specified application domain does not exist in the installation process, it is created; otherwise, the existing application domain is used. This is true if you call <b>DotNetCoCreateObject</b> multiple times with the same <i>szAppDomain</i> .  If you specify a null string ("") for this parameter or call <b>CoCreateObjectDotNet</b> instead of <b>DotNetCoCreateObject</b> , the assembly is loaded into the default application domain after the installation completes; thus, the .NET assembly file is locked until the installation has finished.  For more information on .NET application domains, see the .NET Framework Developer's Guide in the MSDN Library.

## Return Values

The reference can be assigned to a variable of type OBJECT by using the set keyword.

## Additional Information

- The **DotNetCoCreateObject** function is similar to the **CoCreateObjectDotNet** function. The only difference is that with **DotNetCoCreateObject**, you can specify the .NET application domain that should be loaded; the assembly is then run in this domain.

For **CoCreateObjectDotNet**, the .NET assembly is loaded into the default application domain after the installation completes; thus, the .NET assembly file is locked until the installation has finished.

- Any object variable can be released by setting the object variable to the value of NOTHING or reassigning the object with the **CoCreateObject**, **CoCreateObjectDotNet**, **CoGetObject**, or **DotNetCoCreateObject** functions. However, this does not automatically unload the library referenced by the object. You must call the Windows API **CoFreeLibrary** manually to free the library. Otherwise, the library remains loaded until the installation finishes. For more information, see Extending Your Installation with COM Objects.

- This function throws an exception if the object cannot be created. This can occur if the .NET Framework is not installed on the system, or for other reasons. To handle this exception, surround calls to this function with try...catch blocks. For more information, see [Exception Handling](#).

## DotNetUnloadAppDomain



**Project** • The following project types support the **DotNetUnloadAppDomain** function:

- *InstallScript*
- *InstallScript MSI*
- *Basic MSI with InstallScript custom actions*

The **DotNetUnloadAppDomain** function unloads the specified .NET application domain and releases any assemblies that are currently loaded into the specified application domain.



**Note** • Once an application domain is unloaded, all .NET objects that were created with **DotNetCoCreateObject** become invalid. Therefore, you should set these objects to NOTHING using the set command before calling **DotNetUnloadAppDomain**.

### Syntax

```
DotNetUnloadAppDomain ( byval string szAppDomain );
```

### Parameters

**Table 165** • DotNetUnloadAppDomain Parameters

Parameter	Description
<b>szAppDomain</b>	Specifies the .NET application domain to be unloaded.  For more information on .NET application domains, see the .NET Framework Developer's Guide in the MSDN Library.

### Return Values

**Table 166** • DotNetUnloadAppDomain Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	The domain was successfully unloaded.
<b>&lt; ISERR_SUCCESS</b>	The domain was not unloaded.





# Built-In Functions (E-G)

For a list of functions by category, see [Built-In Functions by Category](#).

## Enable

The **Enable** function activates the user interface object or setup feature specified by the parameter nConstant.

By default, an installation runs without a background. To enable window mode, you must call **Enable** with the BACKGROUND constant, and then again with DEFWINDOWMODE or FULLWINDOWMODE. These constants are not supported for use in Basic MSI installations.



---


**Note** • If your script calls the **Disable** function to disable the Next or Back button, that button is disabled in all dialogs displayed after that function call. To re-enable the Next or Back button, you must call **Enable** with the corresponding constant.

### Syntax


```
Enable ( nConstant );
```

## Parameters


**Table 1 • Enable Parameters**

Parameter	Description
<b>nConstant</b>	<p>Specifies the user interface object or operational feature you want to enable. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>BACKBUTTON</b>—Enables the Back button that is displayed in some built-in dialogs. The Back button is enabled by default, but it can be disabled by calling the <b>Disable</b> function.</li> <li>• <b>BACKGROUND</b>—Displays the installation's main background window when the installation is in window mode. When installation is in full-screen mode, which is the default mode, this constant has no effect. To enable window mode, you must call <b>Enable</b> with the constant <b>DEFWINDOWMODE</b> or <b>FULLWINDOWMODE</b>.</li> <li>• <b>CANCELBUTTON</b>—Enables the Cancel button that is displayed in some built-in dialogs and the status dialog.</li> <li>• <b>DEFWINDOWMODE</b>—Configures the main background window to be a normal window with a title bar. If the background window is enabled, its appearance changes immediately. If the background window is not enabled, no change to the screen occurs until <b>Enable</b> is called with the constant <b>BACKGROUND</b>.</li> <li>• <b>DIALOGCACHE</b>—Enables the dialog cache mechanism, which eliminates the screen flash that appears between the display of dialogs. This screen flash is most noticeable in the title bar of installations running in window mode. Note that the dialog caching mechanism works only for dialogs that have <b>BACK</b> and <b>NEXT</b> buttons. Dialog caching is enabled by default.</li> </ul>
 <p><b>Note •</b> <i>DIALOGCACHE has no effect on dialogs that do not have a Next or Back button.</i></p>	
	<ul style="list-style-type: none"> <li>• <b>FULLWINDOWMODE</b>—Configures the main background window to be a maximized window with a title bar. If the background window is enabled, its appearance will change immediately. If the background window is not enabled, no change to the screen will occur until <b>Enable</b> is called with the constant <b>BACKGROUND</b>.</li> <li>• <b>HOURGLASS</b>—Causes the mouse cursor to change to the "Busy" cursor, an hourglass by default.</li> <li>• <b>INDVFILESTATUS</b>—Enables the display (on the second line of the progress indicator) of the fully qualified file name of each file as it is transferred when <b>FeatureMoveData</b>, <b>CopyFile</b>, or <b>XCopyFile</b> is called and the progress indicator is enabled.</li> </ul>

**Table 1 • Enable Parameters (cont.)**

Parameter	Description
<b>nConstant (cont.)</b>	<ul style="list-style-type: none"> <li> <b>LOGGING</b>—Enables the logging of uninstallation information. When logging is enabled, results of operations that the InstallScript engine logs for uninstallation are recorded in the uninstallation log file and are reversed during uninstallation.   Note that logging is enabled automatically by default. If you need to disable logging, it is recommended that you do so by calling <b>Disable</b> with the LOGGING constant immediately before performing operations that should not be logged for uninstallation. Then you can re-enable logging by calling <b>Enable</b> with the LOGGING constant.   For more information about logging, see InstallScript Functions that Are Logged for Uninstallation. </li> <li> <b>NEXTBUTTON</b>—Enables the Next button that is displayed by some built-in dialogs. On most dialogs, the Next button is enabled by default. </li> </ul> <hr/>  <p><b>Note •</b> Calling <b>Enable</b> with the <b>BACKBUTTON</b> constant has no effect on the <b>SdCustomerInformation</b>, <b>SdCustomerInformationEx</b>, <b>SdRegisterUser</b>, or <b>SdRegisterUserEx</b> dialogs since they enable and disable the Next button internally.</p> <ul style="list-style-type: none"> <li> <b>PCRESTORE</b>—Enables System Restore compatibility, which is enabled by default. </li> <li> <b>REGISTRYFUNCTIONS_USETEXTSUBS</b>—Enables text substitutions in strings that are passed to registry functions; this is enabled by default. </li> <li> <b>SELFREGISTERBATCH</b>—Specifies whether to use the “batch method” for registering files copied with <b>XCopyFile</b> and the SELFREGISTER constant. The batch method is enabled by default.   When the batch method is disabled, files are registered immediately when copied with <b>XCopyFile</b> and the SELFREGISTER constant. If the batch method is enabled, registration is postponed until data transfer takes place. </li> <li> <b>SERVICE_DIFX_32</b>—Enables DIFx support for 32-bit platforms. </li> <li> <b>SERVICE_DIFX_AMD64</b>—Enables DIFx support for AMD 64-bit platforms. </li> <li> <b>SERVICE_DIFX_IA64</b>—Enables DIFx support for Itanium 64-bit platforms. </li> <li> <b>STATUS</b>—Enables the display of the progress indicator (status bar). </li> <li> <b>STATUSBBRD</b>—Enables the display of the progress dialog that includes a billboard. </li> <li> <b>STATUSDLG</b>—Enables the display of the dialog style progress indicator (status bar). </li> <li> <b>STATUSEX</b>—Enables the display of the standard Setup Status dialog. </li> <li> <b>STATUSOLD</b>—Enables the display of the old style progress indicator (status bar), which does not have a Cancel button. </li> </ul>

**Table 1 • Enable Parameters (cont.)**

Parameter	Description
<b>nConstant (cont.)</b>	<ul style="list-style-type: none"> <li>• <b>UPDATE_SERVICE_INSTALL</b>—This constant is obsolete.</li> <li>• <b>USE_LOADED_SKIN</b>—Intended for use with custom dialogs that will not work with dialog skins (for example, dialogs that are not the standard size); not recommended for use with built-in dialogs. Enables the use of the skin that you specified in the Specify Skin setting on the Build tab for the release in the Releases view. (If you selected the <b>&lt;Do not use any skin&gt;</b> option in that setting, this constant has no effect.) If you specify a skin, it is displayed by default; <code>Disable(USE_LOADED_SKIN);</code> and <code>Enable(USE_LOADED_SKIN);</code> are called internally by functions that display dialogs that cannot be displayed with a skin. Enabling of the skin applies only to dialogs that are displayed after you call <code>Enable(USE_LOADED_SKIN);</code> in your script. To enable skin use (after disabling it) for a custom dialog, you must call <code>Enable(USE_LOADED_SKIN);</code> before calling <b>WaitOnDialog</b>.</li> </ul>
 <p><b>Project •</b> The <code>USE_LOADED_SKIN</code> constant is applicable to InstallScript projects.</p>	
	<ul style="list-style-type: none"> <li>• <b>WOW64FSREDIRECTION</b>—Enables 64-bit Windows file system redirection. You may need to do this after installing files to the <code>WINSYSDIR64</code> destination. To learn more, see Targeting 64-Bit Operating Systems with InstallScript Installations.</li> </ul>

## Return Values

**Table 2 • Enable Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully enabled the user interface object or setup feature specified by the parameter nConstant.
<b>&lt; 0</b>	Indicates that the function was unable to enable the user interface object or setup feature specified by the parameter nConstant.

## Enable Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the Disable and Enable functions.
 *
 * This script displays two dialogs. In the first box, the
 * Back button is disabled. In the second box, the Next button
 * is disabled and the Back button enabled.
 *
 \*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
```

```

#include "Ifx.h"

export prototype ExFn_Enable(HWND);

function ExFn_Enable(hMSI)
begin

start:

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // The following displays a dialog with the Back button disabled.
    SetupType ("", "", "", TYPICAL, 0);

    // Enable the Back button,
    Enable (BACKBUTTON);

    // Next button is disabled.
    Disable (NEXTBUTTON);

    // The following displays a dialog with only the Back button enabled.
    if (SetupType ("", "", "", TYPICAL, 0) = BACK) then
        // If the Back button is pressed, the Next button is enabled.
        Enable (NEXTBUTTON);
        goto start;
    endif;

end;

```

## EndCurrentDialog



**Project** • This information applies to InstallScript projects.

The **EndCurrentDialog** function closes the currently displayed dialog by calling [EndDialog](#). It removes the dialog and initiates the dialog closing process.

### Syntax

```
EndCurrentDialog ( );
```

### Parameters

None

## Return Values

**Table 3 • EndCurrentDialog Return Values**

Return Value	Description
>= <b>ISERR_SUCCESS</b>	EndCurrentDialog successfully closed the dialog.
< <b>ISERR_SUCCESS</b>	No dialog is currently displayed.

## EndDialog

The **EndDialog** function closes a custom dialog. It removes the dialog and initiates the dialog closing process. Use EndDialog when any of the following conditions exist:

- The Next button or its equivalent has been processed.
- The Cancel button or its equivalent has been processed.
- The Close system menu option has been selected. This action sends the DLG\_CLOSE message.
- Any other situation in which the user ends the dialog operation.

After calling EndDialog to end a custom dialog, call the [ReleaseDialog](#) function to free the memory associated with the custom dialog.



**Note •** You can call [WaitOnDialog](#) to redisplay a custom dialog that was closed by a call to EndDialog provided that you have not called ReleaseDialog to remove the dialog from memory. If you call WaitOnDialog to open a dialog that has been opened and closed previously in your script, you must call [CmdGetHwndDlg](#) again to get the new handle. The old handle is no longer valid.

## Syntax

```
EndDialog ( szDialogName );
```

## Parameters

**Table 4 • EndDialog Parameters**

Parameter	Description
<b>szDialogName</b>	Specifies the name of the dialog to close.

## Return Values

**Table 5 • EndDialog Return Values**

Return Value	Description
<b>0</b>	EndDialog successfully closed the dialog.
<b>&lt; 0</b>	EndDialog was unable to close the dialog.

## EndDialog Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the DefineDialog, EndDialog, and ReleaseDialog
* functions.
*
* This script opens a simple custom dialog that displays
* a bitmap. The dialog can be closed with any of three
* buttons: Back, Next, or Cancel.
*
* The "custom" dialog used in this script is actually the
* InstallShield Sd dialog that is displayed by the built-in
* function SdBitmap. Because this dialog is stored in
* the file _isres.dll, which is already compressed in
* the installation, it can be used in a script as a custom
* dialog.
*
* In order to use this dialog as a custom dialog, the
* script first defines it by calling DefineDialog. It then
* displays the dialog by calling WaitOnDialog. When an event
* ends dialog processing, EndDialog is called to close the
* dialog. Then the dialog is released from memory by
* a call to ReleaseDialog.
*
\*-----*/

// Dialog and control IDs.
#define RES_DIALOG_ID    12027    // ID of dialog itself
```

```

#define RES_PBUT_NEXT      1    // ID of Next button
#define RES_PBUT_CANCEL    9    // ID of Cancel button
#define RES_PBUT_BACK      12   // ID of Back button

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_EndDialog(HWND);

function ExFn_EndDialog(hMSI)
    STRING  szDialogName, szDLLName, szDialog;
    NUMBER  nDialog, nResult, nCmdValue;
    BOOL     bDone;
    HWND     hInstance, hwndParent, hwndDlg;
begin
    // Define the name of a dialog to pass as first
    // parameter to DefineDialog.
    szDialogName = "ExampleDialog";

    // DefineDialog's second parameter will be 0 because the
    // .dll file is in _isres.dll.
    hInstance = 0;

    // DefineDialog's third parameter will be null; installation
    // will search for the dialog in _isuser.dll and _isres.dll.
    szDLLName = "";

    // DefineDialog's fifth parameter will be null because the
    // dialog is identified by its ID in the fourth parameter.
    szDialog = "";

    // This value is reserved and must be 0.
    hwndParent = 0;

    // Define the dialog. The installation's main window will own the
    // dialog (indicated by HWND_INSTALL in parameter 7).
    nResult = DefineDialog (szDialogName, hInstance, szDLLName,
                           RES_DIALOG_ID, szDialog, hwndParent,
                           HWND_INSTALL, DLG_MSG_STANDARD|DLG_CENTERED);

    // Check for an error.
    if (nResult < 0) then
        MessageBox ("An error occurred while defining the dialog.", SEVERE);
        bDone = TRUE;
        abort;
    endif;

    // Initialize the indicator used to control the while loop.
    bDone = FALSE;

    // Loop until done.
    repeat

        // Display the dialog and return the next dialog event.
        nCmdValue = WaitOnDialog(szDialogName);

        // Respond to the event.
        switch (nCmdValue)

```



```

case DLG_CLOSE:
    // The user clicked the window's Close button.
    Do (EXIT);
case DLG_ERR:
    MessageBox ("Unable to display dialog. Setup canceled.", SEVERE);
    abort;
case DLG_INIT:
    // Initialize the back, next, and cancel button enable/disable states
    // for this dialog and replace %P, %VS, %VI with
    // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
    // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
    hwndDlg = CmdGetHwndDlg(szDialogName);
    SdGeneralInit(szDialogName, hwndDlg, 0, "");
case RES_PBUT_CANCEL:
    // The user clicked the Cancel button.
    Do (EXIT);
case RES_PBUT_NEXT:
    bDone = TRUE;
case RES_PBUT_BACK:
    bDone = TRUE;
endswitch;

until bDone;

// Close the dialog.
EndDialog (szDialogName);

// Free the dialog from memory.
ReleaseDialog (szDialogName);

end;

```

## EnterDisk



**Project** • This information applies to the following project types:

- Basic MSI
- InstallScript
- InstallScript MSI

The **EnterDisk** function displays a message box that prompts the end user to insert the next disk. The system variable [SRCDIR](#) contains the default path, which is displayed on the message box. The end user can modify the default path by typing a new path and clicking OK.

**EnterDisk** recognizes the correct disk by searching the disk for the tag file specified by `szTagFile`. If the disk does not contain the tag file, the function calls the **EnterDiskError** function to display an error message box that prompts the end user to enter the correct disk. At build time, InstallShield does not automatically generate tag files in the disk image folders. To use tag files, add them to the disk image folders after the folders are built.



**Note** • You cannot use the [PlaceWindow](#) function in conjunction with the **EnterDisk** function. By default, the message box appears in the center of the desktop, unless the background window mode is enabled. If the installation is in window mode, the message box opens in the center of the background window.

The default title is *Setup Needs the Next Disk*. To change the title, call [SetDialogTitle](#) before calling **EnterDisk**.

## Syntax

```
EnterDisk (szMsg, szTagFile);
```

## Parameters

**Table 6 • EnterDisk Parameters**

Parameter	Description
<b>szMsg</b>	Specifies the message that prompts the end user to insert the proper disk.
<b>szTagFile</b>	Specifies the name of the tag file. <b>EnterDisk</b> searches for this file on the inserted disk. If the file is not found, the function calls the <b>EnterDiskError</b> function to display a message that asks the user to insert the correct disk. If you pass a null string ("") in this parameter, the function does not search for any file; it assumes that the correct disk is being used.

## Return Values

**Table 7 • EnterDisk Return Values**

Return Value	Description
<b>OK (1)</b>	Indicates that the user clicked the OK button.
<b>&lt; 0</b>	Indicates that an unspecified error has occurred.

## Additional Information

The dialog that is displayed by the **EnterDisk** function cannot be displayed with a skin; it appears the same regardless of whether you have specified a skin.

## EnterDisk Example



**Project •** This information applies to the following project types:

- Basic MSI
- InstallScript
- InstallScript MSI



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
```

```

* InstallShield Example Script
*
* This example demonstrates the EnterDisk function.
*
* EnterDisk is called to prompt the user to insert a disk or
* to specify a path. EnterDisk then searches for the tag file
* at that location.
*
\*-----*/

#include "ifx.h"

    export prototype ExFn_EnterDisk(HWND);

function ExFn_EnterDisk(hMSI)
    STRING szMsg, szTagFile;
begin

    // Set up parameters for call to EnterDisk.
    szMsg      = "Please insert disk 2";
    szTagFile = "ISExampl.txt";

    // Display the EnterDisk dialog.
    EnterDisk (szMsg, szTagFile);

end;

```

## EnterDiskError



**Project** • This information applies to the following project types:

- Basic MSI
- InstallScript
- InstallScript MSI

The **EnterDiskError** function checks whether a specified path and file exists. The function displays an appropriate error if the file does not exist in the specified path; then it returns success or failure, depending on whether the specified file exists.



**Tip** • **EnterDiskError** can also check for the presence of a specific path, without a particular file.

### Syntax

```
EnterDiskError (byval string szPath, byval string szFile);
```

## Parameters

**Table 8 • EnterDiskError Parameters**

Parameter	Description
<b>szPath</b>	Specify the path to be checked.
<b>szFile</b>	Specify the name of the file to be checked.  If you want <b>EnterDisk</b> to check for just the path, but not a specific file, specify a null string ("").

## Return Values

**Table 9 • EnterDiskError Return Values**

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	The specified path and file exist.
<b>&lt; ISERR_SUCCESS</b>	The specified path and file do not exist.

## Additional Information

Since the **EnterDiskError** function is typically called only internally by the **EnterDisk** function, it does not include any silent mode handling, other than the normal **MessageBox** silent mode handling.

By default, the dialog displays the same error message and the default message box title for the installation, regardless of the error that occurs. You can change this behavior by calling the **SetErrorTitle** and **SetErrorMsg** functions with `nErrorId` set as follows:

- **ERR\_BOX\_DISKID**—Customize the title or message displayed when the specified disk does not exist.
- **ERR\_BOX\_BADPATH**—Customize the title or message displayed when the specified path does not exist.
- **ERR\_BOX\_BADTAGFILE**—Customize the title or message displayed when the specified file does not exist. (If a null string ("") is specified for `szFile`, no error check occurs, and no message is displayed.)

# EnterLoginInfo



**Project •** This information applies to the following project types.

- *InstallScript*
- *InstallScript MSI*

The **EnterLoginInfo** function displays a dialog that enables the end user to specify a user name and password. Note that the dialog does not validate or use the specified information. In addition, the dialog does not perform any error checking for the specified information.

The **EnterLoginInfo** dialog is typically used to let the end user specify a network user name and password. The **SdLogonUserInformation** dialog is similar to the **EnterLoginInfo** dialog, but the **SdLogonUserInformation** dialog provides additional options. If your installation needs to obtain information to log into a SQL Server, **SQLServerLogin** offers additional functionality.

## Syntax

```
EnterLoginInfo (byval string szMsg, byref string svUser, byref string svPassword);
```

## Parameters

**Table 10 • EnterLoginInfo Parameters**

Parameter	Description
<b>szMsg</b>	Specifies the static text to be displayed in the dialog. To display the default text, pass a null string ("" ) in this parameter.
<b>svUser</b>	Specifies the default value that is initially displayed in the User Name box when the function is called, and specifies the string that the end user specified when the function returns.  The dialog does not allow more than 255 characters to be entered in the User Name box.
<b>svPassword</b>	Specifies the default value that is initially displayed in the Password box when the function is called, and specifies the string that the end user specified when the function returns.  The dialog does not allow more than 255 characters to be entered in the Password box.

## Return Values

**Table 11 • EnterLoginInfo Return Values**

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	The function was successful.
<b>&lt; ISERR_SUCCESS</b>	The function was not successful.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

# EnterPassword



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **EnterPassword** function displays a dialog that queries the end user for a password; the characters that the end user types in the edit box are displayed as asterisks (\*).



**Note** • To check the returned password, you can call [FeatureValidate](#), as is done in the default code for the *OnCheckMediaPassword* event handler function.

## Syntax

```
EnterPassword ( szMsg, szDefault, svResult );
```

## Parameters

**Table 12** • EnterPassword Parameters

Parameter	Description
<b>szMsg</b>	Specifies a message to display in the dialog. This text is considered a static control. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>szDefault</b>	Specifies the text that is initially displayed in the edit box.
<b>svResult</b>	Returns the text that the end user enters in the edit box.

## Return Values

**Table 13** • EnterPassword Return Values

Return Value	Description
<b>NEXT</b>	Indicates that the user clicked the Next button.
<b>BACK</b>	Indicates that the user clicked the Back button.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the dialog could not be displayed.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

# ExistsDir

The **ExistsDir** function checks for the existence of a specified directory on the target system or the Internet.

## Syntax

```
ExistsDir ( szPath );
```

## Parameters

**Table 14** • ExistsDir Parameters

Parameters	Description
<b>szPath</b>	Specifies the fully qualified path of the directory to find on the target system, or the valid Uniform Resource Locator (URL) of the directory to find on the Internet. To check the validity of a URL, call <code>Is(VAILD_PATH, szURL)</code> .

## Return Values

**Table 15** • ExistsDir Return Values

Return Value	Description
<b>EXISTS (0)</b>	Indicates that the specified directory exists.
<b>NOTEXISTS (-1)</b>	Indicates that the specified directory does not exist.
<b>All other negative values</b>	Indicates that the function was unable to determine whether the specified directory exists. You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

## ExistsDir Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the ExistsDir function.
 *
 * AskPath is called to get a directory name from the user.
 * Then, ExistsDir is called to determine whether the directory
 * exists.
 *
 \*-----*/
```

```

#define TITLE_TEXT "ExistsDir Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_ExistsDir(HWND);

function ExFn_ExistsDir(hMSI)
    STRING svPath;
begin

    // Get the path to be created.
    AskPath ("Please enter a path:", "", svPath);

    // Check for the existence of the directory.
    if (ExistsDir (svPath) = EXISTS) then
        sprintfBox (INFORMATION, TITLE_TEXT, "%s already exists.", svPath);
    else
        sprintfBox (INFORMATION, TITLE_TEXT, "%s does not exist", svPath);
    endif;

end;

```

## ExistsDisk

The **ExistsDisk** function checks for the existence of a specified disk drive on the target system.

### Syntax

```
ExistsDisk ( szDisk );
```



## Parameters

**Table 16** • ExistsDisk Parameters

Parameter	Description
<b>szDisk</b>	Specifies the letter of the disk drive.

## Return Values

**Table 17** • ExistsDisk Return Values

Return Value	Description
<b>EXISTS (0)</b>	Indicates that the specified drive exists on the target system.
<b>NOTEXISTS (-1)</b>	Indicates that the specified drive does not exist on the target system.

## ExistsDisk Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ExistsDisk function.
*
* AskText is called to get a disk drive name from the user.
* Then, ExistsDir is called to determine whether the drive
* exists.
*
\*-----*/

#define TITLE_TEXT "ExistsDisk Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ExistsDisk(HWND);

function ExFn_ExistsDisk(hMSI)
    STRING svDrive;
begin
    if (AskText ("Enter the letter of a disk drive.", "C", svDrive) = NEXT) then
        // Check for the existence of the specified drive.
        if (ExistsDisk (svDrive) = EXISTS) then
            sprintfBox (INFORMATION, TITLE_TEXT, "Drive %s exists.", svDrive);
        else
            sprintfBox (INFORMATION, TITLE_TEXT, "Drive %s does not exist",
```

```
svDrive);
    endif;
endif;

end;
```

## EzBatchAddPath

The **EzBatchAddPath** function modifies the default batch file by adding a path either to the search path in a PATH command or to the value assigned to an environment variable. Unless it is changed by a call to [BatchSetFileName](#), the default batch file is the Autoexec.bat file that was executed by the system during the boot sequence. To determine the fully qualified name of the default batch file, call [BatchGetFileName](#). To change the name of the batch file to be used by EzBatchAddPath, call [BatchSetFileName](#).




---

**Caution** • *EzBatchAddPath does not make a backup copy of the file it modifies.*

*EzBatchAddPath can fail if the default batch file is hidden or read-only.*




---


**Note** • *Do not mix Ez batch file functions and advanced batch file functions. After calling [BatchFileLoad](#) to load a batch file in memory, you cannot call any of the Ez batch file functions until you call [BatchFileSave](#) to save the file.*

### Syntax

```
EzBatchAddPath ( szKey, szPath, szRefDir, nPosition );
```

## Parameters

**Table 18 • EzBatchAddPath Parameters**

Parameter	Description
<b>szKey</b>	Specifies the name of the environment variable to modify. For example, to modify the PATH statement, specify "PATH" here. If the specified environment variable is not found in the default file batch file, a complete SET statement is created for that environment variable and inserted into the file.
	 <p><b>Caution</b> • The <i>EzBatchAddPath</i> function does not support long file names. Call <a href="#">LongPathToShortPath</a> to convert the long path to its short path equivalent before passing it to <i>EzBatchAddPath</i>. For an explanation of long file names, see <i>Long File Names</i>.</p>
<b>szPath</b>	Specifies the path to add to the current value of the environment variable. A delimiting semicolon is inserted to separate it from other paths in the search path.
<b>szRefDir</b>	Specifies the reference key (a path) relative to which you are adding the new path specified by szPath. If this is a null string (""), the directory is added to the beginning or end of the search path, depending on the value of nPosition. If the path specified by szRefDir is not found in the search path, the value of szKey is added to the end.
<b>nPosition</b>	<p>Specifies where in the search path to add the new path. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>BEFORE</b>—The new path is inserted before the path specified by szRefDir. If szRefDir contains a null string (""), the directory is added to the front of the search path.</li> <li>• <b>AFTER</b>—The new path is inserted after the path specified by szRefDir. If szRefDir contains a null string (""), the directory is added to the end of the search path.</li> </ul>

## Return Values

**Table 19 • EzBatchAddPath Return Values**

Return Value	Description
<b>0</b>	EzBatchAddPath successfully added the path to the batch file.
<b>&lt; 0</b>	EzBatchAddPath was unable to add the path to the batch file.

## EzBatchAddPath Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the EzBatchAddPath function.
 *
 * First, BatchSetFileName is called to change the default batch
 * file to EXAMPLE_BATCH. Then EzBatchAddPath is then called to
 * add C:\WINDOWS to the beginning of the PATH statement. A
 * second call to EzBatchAddPath adds C:\UTILS after the WINDOWS
 * keyword in the PATH statement.
 *
 * Note: Before running this script, create a batch file
 *       named ISExempl.bat and store it in the root of
 *       drive C.
 *
\*-----*/

#define EXAMPLE_BATCH "C:\\ISExempl.bat"
#define TITLE "EzBatchAddPath example"
#define MSG "Successful.\n\n%s added to %s statement."

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_EzBatchAddPath(HWND);

function ExFn_EzBatchAddPath(hMSI)
    STRING szKey, szPath, szRefDir;
begin

    // Set the default batch file.
    BatchSetFileName (EXAMPLE_BATCH);

    // Set up parameters for next call to EzBatchAddPath.
    szKey      = "PATH";
    szPath     = "C:\\WINDOWS";
    szRefDir   = "";

    if (EzBatchAddPath (szKey, szPath, szRefDir, BEFORE) < 0) then
        // Report the error.
        MessageBox ("EzBatchAddPath failed.", SEVERE);
    else

        // Report success.
        SprintfBox (INFORMATION, TITLE, MSG, szPath, szKey);

        // Set up parameters for next call to EzBatchAddPath.
        szKey      = "PATH";
        szPath     = "C:\\UTILS";
        szRefDir   = "WINDOWS";

        if (EzBatchAddPath (szKey, szPath, szRefDir, AFTER) < 0) then
            // Report the error.

```

```

        MessageBox ("EzBatchAddPath failed.", SEVERE);
    else
        // Report success.
        sprintfBox (INFORMATION, TITLE, MSG, szPath, szKey);
    endif;

endif;

end;

```

## EzBatchAddString

The **EzBatchAddString** function adds a line of text to the default batch file; unless it is changed by a call to [BatchSetFileName](#), the default batch file is the Autoexec.bat file that was executed by the system during the boot sequence. To determine the fully qualified name of the default batch file, call [BatchGetFileName](#). To change the name of the batch file to be used by EzBatchAddPath, call BatchSetFileName.




---

**Caution** • The EzBatchAddString function can fail if the default batch file is hidden or read-only.

EzBatchAddString does not make a backup copy of the file it modifies.




---



**Note** • Do not mix Ez batch file functions and advanced batch file functions. After calling [BatchFileLoad](#) to load a batch file in memory, you cannot call any of the Ez batch file functions until you call [BatchFileSave](#) to save the file.

### Syntax

```
EzBatchAddString ( szLine, szRefKey, nOptions );
```

## Parameters

**Table 20 • EzBatchAddString Parameters**

Parameter	Description
<b>szLine</b>	<p>Specifies the line of text to add to the file. Unless you specify NOSET in nOptions, this function assumes that szLine is an environment variable; the text "SET" is inserted at the front of the string before szLine is written to the default batch file.</p> <p> <b>Caution</b> • The EzBatchAddString function does not support long file names. If you are using this function to add a line that contains a long path, call <a href="#">LongPathToShortPath</a> to convert the long path to its short path equivalent before adding it to the string to be placed in the batch file. For an explanation of long file names, see <a href="#">Long File Names</a>.</p>
<b>szRefKey</b>	<p>Specifies the reference key relative to which you want to add szLine in the default batch file. EzBatchAddString searches the default batch file for the reference key and places the contents of szLine before or after that line, depending on the value of nOptions.</p> <p> <b>Note</b> • EzBatchAddString looks for the appropriate reference keyword in the parameter szRefKey. For example, the keyword for an environment variable is the name of the environment variable itself.</p>
<b>nOptions</b>	<p>Specifies the option to use. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>BEFORE</b>—szLine is added before the line containing szRefKey. If szRefKey is a null string (""), szLine is added as the first line of the file.</li> <li>• <b>AFTER</b>—szLine is added after the line containing szRefKey. If szRefKey is a null string (""), szLine is added as the last line of the file.</li> <li>• <b>REPLACE</b>—szLine replaces an existing line in the file. If multiple lines with same key exist, EzBatchAddString replaces only the last line that contains the key.</li> <li>• <b>NOSET</b>—Specifies that the text "SET" should not be inserted at the front of the string in szLine.</li> <li>• <b>COMMAND</b>—Indicates that the reference key you are searching for is a DOS command or program name (not an environment variable).</li> </ul> <p>The NOSET and COMMAND constants can be combined with each other or the BEFORE, AFTER, or REPLACE constants using the logical OR operator. For example, when the reference key you are searching for is a DOS command or program name (not an environment variable), use the OR operator to combine the constant COMMAND with the constant AFTER, as shown below:</p> <pre>EzBatchAddString (szLine, szRefKey, AFTER   COMMAND);</pre>

## Return Values

**Table 21** • EzBatchAddString Return Values

Return Value	Description
0	EzBatchAddString successfully added the text string to the specified batch file.
< 0	EzBatchAddString was unable to add the text string.

## EzBatchAddString Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the EzBatchAddString function.
*
* This script performs four operations on a batch file:
*
* -- First, it adds a comment line at the top of the file.
*
* -- Next, it adds the line "SET TEMP=C:\EXAPP3", placing it
*    it after the PATH statement.
*
* -- Then, it adds the command "C:\EXAPP3\EXAPP.EXE", placing
*    it after the "SET TEMP=C:\EXAPP3" statement.
*
* -- Finally, it replaces the existing PATH statement with a
*    new PATH statement.
*
* Note: Before running this script, create a batch file
*       named ISExempl.bat and store it in the root of
*       drive C. For best effect, include a PATH statement
*       in that file.
*
\*-----*/

#define BATCH_FILE "C:\\ISExempl.bat"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_EzBatchAddString(HWND);

function ExFn_EzBatchAddString(hMSI)
    STRING szLine, szRefKey;
    NUMBER nOptions;
begin

    // Set the default batch file.

```

```

BatchSetFileName (BATCH_FILE);

// Set up parameters for next call to EzBatchAddString.
szLine    = "rem This is the first line of the file.";
szRefKey   = "";
nOptions  = BEFORE|NOSET;

// Add a comment line to the top of the batch file.
if (EzBatchAddString (szLine, szRefKey, nOptions) < 0) then
    MessageBox ("First call to EzBatchAddString failed.", SEVERE);
else
    MessageBox ("First call to EzBatchAddString successful.", INFORMATION);
endif;

// Set up parameters for next call to EzBatchAddString.
szLine    = "TEMP=C:\\EXAPP3";
szRefKey   = "PATH";
nOptions  = AFTER;

// Add a line after the PATH statement.
if (EzBatchAddString (szLine, szRefKey, nOptions) < 0) then
    MessageBox ("Second call to EzBatchAddString failed.", SEVERE);
else
    MessageBox ("Second call to EzBatchAddString successful.", INFORMATION);
endif;

// Set up parameters for next call to EzBatchAddString.
szLine    = "C:\\EXAPP3\\EXAPP.EXE";
szRefKey   = "TEMP";
nOptions  = AFTER|NOSET;

// Add a command line after the SET TEMP statement.
if (EzBatchAddString (szLine, szRefKey, nOptions) < 0) then
    MessageBox ("Third call to EzBatchAddString failed.", SEVERE);
else
    MessageBox ("Third call to EzBatchAddString successful.", INFORMATION);
endif;

// Set up parameters for next call to EzBatchAddString.
szLine    = "PATH=C:\\;C:\\Windows";
szRefKey   = "PATH";
nOptions  = AFTER|NOSET|REPLACE|COMMAND;

// Replace the PATH statement.
if (EzBatchAddString (szLine, szRefKey, nOptions) < 0) then
    MessageBox ("Fourth call to EzBatchAddString failed.", SEVERE);
else
    MessageBox ("Fourth call to EzBatchAddString successful.", INFORMATION);
endif;

endprogram

```



# EzBatchReplace

The **EzBatchReplace** function replaces an existing line of text in the default batch file; unless it is changed by a call to [BatchSetFileName](#), the default batch file is the Autoexec.bat file that was executed by the system during the boot sequence. To determine the fully qualified name of the default batch file, call [BatchGetFileName](#). To change the name of the batch file to be used by EzBatchAddPath, call BatchSetFileName.

Some common keys in a batch file are PATH, COMSPEC, TEMP, Smartdrv.exe, Win.com, and Share.exe.



---

**Caution** • *The EzBatchReplace function may fail if the default batch file is hidden or read-only.*

*EzBatchReplace does not make a backup copy of the file it modifies.*



---


**Note** • *Do not mix Ez batch file functions and advanced batch file functions. After calling [BatchFileLoad](#) to load a batch file in memory, you cannot call any of the Ez batch file functions until you call [BatchFileSave](#) to save the file.*

## Syntax

```
EzBatchReplace ( szNewString );
```

## Parameters

**Table 22 • EzBatchReplace Parameters**

Parameter	Description
<b>szNewString</b>	Specifies the new string to insert in place of an existing line in the file. EzBatchReplace parses szNewString and determines the key of the string. It then searches the default batch file for a line that contains the same key. The function replaces the last line found with the same key.
 <p><b>Caution •</b> The EzBatchReplace function does not support long file names. If you are using this function to replace a line that contains a long path, call <a href="#">LongPathToShortPath</a> to convert the long path to its short path equivalent before replacing it in the string in the batch file. For an explanation of long file names, see <a href="#">Long File Names</a>.</p>	

## Return Values

**Table 23 • EzBatchReplace Return Values**

Return Value	Description
<b>0</b>	EzBatchReplace successfully replaced the line of text.
<b>&lt; 0</b>	EzBatchReplace was unable to replace the line of text.

## EzBatchReplace Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the EzBatchReplace function.
*
* EzBatchReplace is called to replace lines in a batch file.
*
* This script replaces three lines in the specified batch file.
* First, it replaces the SET COMSPEC command. Then it replaces
* the PATH statement. Finally, it replaces the command line
* that references SMARTDRV.EXE.
*
* Note: Before running this script, create a batch file
*       named ISExempl.bat and store it in the root of
*       drive C. For best effect, that file should include
*       the following lines:
```

```

*
*      SET COMSPEC=C:\COMMAND.COM
*      PATH C:\
*      SMARTDRV.EXE
*
\*-----*/

#define EXAMPLE_BAT "C:\\ISExempl.bat"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_EzBatchReplace(HWND);

function ExFn_EzBatchReplace(hMSI)
begin

    // Set the default batch file.
    BatchSetFileName (EXAMPLE_BAT);

    // Replace the SET COMSPEC statement.
    if (EzBatchReplace ("SET COMSPEC=C:\\DOS\\COMMAND.COM") < 0) then
        MessageBox ("First call to EzBatchReplace failed.", SEVERE);
    else
        MessageBox ("First call to EzBatchReplace succeeded.", INFORMATION);
    endif;

    // Replace the PATH command.
    if (EzBatchReplace ("PATH C:\\DOS;C:\\MYAPP") < 0) then
        MessageBox ("Second call to EzBatchReplace failed.", SEVERE);
    else
        MessageBox ("Second call to EzBatchReplace succeeded.", INFORMATION);
    endif;

    // Replace the SMARTDRIVE statement.
    if (EzBatchReplace ("SMARTDRV.EXE /P 1024 /C 512") < 0) then
        MessageBox ("Third call to EzBatchReplace failed.", SEVERE);
    else
        MessageBox ("Third call to EzBatchReplace succeeded.", INFORMATION);
    endif;

end;

```

## EzConfigAddDriver

The **EzConfigAddDriver** function adds a device driver statement to the default system configuration file. You can specify the position of the driver statement relative to another driver statement. For example, an application may require loading a device driver before or after the Windows Himem.sys driver.

Unless changed by a call to [ConfigSetFileName](#), the default system configuration file is the Config.sys file that was executed by the system during the boot sequence. To make another file the default system configuration file, call [ConfigSetFileName](#). To determine the fully qualified name of the default system configuration file, call [ConfigGetFileName](#).



**Note** • Do not mix the Ez configuration file functions and the advanced configuration file functions. After calling [ConfigFileLoad](#), you cannot call any of the Ez configuration file functions until you call [ConfigFileSave](#) to save the file.

## Syntax

```
EzConfigAddDriver ( szDriver, szRefKey, nPosition );
```

## Parameters

**Table 24** • EzConfigAddDriver Parameters

Parameter	Description
<b>szDriver</b>	Specifies the fully qualified name of the driver to add to the file. If the driver already exists in one or more places in the system configuration file, this function replaces only the last occurrence of the line containing the driver.
<b>szRefKey</b>	Specifies the name of the device driver relative to which you are adding szDriver.
<b>nPosition</b>	Indicates whether szDriver is to be added before or after the driver specified by szRefKey. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>BEFORE</b>—The statement is added before the line containing szRefKey. If szRefKey contains a null string (""), the driver is inserted as the first driver in the system configuration file.</li> <li>• <b>AFTER</b>—The statement is added after the line containing szRefKey. If szRefKey contains a null string (""), the driver is inserted as the last driver in the system configuration file.</li> </ul>

## Return Values

**Table 25** • EzConfigAddDriver Return Values

Return Value	Description
<b>0</b>	EzConfigAddDriver successfully added the device driver statement to the file.
<b>&lt; 0</b>	EzConfigAddDriver was unable to add the driver statement.

## EzConfigAddDriver Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
```

```

*
* Demonstrates the EzConfigAddDriver function.
*
* This example calls EzConfigAddDriver to add four statements
* to a configuration file.
*
* -- The first call to EzConfigAddDriver adds a new line of
* text after the last line.
*
* -- The second call adds a new line of text before the first
* line.
*
* -- The third call adds the driver C:\DRDOS\HIDOS.SYS after
* the key HIMEM.SYS
*
* -- The fourth call adds the driver EXAPP.SYS before the key
* HIMEM.SYS.
*
* Note: Before running this script, create a configuration file
* named ISExempl.sys in the root of drive C. For best
* effect, include the following line in that file:
*
*     DEVICE=C:\\Himem.sys
*
\*-----*/

#define EXAMPLE_SYS "C:\\ISExempl.sys"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_EzConfigAddDriver(HWND);

function ExFn_EzConfigAddDriver(hMSI)
    STRING szDriver, szRefKey;
begin

    // Set the default configuration file.
    ConfigSetFileName (EXAMPLE_SYS);

    // Set up parameters for next call to EzConfigAddDriver.
    szDriver = "C:\\NEW\\MYAPP.DRV";
    szRefKey = "";

    // Add a driver to the end of the configuration file.
    if (EzConfigAddDriver (szDriver, szRefKey, AFTER) < 0) then
        MessageBox ("First call to EzConfigAddDriver failed.", SEVERE);
    else
        MessageBox ("First call to EzConfigAddDriver successful.", INFORMATION);
    endif;

    // Set up parameters for next call to EzConfigAddDriver.
    szDriver = "C:\\SYSTEM\\DMDVR.BIN";
    szRefKey = "";

    // Add a driver to the top of the configuration file.
    if (EzConfigAddDriver (szDriver, szRefKey, BEFORE) < 0) then
        MessageBox ("Second call to EzConfigAddDriver failed.", SEVERE);
    else

```

```

        MessageBox ("Second call to EzConfigAddDriver successful.", INFORMATION);
    endif;

    // Set up parameters for next call to EzConfigAddDriver.
    szDriver  = "C:\\DRDOS\\HIDOS.SYS";
    szRefKey  = "HIMEM.SYS";

    // Add the "HIDOS.SYS" driver after the "HIMEM.SYS" key.
    if (EzConfigAddDriver (szDriver, szRefKey, AFTER) < 0) then
        MessageBox ("Third call to EzConfigAddDriver failed.", SEVERE);
    else
        MessageBox ("Third call to EzConfigAddDriver successful.", INFORMATION);
    endif;

    // Set up parameters for next call to EzConfigAddDriver.
    szDriver  = "EXAPP.SYS";
    szRefKey  = "HIMEM.SYS";

    // Add the "EXAPP.SYS" driver before the "HIMEM.SYS" key.
    if (EzConfigAddDriver (szDriver, szRefKey, BEFORE) < 0) then
        MessageBox ("Fourth call to EzConfigAddDriver failed.", SEVERE);
    else
        MessageBox ("Fourth call to EzConfigAddDriver successful.", INFORMATION);
    endif;

end;

```

## EzConfigAddString

The **EzConfigAddString** function adds a line of text to the default system configuration file. You can specify the position of the line you add in reference to another statement in the file.

Unless changed by a call to [ConfigSetFileName](#), the default system configuration file is the Config.sys file that was executed by the system during the boot sequence. To make another file the default system configuration file, call [ConfigSetFileName](#). To determine the fully qualified name of the default system configuration file, call [ConfigGetFileName](#).



**Note** • Do not mix the Ez configuration file functions and the advanced configuration file functions. After calling [ConfigFileLoad](#), you cannot call any of the Ez configuration file functions until you call [ConfigFileSave](#) to save the file.

### Syntax

```
EzConfigAddString ( szLine, szRefKey, nOptions );
```

## Parameters

**Table 26 • EzConfigAddString Parameters**

Parameter	Description
<b>szLine</b>	Specifies the line of text to add to the system configuration file.
<b>szRefKey</b>	Specifies the reference key relative to which you want to position szLine in the system configuration file. EzConfigAddString searches the system configuration file for the reference key and places the contents of the parameter szLine before or after the line containing the key, depending on which constants is passed in nOptions.
<b>nOptions</b>	Indicates whether the line specified by szLine is to be added before or after the line containing szRefKey. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>BEFORE</b>—The line specified by szLine is added before the line containing szRefKey. If szRefKey contains a null string (""), szLine is inserted as the first line of the system configuration file.</li> <li>• <b>AFTER</b>—The line specified by szLine is added after the line containing szRefKey. If szRefKey contains a null string (""), szLine is inserted as the last line of the system configuration file.</li> </ul>

## Return Values

**Table 27 • EzConfigAddString Return Values**

Return Value	Description
<b>0</b>	EzConfigAddString successfully added the string to the default system configuration file.
<b>&lt; 0</b>	EzConfigAddString was unable to add the text string.

## EzConfigAddString Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the EzConfigAddString function.
*
* This example calls EzConfigAddString to add four lines to a
* configuration file.
```

```

*
* -- The first call adds a new line of text at the end of
* the file.
*
* -- The second call adds a new line of text at the top of
* the file.
*
* -- The third call adds the line FASTOPEN=512 after the key
* LASTDRIVE.
*
* -- The fourth call adds the line EXAPPHI=ON before the key
* EXAPPHI.SYS.
*
* Note: Before running this script, create a configuration
* file named ISExempl.sys and store it in the root of
* drive C. For best effect, that file should include
* the following lines:
*
* LASTDRIVE=F
* DEVICE=Exapphi.sys
*
\*-----*/

#define EXAMPLE_SYS "C:\\ISExempl.sys"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_EzConfigAddString(HWND);

function ExFn_EzConfigAddString(hMSI)
    STRING szLine, szRefKey;
begin

    // Set the default configuration file.
    ConfigSetFileName (EXAMPLE_SYS);

    // Set up parameters for next call to EzConfigAddString.
    szLine = "SHELL=C:\\COMMAND.COM /P /E:512";
    szRefKey = "";

    // Add a line to the end of the file.
    if (EzConfigAddString (szLine, szRefKey, AFTER) < 0) then
        MessageBox ("First call to EzConfigAddString failed.", SEVERE);
    else
        MessageBox ("First call to EzConfigAddString succeeded.", INFORMATION);
    endif;

    szLine = "DEVICE=C:\\System\\Dmdrvr.bin";
    szRefKey = "";

    // Add a line to the top of the file.
    if (EzConfigAddString (szLine, szRefKey, BEFORE) < 0) then
        MessageBox ("Second call to EzConfigAddString failed.", SEVERE);
    else
        MessageBox ("Second call to EzConfigAddString succeeded.", INFORMATION);
    endif;

    // Set up parameters for next call to EzConfigAddString.

```



```

szLine    = "FASTOPEN=512";
szRefKey = "LASTDRIVE";

// Add a line to the file after the key "LASTDRIVE".
if (EzConfigAddString (szLine, szRefKey, AFTER) < 0) then
    MessageBox ("Third call to EzConfigAddString failed.", SEVERE);
else
    MessageBox ("Third call to EzConfigAddString succeeded.", INFORMATION);
endif;

// Set up parameters for next call to EzConfigAddString.
szLine    = "EXAPPHI=ON";
szRefKey = "EXAPPHI.SYS";

// Add a line to the file before the key "Exapphi.sys".
if (EzConfigAddString (szLine, szRefKey, BEFORE) < 0) then
    MessageBox ("Fourth call to EzConfigAddString failed.", SEVERE);
else
    MessageBox ("Fourth call to EzConfigAddString succeeded.", INFORMATION);
endif;

end;

```

## EzConfigGetValue

The **EzConfigGetValue** function retrieves the numeric value of a parameter, such as FILES or BUFFERS, from the default system configuration file.

Unless changed by a call to [ConfigSetFileName](#), the default system configuration file is the Config.sys file that was executed by the system during the boot sequence. To make another file the default system configuration file, call [ConfigSetFileName](#). To determine the fully qualified name of the default system configuration file, call [ConfigGetFileName](#).




---

**Note** • Do not mix the Ez configuration file functions and the advanced configuration file functions. After calling [ConfigFileLoad](#), you cannot call any of the Ez configuration file functions until you call [ConfigFileSave](#) to save the file.

### Syntax

```
EzConfigGetValue ( szRefKey, nvValue );
```

## Parameters

**Table 28 • EzConfigGetValue Parameters**

Parameter	Description
<b>szRefKey</b>	Specifies the name of the parameter whose value is to be retrieved.
<b>nvValue</b>	Returns the numeric value of the key specified by szRefKey.

## Return Values

**Table 29 • EzConfigGetValue Return Values**

Return Value	Description
<b>0</b>	EzConfigGetValue successfully retrieved the value.
<b>&lt; 0</b>	EzConfigGetValue was unable to retrieve the value.

## EzConfigGetValue Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the EzConfigGetValue function.
*
* EzConfigGetValue is called to retrieve a value from a key in
* the default configuration file. Unless changed by a call to
* ConfigSetFileName, the default configuration file is the
* Config.sys file that resides in the root of the boot drive.
*
\*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_EzConfigGetValue(HWND);

function ExFn_EzConfigGetValue(hMSI)
    NUMBER nvValue;
begin

    // Retrieve the numeric value of the FILES key from the
    // default configuration file.
    if (EzConfigGetValue ("FILES", nvValue) < 0) then
        // Report the error.
        MessageBox ("EzConfigGetValue failed.", SEVERE);
    
```

```

else
    // Display the value of the FILES key.
    sprintfBox (INFORMATION, "EzConfigGetValue Example",
               "FILES = %d", nvValue);
endif;

end;

```

## EzConfigSetValue

The **EzConfigSetValue** function sets the value of a command in the default system configuration file.

Unless changed by a call to [ConfigSetFileName](#), the default system configuration file is the Config.sys file that was executed by the system during the boot sequence. To make another file the default system configuration file, call [ConfigSetFileName](#). To determine the fully qualified name of the default system configuration file, call [ConfigGetFileName](#).




---

**Note** • Do not mix the Ez configuration file functions and the advanced configuration file functions. After calling [ConfigFileLoad](#), you cannot call any of the Ez configuration file functions until you call [ConfigFileSave](#) to save the file.

### Syntax

```
EzConfigSetValue (szRefKey, nValue);
```

## Parameters

**Table 30** • EzConfigSetValue Parameters

Parameter	Description
<b>szRefKey</b>	Specifies the command whose value is to be changed or added in the default system configuration file. If the key does not exist in that file, it is added.
<b>nValue</b>	Specifies the new value of the command specified by szRefKey.

## Return Values

**Table 31** • EzConfigSetValue Return Values

Return Value	Description
<b>0</b>	EzConfigSetValue successfully set the value.
<b>&lt; 0</b>	EzConfigSetValue was unable to set the value.

## EzConfigSetValue Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the EzConfigSetValue function.
*
* EzConfigSetValue is called to add or change the BUFFER and
* FILES commands in the default configuration file.
*
* Note: The first time you run this script, it will create a
*       file named ISExempl.sys in the root of drive C. You
*       may delete that file when you have finished analyzing
*       this script.
*
\*-----*/

#define EXAMPLE_SYS "C:\\ISExempl.sys"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_EzConfigSetValue(HWND);

function ExFn_EzConfigSetValue(hMSI)
    STRING szRefKey, szMsg;
    NUMBER nValue;

```

```

begin

    // Set the default configuration file.
    ConfigSetFileName (EXAMPLE_SYS);

    // Add or change the BUFFERS command.
    if (EzConfigSetValue ("BUFFERS", 30) < 0) then
        MessageBox ("Unable to set Buffers in " + EXAMPLE_SYS + ".", SEVERE);
    else
        MessageBox ("Buffers set to 30 in " + EXAMPLE_SYS+".", INFORMATION);
    endif;

    // Add or change the FILES command.
    if (EzConfigSetValue ("FILES", 30) < 0) then
        MessageBox ("Unable to set Files in " + EXAMPLE_SYS + ".", SEVERE);
    else
        MessageBox ("Files set to 30 in " + EXAMPLE_SYS + ".", INFORMATION);
    endif;

end;

```

## EzDefineDialog

The **EzDefineDialog** function defines a custom dialog.




---



**Note** • This function does not display the custom dialog. To display a custom dialog, call [WaitOnDialog](#).

### Syntax

```
EzDefineDialog ( szDialogName, szDLLName, szDialogID, nDialogID );
```

## Parameters

**Table 32 • EzDefineDialog Parameters**

Parameter	Description
<b>szDialogName</b>	<p>Specifies the name to associate with the dialog identified by szDialogID or nDialogID. To process this dialog, use this name in subsequent calls to custom dialog functions.</p> <p>Note that the dialog's name is case-sensitive; you must use it exactly as you have specified it in this parameter.</p>
<b>szDLLName</b>	<p>Specifies the name of the .dll file that contains the dialog resource. If this name is not qualified (that is, if you do not specify the drive and path with the file name), the InstallScript engine searches for the .dll file in the Windows folder. If it is not found there, the InstallScript engine searches the folders that are specified in the search path. When the dialog is located in <b>_isuser.dll</b>, you can specify a null string (""), the InstallScript engine automatically checks <b>_isuser.dll</b> if this parameter is specified as a null string ("").</p> <p> <b>Note •</b> When you use a .dll file other than <b>_isres.dll</b> or <b>_isuser.dll</b>, you must call <a href="#">UseDLL</a> to load the .dll file before calling <b>EzDefineDialog</b>. To unload the .dll file from memory, call <a href="#">UnUseDLL</a> after calling <a href="#">ReleaseDialog</a>.</p>
<b>szDialogID</b>	<p>Specifies the dialog's resource ID if you use a string rather than a number to identify the resource. If this parameter is a null string (""), nDialogID is used to identify the dialog resources. It is recommended that you use nDialogID rather than szDialogID to identify the dialog resource.</p> <p> <b>Note •</b> If you created the dialog in the Dialogs view, the dialog is created with a string ID; therefore, you must specify this name as the szDialogID parameter to define the dialog.</p> <p>If you override an existing dialog in the Dialogs view, the dialog has a numeric ID, and you must specify the ID in the nDialogID parameter.</p> <p>If you want to use numeric IDs in all cases—which is what is recommended—you must edit the dialog in the Dialogs view to change the <b>ISResourceID</b> property of the created dialog to the desired Dialog ID instead of 0. Note that if you do this, the dialog name is no longer used when the dialog is created; it is used only in the Dialogs view in InstallShield to identify the dialog. The dialog is built with the correct numeric ID.</p>
<b>nDialogID</b>	<p>Specifies the dialog's resource ID if you use a number rather than a string to identify the resource. This parameter is used only when szDialogID is a null string ( ""). It is recommended that you use nDialogID rather than szDialogID to identify the dialog resource.</p>

## Return Values

**Table 33 • EzDefineDialog Return Values**

Return Value	Description
<b>0</b>	<b>EzDefineDialog</b> successfully defined the dialog.
<b>DLG_ERR_ALREADY_EXISTS (-3)</b>	Indicates that you are trying to define a dialog that has already been defined in the installation script. You cannot define two dialogs with the same name.
<b>DLG_ERR (-1)</b>	Indicates an unspecified error condition.

## EzDefineDialog Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the EzDefineDialog function.
*
* This script opens a simple custom dialog that displays
* a bitmap. The dialog can be closed with any of three
* buttons: Back, Next, or Cancel.
*
* The "custom" dialog used in this script is actually the
* InstallShield Sd dialog that is displayed by the built-in
* function SdBitmap. Because this dialog is stored in
* the file _isres.dll, which is already compressed in
* the installation, it can be used in a script as a custom
* dialog.
*
* In order to use this custom dialog, the script first defines
* it by calling EzDefineDialog. It then displays the dialog
* and gets dialog events by calling WaitOnDialog. When an
* event ends dialog processing, EndDialog is called to close
* the dialog. Then the dialog is released from memory by
* a call to ReleaseDialog.
*
\*-----*/

// Dialog and control IDs.
#define RES_DIALOG_ID    12027  // ID of dialog itself
#define RES_PBUT_NEXT    1      // ID of Next button
#define RES_PBUT_CANCEL  9      // ID of Cancel button
#define RES_PBUT_BACK    12     // ID of Back button

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"
```

```

export prototype ExFn_EzDefineDialog(HWND);

function ExFn_EzDefineDialog(hMSI)
    STRING  szDialogName, szDLLName, szDialog;
    NUMBER  nDialog, nResult, nCmdValue;
    BOOL     bDone;
    HWND     hInstance, hwndParent, hwndDlg;
begin

    // Specify a name to identify the custom dialog in this installation.
    szDialogName = "CustomDialog";

    // Define the dialog. Pass a null string in the second parameter
    // to get the dialog from _isuser.dll or _isres.dll. Pass a null
    // string in the third parameter because the dialog is identified
    // by its ID in the fourth parameter.
    nResult = EzDefineDialog (szDialogName, "", "", RES_DIALOG_ID);

    if (nResult < 0) then
        // Report an error; then terminate.
        MessageBox ("Error in defining dialog", SEVERE);
        abort;
    endif;

    // Initialize the indicator used to control the while loop.
    bDone = FALSE;

    // Loop until done.
    repeat

        // Display the dialog and return the next dialog event.
        nCmdValue = WaitOnDialog(szDialogName);

        // Respond to the event.
        switch (nCmdValue)
            case DLG_CLOSE:
                // The user clicked the window's Close button.
                Do (EXIT);
            case DLG_ERR:
                MessageBox ("Unable to display dialog. Setup canceled.", SEVERE);
                abort;
            case DLG_INIT:
                // Initialize the back, next, and cancel button enable/disable states
                // for this dialog and replace %P, %VS, %VI with
                // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
                // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
                hwndDlg = CmdGetHwndDlg(szDialogName);
                SdGeneralInit(szDialogName, hwndDlg, 0, "");
            case RES_PBUT_CANCEL:
                // The user clicked the Cancel button.
                Do (EXIT);
            case RES_PBUT_NEXT:
                bDone = TRUE;
            case RES_PBUT_BACK:
                bDone = TRUE;
            if (SdIsStdButton( nCmdValue ) && SdDoStdButton( nCmdValue )) then
                bDone = TRUE;
            endif;
        endswitch;
    repeat

```



```

until bDone;

// Close the dialog.
EndDialog (szDialogName);

// Free the dialog from memory.
ReleaseDialog (szDialogName);

end;

```

## FeatureAddCost



**Project** • This information applies to InstallScript projects.

The **FeatureAddCost** function specifies that the feature includes additional installation operations that should be accounted for when updating the progress bar during the installation. Specifically, the **FeatureAddCost** function adds the value of nCost to the specified feature. This function can be called anytime before calling **StatusUpdate**, but it should typically be used before displaying a feature dialog so that the dialog shows the intended size for the feature to be installed.



**Important** • The **FeatureAddCost** function is supported only for file media. Use **FeatureGetData** or **FeatureSetData** to set the size of script-created features.



**Note** • The information specified in this function is not remembered in subsequent installations. This function should be called each time that the feature is being installed, including during repair or maintenance mode. Also, the setup engine has no way to determine whether the specified cost is for a single file or multiple files. Therefore, it does not attempt to adjust the specified size based on the cluster size of the target drive. Thus, you should use the **GetAndAddFileCost**, **CalculateAndAddFileCost**, or **GetAndAddAllFilesCost** functions to determine the cost of the file, taking into account cluster size, before calling this function.

### Syntax

```
FeatureAddCost ( szMediaSource, szFeature, szTarget, nCostHigh, nCostLow );
```

## Parameters

**Table 34 • FeatureAddCost Parameters**

Parameter	Description
<b>szMediaSource</b>	The media source, typically MEDIA.
<b>szFeature</b>	Specifies the feature to add this cost to. Note that a specific feature in the media must be specified as installation cost is calculated by feature. Cost cannot be specified for the entire media.
<b>szTarget</b>	Specifies the target directory for the cost specified by nCost. This value is used when determining which volume or drive this cost applies to and when displaying the size required for the feature in the feature dialog. Currently, only a single szTarget variable for custom cost is supported for each feature. If you call this function multiple times for the same feature, szTarget variable should be the same each time. Otherwise, the value specified during the last call is used for the target of all additional cost for the feature.
<b>nCostHigh</b>	Specifies the upper 31 bits of the additional cost to be added to the feature.
<b>nCostLow</b>	Specifies the lower 31 bits of the additional cost to be added to the feature.

## Return Values

**Table 35 • FeatureAddCost Return Values**

Return Value	Description
<b>ISERR_SUCCESS</b>	Indicates that the function successfully added the cost.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was unable to add the cost.

# FeatureAddItem



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **FeatureAddItem** function adds a feature to a script-created feature set. If a script-created feature set with the name specified by szFeatureSet does not already exist, this function creates it.

Call **FeatureAddItem** once for each feature you want to add to a given script-created feature set. You can create multiple script-created feature sets, each with a unique name (szFeatureSet parameter).




**Caution** • This function cannot be used with file media libraries.

## Syntax

```
FeatureAddItem ( szFeatureSet, szFeature, nDataSize, bSelected );
```

## Parameters

**Table 36 • FeatureAddItem Parameters**

Parameter	Description
<b>szFeatureSet</b>	Specifies the name of the script-created feature set to which an item will be added. If the script-created feature set does not exist, <b>FeatureAddItem</b> creates it.
<b>szFeature</b>	Specifies the name of the feature to be added. Do not use a null string (""). To learn how to refer to top-level features and subfeatures, see <a href="#">Specifying Features and Subfeatures in Function Calls</a> .
<b>nDataSize</b>	Specifies the size in bytes of the data the feature represents. If the feature is a series of files, this parameter specifies the total uncompressed size of all the files.
<b>bSelected</b>	Specifies the default selection setting of the feature. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>TRUE</b>—Indicates that the feature is selected by default. If TRUE is passed to select a subfeature whose parent feature is not selected, the subfeature is not selected—despite passing TRUE in the bSelected parameter.</li> <li>• <b>FALSE</b>—Indicates that the feature is not selected by default.</li> </ul> <div>  <p><b>Note</b> • Default selection settings are cleared when the user selects a feature or subfeature displayed in a dialog. If the user clears a selected feature, all of its subfeatures are cleared. If the user clears all of a feature's subfeatures, the feature's selection setting is cleared.</p> </div>

## Return Values

**Table 37 • FeatureAddItem Return Values**

Return Value	Description
<b>0</b>	<b>FeatureAddItem</b> successfully added the item to the feature or subfeature to the script-created feature set.
<b>&lt; 0</b>	<b>FeatureAddItem</b> was unable to add the item to the feature or subfeature to the script-created feature set. For additional information, call <a href="#">FeatureError</a> .

## Additional Information

- To display a single level of features to the end user for selection, use [FeatureDialog](#), [SdFeatureDialog](#), or [SdFeatureDialogAdv](#).

- To display features and their subfeatures, use [SdFeatureDialog2](#), [SdFeatureMult](#), or [SdFeatureTree](#).
- You can use **FeatureAddItem** and script-created feature sets to allow users to select from options other than file media feature options:
  1. Call **FeatureAddItem** to create a script-created feature set containing the options you want.
  2. To display those options for selection, call **SdAskOptions** or **SdAskOptionsList** with the script-created feature set name in the parameter szFeature.
  3. To determine which options were selected, call **FeatureIsItemSelected**.

## FeatureAddItem Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the FeatureAddItem function.
 *
 * In this script, FeatureAddItem is called to create a
 * script-created feature set consisting of drives attached
 * to the target system. These features are displayed in an
 * SdAskOptionsList dialog so that the end user can select
 * from among the drives.
 *
 \*-----*/

LIST listDrives, listFeatures;
STRING szSaveMEDIAValue, svDrive;
LONG nResult;
BOOL bSelected;

#include "ifx.h"

function OnBegin()
begin

    // Create the list to store drive names.
    listDrives = ListCreate(STRINGLIST);

    // Add all fixed drives to the list.
    GetValidDrivesList(listDrives, FIXED_DRIVE, -1);

    // Save the value of the system variable MEDIA so it can
    // be restored for later calls to transfer data.
    szSaveMEDIAValue = MEDIA;

    // Specify a name for the script-created feature set.
    MEDIA = "Drives";

    // Put the drive names into the script-created feature set.
    nResult = ListGetFirstString(listDrives, svDrive);
    while (nResult != END_OF_LIST)
        FeatureAddItem(MEDIA, svDrive, 0, FALSE);
        nResult = ListGetNextString(listDrives, svDrive);
    endwhile;
end

```

```

// Display the list of drives. Let user select one drive.
SdAskOptionsList("Select a Drive" ,
"Select one of the drives attached to your system." ,
"" , EXCLUSIVE);

// Find the selected drive in the list.
bSelected = FALSE;
nResult = ListGetFirstString(listDrives, svDrive);
bSelected = ComponentIsItemSelected (MEDIA , svDrive);
while ((nResult != END_OF_LIST) && !bSelected);
    nResult = ListGetNextString(listDrives, svDrive);
    bSelected = ComponentIsItemSelected (MEDIA , svDrive);
endwhile;

// Release the list from memory.
ListDestroy (listDrives);
if bSelected then
    MessageBox("You selected drive " + svDrive + ".", INFORMATION);
endif;

// Restore MEDIA to its previous value for file transfer.
MEDIA = szSaveMEDIAValue;

end;

```

## FeatureAddUninstallCost



**Project** • This information applies to InstallScript projects.


The **FeatureAddUninstallCost** function specifies that the feature includes additional uninstallation operations that must be accounted for when updating the progress bar during uninstallation. You do not have to call this function for operations that are included in the uninstallation log file. This includes operations that are handled by the **XCopyFile** function that may or may not have had to call the **FeatureAddCost** function to update the progress bar during the installation. This function is only needed for operations that were carried out by functionality other than the InstallScript engine. This function can be called anytime before calling **StatusUpdate** and can be called multiple times.

### Syntax

```
FeatureAddUninstallCost ( szMediaSource, szFeature, nOps, nOpType);
```

## Parameters

**Table 38 • FeatureAddUninstallCost Parameters**

Parameter	Description
<b>szMediaSource</b>	The media source, typically MEDIA.
<b>szFeature</b>	Specifies the feature to add this cost to. Note that a specific feature in the media must be specified. Uninstallation cost is calculated by feature. Cost cannot be specified for the entire media.
<b>nOps</b>	The number of operations to be added.
<b>nOpType</b>	<p>Indicates the type of operation. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>FEATURE_OPCOST_UNINSTALL_FILE</b>—Specifies the operation as uninstalling a file. You can also specify this constant using the value 4096. This enables you to experiment with different sizes to determine what size to use for your custom operation.</li> <li>• <b>FEATURE_OPCOST_UNINSTALL_REGORINI</b>—Specifies the operation as uninstalling a registry file. You can also specify this constant using the value 2048. This enables you to experiment with different sizes to determine what size to use for your custom operation.</li> <li>• <b>FEATURE_OPCOST_UNINSTALL_UNREGFILE</b>—Specifies the operation as unregistering a file. You can also specify this constant using the value 16384. This enables you to experiment with different sizes to determine what size to use for your custom operation.</li> </ul> <p> <b>Note</b> • You can also pass a numeric value in this parameter to indicate an operation of a specific cost. You may need to experiment with different values to determine the appropriate value for a specified custom operation. Note that the total uninstall cost added is nOps multiplied by nOpType.</p>

## Return Values

**Table 39 • FeatureAddUninstallCost Return Values**

Return Value	Description
<b>ISERR_SUCCESS</b>	Indicates that the function successfully added the cost.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was unable to add the cost.

# FeatureCompareSizeRequired



---

**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **FeatureCompareSizeRequired** function determines whether the target folder contains enough free space for the selected features specified by `szFeatureSource`, which must be a file media. If the target folder does not have sufficient free space, the fully qualified folder name is returned in `svTarget` and the amount of required free space is returned in `nvSize`.

If your setup specifies a destination folder for a feature in a file library at run time, it must do so by calling [FeatureSetTarget](#) before checking for free space with **FeatureCompareSizeRequired**.



---


**Note** • This function cannot be used with script-created feature sets.

## Syntax

```
FeatureCompareSizeRequired ( szFeatureSource, svTarget, nvSize );
```

## Parameters

**Table 40** • FeatureCompareSizeRequired Parameters

Parameter	Description
<b>szFeatureSource</b>	You must pass the system variable <a href="#">MEDIA</a> in this parameter. No other value is valid.
<b>svTarget</b>	Returns a null string ("" ) if there is sufficient free space available on the target drive. Returns the target path if there is <i>not</i> sufficient free space on the target drive.   <b>Note</b> • The parameter <i>svTarget</i> is used only to return a folder name. You cannot use the parameter to specify a destination folder. <b>FeatureCompareSizeRequired</b> checks the drive that is indicated by the destination folder that you specified for each component in the Components view. If <i>szFeatureSource</i> is a file media with features to be installed in the General Application Destination, you must assign a destination path to <i>TARGETDIR</i> (in InstallScript installations) or <i>INSTALLDIR</i> (in InstallScript MSI installations) before calling <b>FeatureCompareSizeRequired</b> . You can obtain a destination path from an end user by calling <a href="#">AskDestPath</a> or a feature dialog.
<b>nvSize</b>	Returns 0 if there is sufficient free space available on the target drive. Returns the size (in bytes) required if there is <i>not</i> sufficient free space on the target drive.

## Return Values

**Table 41** • FeatureCompareSizeRequired Return Values

Return Value	Description
<b>0</b>	FeatureCompareSizeRequired was successful.
<b>&lt; 0</b>	FeatureCompareSizeRequired failed. Call <a href="#">FeatureError</a> for additional information.

## FeatureCompareSizeRequired Example

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the FeatureAddItem, FeatureSetData,
* FeatureCompareSizeRequired, and SdFeatureMult functions.
*
* Notes: DESTDIR must reference a drive with very little space
*        available, such as a floppy drive. Remember to put a
*        diskette into the floppy drive before running this

```



```

*      script.
*
\*-----*/

#define FEAT1          "CAD Prog. Files"
#define FEAT1SIZE      25000000
#define FEAT1DESC      "CAD program EXEs and DLLs."
#define FEAT2          "CAD Templates"
#define FEAT2SIZE      18000000
#define FEAT2DESC      "CAD template files."
#define SUBFEAT1       "Industrial"
#define SUBFEAT1SIZE   7000000
#define SUBFEAT1DESC   "CAD industrial engineering template files."
#define SUBFEAT2       "Civil"
#define SUBFEAT2SIZE   5000000
#define SUBFEAT2DESC   "CAD civil engineering template files."
#define SUBFEAT3       "Mechanical"
#define SUBFEAT3SIZE   6000000
#define SUBFEAT3DESC   "CAD mechanical engineering template files."
#define DESTDIR        "a:\\"
#define SDFEATTITLE    "Displaying Script-created Features"
#define SDFEATMSG      "Make sure all features are selected."
#define FEATSIZEERRMSG "Make sure target drive is accessible."

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_FeatureCompareSizeRequired(HWND);

function ExFn_FeatureCompareSizeRequired(hMSI)
    STRING svDir, svTarget;
    NUMBER nvSize, nData;
begin

    // Disable the Back button in setup dialogs. Disable (BACKBUTTON);
    // Make a script-created feature set that includes subfeatures
    // and give it the media name "Run-time CAD".
    MEDIA = "Run-time CAD";

    FeatureAddItem (MEDIA, FEAT1, FEAT1SIZE, TRUE);

    FeatureSetData (MEDIA, FEAT1, FEATURE_FIELD_DESCRIPTION,
                    nData, FEAT1DESC);

    FeatureAddItem (MEDIA, FEAT2, FEAT1SIZE, TRUE);

    FeatureSetData (MEDIA, FEAT2, FEATURE_FIELD_DESCRIPTION, nData, FEAT2DESC);

    FeatureAddItem (MEDIA, FEAT2 + "\\" + SUBFEAT1, SUBFEAT1SIZE, TRUE);

    FeatureSetData (MEDIA, FEAT2 + "\\" + SUBFEAT1, FEATURE_FIELD_DESCRIPTION,
                    nData, SUBFEAT1DESC);

    FeatureAddItem (MEDIA, FEAT2 + "\\" + SUBFEAT2, SUBFEAT2SIZE, TRUE);

    FeatureSetData (MEDIA, FEAT2 + "\\" + SUBFEAT2, FEATURE_FIELD_DESCRIPTION,
                    nData, SUBFEAT2DESC);

    FeatureAddItem (MEDIA, FEAT2 + "\\" + SUBFEAT3, SUBFEAT3SIZE, TRUE);

```

```

FeatureSetData (MEDIA, FEAT2 + "\\\" + SUBFEAT3, FEATURE_FIELD_DESCRIPTION,
               nData, SUBFEAT3DESC);

// Display the script-created features and subfeatures.
SdFeatureMult (SDFEATTITLE, SDFEATMSG, svDir, "");

// Set INSTALLDIR to DESTDIR, which, if you define as a floppy drive
// (with a diskette in it), causes FeatureCompareSizeRequired
// to display its 'not enough space' message.
nvSize = 0;

INSTALLDIR = DESTDIR;

if (FeatureCompareSizeRequired(MEDIA, svTarget, nvSize) < 0) then
    MessageBox (FEATSIZEERRMSG, SEVERE);
endif;

end;

```

## FeatureConfigureFeaturesFromSuite



**Project** • This information applies to InstallScript projects.



**Note** • This function is available for InstallScript installations that may be included as InstallScript packages in an Advanced UI or Suite/Advanced UI installation. It is also available in an InstallScript installation that is included in an Advanced UI or Suite/Advanced UI installation as an executable package. For more information, see [Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project](#).

In addition, this function is available in an InstallScript installation that is launched directly (that is, not from an Advanced UI or Suite/Advanced UI installation).

The **FeatureConfigureFeaturesFromSuite** function sets feature states for the current InstallScript package that is running in an Advanced UI or Suite/Advanced UI installation based on the values of the Advanced UI or Suite/Advanced UI properties **ISFeatureInstall** and **ISFeatureRemove**. The function is called by the default code in the [OnSuiteInstallBefore event](#) (for an install operation) and [OnSuiteMaintBefore event](#) (for a modify operation).

The **FeatureConfigureFeaturesFromSuite** function also lets you set the feature state for InstallScript installations that are launched as an executable package in an Advanced UI or Suite/Advanced UI installation, or that are launched directly outside an Advanced UI or Suite/Advanced UI installation. In these cases, the function sets feature states based on the **ISFeatureInstall** and **ISFeatureRemove** key-value pairs.

### Syntax

```
FeatureConfigureFeaturesFromSuite (string szCommandLine);
```

## Parameters

**Table 42** • FeatureConfigureFeaturesFromSuite Parameters

Parameter	Description
<b>szCommandLine</b>	<p>Specify the command line that you want to use to set <b>ISFeatureInstall</b>, <b>ISFeatureRemove</b>, or both. Set these to a comma-delimited list of feature names as follows:</p> <pre>ISFeatureInstall=Feature1,Feature2 ISFeatureRemove=Feature3</pre> <p>In the above example, Feature1 and Feature2 are selected to be installed; Feature3 is selected to be removed, if it is present.</p> <p>If any features are used in the values of both properties, the features that are set for the <b>ISFeatureRemove</b> property supersede the <b>ISFeatureInstall</b> property.</p> <p>If szCommandLine does not set either property or both properties, the call to <b>FeatureConfigureFeaturesFromSuite</b> has no effect.</p> <p>To learn how to refer to top-level features and subfeatures for <b>ISFeatureInstall</b> and <b>ISFeatureRemove</b>, see Specifying Features and Subfeatures in Function Calls.</p>

## Return Values

This function has no return values.

# FeatureDialog



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **FeatureDialog** function displays a dialog that enables the end user to select one or more items from a list of features in the installation. The end user can also select a destination location.

When the end user clicks the Browse button, the Choose Folder dialog, which displays a list of existing folders, opens. The end user can select an existing folder from the list or enter a new folder name in the Path field.

**FeatureDialog** returns the name of the specified folder in svDir. If the user specifies a folder that does not currently exist, the installation creates the folder.



**Caution** • If your installation does not use a setup type dialog, you must call [FeatureSetupTypeSet](#) to specify a setup type that has been defined in the Setup Types view in InstallShield before calling **FeatureDialog**.

## Syntax

```
FeatureDialog ( szTitle, szMsg, svDir, szFeature );
```

## Parameters

**Table 43 • FeatureDialog Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the dialog title. To display the default title (“Select Features”), pass a null string (“”) in this parameter.
<b>szMsg</b>	Specifies the message to display in the dialog. To display the default instructions for this dialog, pass a null string (“”) in this parameter.
<b>svDir</b>	<p>Specifies the default destination location variable. Returns the folder selected by the end user. The location returned in svDir does not affect file transfer unless you assign it to TARGETDIR (in InstallScript installations) or INSTALLDIR (in InstallScript MSI installations).</p> <p>If the default folder specified by svDir does not already exist on the end user’s system, it will not be created unless the end user clicks the Browse button and follows the steps to create it from the Choose Folder dialog. Therefore, whenever you specify a default folder, you must call <a href="#">ExistsDir</a> when FeatureDialog returns in order to determine whether that folder exists. If it does not exist, call <a href="#">CreateDir</a> to create it on the end user’s system.</p>
<b>szFeature</b>	<p>Specifies the feature whose subfeatures are displayed for selection. Pass a null string (“”) in this parameter to display all top-level features. To learn how to refer to top-level features and subfeatures, see <a href="#">Specifying Features and Subfeatures in Function Calls</a>.</p> <p>FeatureDialog searches for the requested features in the script-created feature set specified by the system variable <a href="#">MEDIA</a>.</p> <div data-bbox="626 1213 665 1260" data-label="Image"> </div> <p><b>Note •</b> <i>If necessary, feature names are truncated to allow the display of the largest possible feature size. The space required to display the size depends on the maximum feature size (2 GB), the feature size options currently in use, and the font used to display feature information in the dialog. Feature size options are set with the <a href="#">DialogSetInfo</a> function.</i></p> <p><i>When the space required to display the maximum possible size has been determined, all feature names are truncated automatically—if necessary—to fit the remaining space. This ensures that feature names will not overlap feature sizes.</i></p> <p><i>Note that the name of a feature that requires less space to display its size (or that is not selected) may still be truncated under this method. To maximize performance and ensure that complete feature names are displayed, make feature names or display names smaller than the space available in the dialog.</i></p>

## Return Values

**Table 44 • FeatureDialog Parameters**

Return Value	Description
<b>NEXT (1)</b>	Indicates that the end user clicked the Next button.
<b>BACK (12)</b>	Indicates that the end user clicked the Back button.
<b>&lt; 0</b>	Unable to display the FeatureDialog dialog. Call <a href="#">FeatureError</a> for additional information.

## Additional Information

The Choose Folder dialog indicates the amount of disk space that each feature occupies. A feature's size is displayed as 0 if it is not selected. Once it has been selected, its actual size is displayed.

## FeatureDialog Example



**Project •** This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the FeatureDialog function.
 *
 * This example script displays a dialog that displays a list
 * of features in the setup that the user can install and the
 * amount of space that each feature occupies.
 *
 * Comments: To run this example script, create a project (or
 *           insert into a project) with several features
 *           and/or subfeatures with components containing
 *           files.
 *
 \*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

// Include iswi.h for Windows Installer API function prototypes and constants,
// and to declare code for the OnBegin and OnEnd events.
#include "iswi.h"

// The keyword export identifies MyFunction() as an entry-point function.
// The argument it accepts must be a handle to the Installer database.
export prototype MyFunction(HWND);
```

```
// To Do: Declare global variables, define constants, and prototype user-
//         defined and DLL functions here.

function MyFunction(hMSI)

    STRING szTitle, szMsg, svDir;

begin

    svDir = INSTALLDIR;
    szTitle = "Select Features";
    szMsg = "Select the features you want to install on your computer.";

    // Display available features.
    FeatureDialog (szTitle, szMsg, svDir, "");

end;
```

## FeatureError



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

**FeatureError** does not return error information for unsupported functions. For example, ComponentInitialize, ComponentUpdate, and ComponentValidate are not supported as feature functions. FeatureError does not work for these functions.

The FeatureError function obtains additional error information when a feature function returns a value less than zero. The following code fragment shows a typical implementation of FeatureError:

```
nResult = FeatureGetData (MEDIA, svFeature, FEATURE_FIELD_SELECTED, nvResult,
                          svResult);

if(nResult < 0) then
    FeatureError(svFeatureSource, svFeature, svComponent, svFile, nvError);

    sprintfBox (INFORMATION, "FeatureGetData Error Information",
               "FeatureGetData had the following error:\n\n" +
               "Media Name: %s\nFeature: %s\nComponent: %s\n" +
               "File: %s\nError Number: %ld"
               svFeatureSource, svFeature, svComponent, svFile, nvError);

endif;
```

The FeatureError function should be called only after another feature function returns a value less than zero. FeatureError might return invalid error codes if called when another feature function has not returned a value that is less than zero.



**Note** • FeatureError returns information for all parameters (media name, feature name, component name, file name, and error code) only for [script-created feature sets](#). For features in the file media (MEDIA), FeatureError returns only the error code.

## Syntax

```
FeatureError ( svFeatureSource, svFeature, svComponent, svFile, nvError );
```

## Parameters

**Table 45 • FeatureError Parameters**

Parameter	Description
<b>svFeatureSource</b>	Returns the media name of the script-created feature set if relevant to the error indicated by nvError.
<b>svFeature</b>	Returns the name of the feature if relevant to the error indicated by nvError.
<b>svComponent</b>	Returns the name of the component if relevant to the error indicated by nvError.
<b>svFile</b>	Returns the name of the file if relevant to the error indicated by nvError.
<b>nvError</b>	Returns a code that identifies the type of error that occurred in a previous call to a feature-related function. These error codes are described below.

## Error Codes

The following table describes the error codes returned by FeatureError.



**Note** • After correcting errors involving the media, you must rebuild the project.

**Table 46 • FeatureError Error Codes**

Code	Description	Cause
<b>-101</b>	Cannot add feature.	<a href="#">FeatureAddItem</a> was unable to add a feature to the script-created feature set.
<b>-102</b>	Specified feature already exists.	FeatureAddItem was called twice with the same media name and feature name.
<b>-103</b>	Specified feature cannot be selected or deselected.	<a href="#">FeatureSelectItem</a> was used to select or deselect a feature that is required by a currently selected feature. Deselect the dependent feature (that requires the other feature) before attempting to select or deselect the required feature.
<b>-105</b>	Specified feature cannot be found in the media.	An attempt was made to access a feature that does not exist. This error occurs when a feature name is specified incorrectly in a feature function call. Feature names must be specified exactly as they appear in the Setup Design view or in the call to <a href="#">FeatureAddItem</a> . Feature names are case-sensitive.



**Table 46 • FeatureError Error Codes (cont.)**

Code	Description	Cause
<b>-108</b>	Out of disk space.	The target disk or directory has insufficient free space, the disk space cannot be determined because TARGETDIR (in an InstallScript installation) or INSTALLDIR (in an InstallScript MSI installation) is invalid, or a script-defined folder of a feature has not been set.
<b>-118</b>	Attempted operation not allowed with script-created feature sets.	A script-created feature set name was passed to a feature function that operates only on file media.
<b>-125</b>	The list specified in the feature function is not valid.	When calling <a href="#">FeatureListItems</a> , verify that the list you are passing to the function is valid.
<b>-126</b>	Attempted operation not allowed with file media library.	A file media name was passed to a feature function (for example, <a href="#">FeatureAddItem</a> ), that operates only on script-created feature sets.
<b>-132</b>	The specified media cannot be found.	The media has been declared, but it not associated with any features. Make sure that script-created features are associated with the media.
<b>-136</b>	Unable to allocate memory.	Insufficient memory is available to the setup. Display a message to the end user to close all other applications or to cancel the setup, reboot the system, and restart the setup.
<b>-137</b>	Specified option is not valid.	An invalid option has been specified for a feature function—for example, specifying only a component when both a component and file name are required.
<b>-147</b>	Invalid value passed to a feature-related function.	One of the values passed to a feature function is invalid. This error can be caused, for example, by passing an empty string in the second parameter of <a href="#">FeatureAddItem</a> .

## Return Values

**Table 47 • FeatureError Return Values**

Return Value	Description
<b>0</b>	FeatureError was successful.
<b>&lt; 0</b>	FeatureError failed.

## FeatureError Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the FeatureError function.
 *
 * This example script demonstrates a function that adds a feature
 * to a script-created feature set.
 *
 * Comments: To run this example script, create a project (or
 *           insert into a project) with several features and/or
 *           subfeatures with components containing files.
 *
\*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

// Include iswi.h for Windows Installer API function prototypes and constants,
// and to declare code for the OnBegin and OnEnd events.
#include "iswi.h"

// The keyword export identifies MyFunction() as an entry-point function.
// The argument it accepts must be a handle to the Installer database.
export prototype MyFunction(HWND);
prototype HandleMoveDataError(number);

// To Do: Declare global variables, define constants, and prototype user-
//        defined and DLL functions here.

function MyFunction(hMSI)

    STRING  szTitle, szMsg;
    NUMBER  listID, nResult;

begin

    szTitle = "List MEDIA Features";
    szMsg   = "MEDIA contains the following top-level features:";

    // Initialize the string list.
    listID = ListCreate (STRINGLIST);

    // Create a list of top-level features in the specified media.

```

```

nResult = FeatureListItems (MEDIA, "", listID);

// Call the error handler function.
HandleMoveDataError (nResult);

// Display a list of top-level features.
SdShowInfoList (szTitle, szMsg, listID);

end;

function HandleMoveDataError(nResult)

    STRING szErrMsg, svFeature , svComponent , svFile;

begin

    svFeature = "";
    svComponent = "";
    svFile = "";

    // In cases where FeatureListItems returns a value not equal to zero, display an
    // error message.
    switch (nResult)
    case 0:
        return 0;
    default:
        FeatureError (MEDIA , svFeature , svComponent , svFile , nResult);
        szErrMsg = "An error occurred during the process: %d" + "\n\n" +
            "Feature: " + svFeature + "\n" +
            "Component: " + svComponent + "\n" +
            "File: " + svFile;
        sprintfBox (SEVERE, "", szErrMsg, nResult);
        return nResult;
    endswitch;

end;

```

## FeatureErrorInfo



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **FeatureErrorInfo** function, which is called in the default code for the OnComponentError event handler, returns information about a file transfer error that does not generate any error event other than FeatureError—for example, FileLocked or SelfRegistrationError.



**Note** • This function returns information only about file transfer errors. For information about errors from script-created features, call [FeatureError](#).

## Syntax

```
FeatureErrorInfo ( );
```

## Parameters

None

## Return Values

A reference that can be assigned to a variable of type OBJECT by using the set keyword. Upon assignment, that variable (oErrorInfo in the following example) has the following properties:

**Table 48 • FeatureErrorInfo Return Values**

Property	Description
<b>oErrorInfo.Feature</b>	An object whose properties provide information about the feature that was being transferred when the error occurred. If the error is not associated with a particular feature, or the feature cannot be identified, this property is not set; test for this case by checking the value of IsObject (oErrorInfo.Feature).
<b>oErrorInfo.Feature.DisplayName</b>	The display name of the feature that was being transferred when the error occurred. If you did not specify a display name for this feature, this string is null ("").
<b>oErrorInfo.Feature.Name</b>	The name of the feature that was being transferred when the error occurred.
<b>oErrorInfo.Feature.Description</b>	A string describing the file transfer error. This string may be null ("").
<b>oErrorInfo.LastError</b>	The numeric code for the file transfer error.

## FeatureErrorInfo Example

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the FeatureError function.
 *
 * This example script demonstrates a function that adds a feature
 * to a script-created feature set.
 *
 * Comments: To run this example script, create a project (or
 *           insert into a project) with several features and/or
 *           subfeatures with components containing files.
 *
 \*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

// Include iswi.h for Windows Installer API function prototypes and constants,
// and to declare code for the OnBegin and OnEnd events.
```

```

#include "iswi.h"

// The keyword export identifies MyFunction() as an entry-point function.
// The argument it accepts must be a handle to the Installer database.
export prototype MyFunction(HWND);
prototype HandleMoveDataError(number);

// To Do: Declare global variables, define constants, and prototype user-
//        defined and DLL functions here.

function MyFunction(hMSI)

    STRING  szTitle, szMsg;
    NUMBER  listID, nResult;

begin

    szTitle = "List MEDIA Features";
    szMsg   = "MEDIA contains the following top-level features:";

    // Initialize the string list.
    listID = ListCreate (STRINGLIST);

    // Create a list of top-level features in the specified media.
    nResult = FeatureListItems (MEDIA, "", listID);

    // Call the error handler function.
    HandleMoveDataError (nResult);

    // Display a list of top-level features.
    SdShowInfolist (szTitle, szMsg, listID);

end;

function HandleMoveDataError(nResult)

    STRING szErrMsg, svFeature , svComponent , svFile;

begin

    svFeature = "";
    svComponent = "";
    svFile = "";

    // In cases where FeatureListItems returns a value not equal to zero, display an
    // error message.
    switch (nResult)
    case 0:
        return 0;
    default:
        FeatureError (MEDIA , svFeature , svComponent , svFile , nResult);
        szErrMsg = "An error occurred during the process: %d" + "\n\n" +
            "Feature: " + svFeature + "\n" +
            "Component: " + svComponent + "\n" +
            "File: " + svFile;
        SprintfBox (SEVERE, "", szErrMsg, nResult);
        return nResult;
    endswitch;
end;

```

```
end;
```

## FeatureFileEnum



---

**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **FeatureFileEnum** function builds a list of the files in a component associated with the specified feature or a list of the components associated with a particular feature.



---


**Note** • This function cannot be used with script-created feature sets.

### Syntax

```
FeatureFileEnum ( szFeatureSource, szFeature, szQuery, listFilesorComponents, nOption );
```

## Parameters

**Table 49 • FeatureFileEnum Parameters**

Parameter	Description
<b>szFeatureSource</b>	Pass MEDIA in this parameter.
<b>szFeature</b>	Specifies the name of the feature to be enumerated. Do not pass a null string ("") in this parameter. <b>FeatureFileEnum</b> requires a specific feature name.
<b>szQuery</b>	<p>To query the components in a particular feature, specify a component specification (same format as a file specification). Delimit the component specification with double backslashes and enclose the expression in double quotation marks.</p> <p>The component specification can include wild-card characters: an asterisk (*) as a substitute for zero or more characters, or a question mark (?) as a substitute for a single character. The following example specifies all of the components in the feature specified in szFeature:</p> <pre>"*.*"</pre> <p>To query the files in a particular component, specify a component name and a file specification. Delimit the component name and the file specification with double backslashes and enclose the expression in double quotation marks. You can also specify only a file specification. The file name part of the file specification may include wild-card characters. The following example specifies all of the files in the component Graphic_Examples:</p> <pre>"Graphic_Examples\\*.*"</pre> <p></p> <p><b>Note •</b> <i>There are limitations when using this functionality in InstallScript MSI installations (both when querying components and files), because complex wild-card expressions are not supported.</i></p> <p><i>When querying files, you must specify one of the following as szQuery:</i></p> <ul style="list-style-type: none"> <li>• <code>&lt;Component Name&gt;\\*.*</code></li> <li>• <code>&lt;Component Name&gt;\\&lt;Complete File Name&gt;</code></li> <li>• <code>&lt;Component Name&gt;\\*.*&lt;Extension&gt;</code></li> <li>• <code>&lt;Component Name&gt;\\&lt;File Name&gt;.*</code></li> </ul> <p><i>When querying components, you must specify one of the following as szQuery:</i></p> <ul style="list-style-type: none"> <li>• <code>*.*</code></li> <li>• <code>&lt;Complete Component&gt;</code></li> </ul>
<b>listFilesOrComponents</b>	When querying components, returns a list of the components that match szQuery. When querying files, returns a list of the names of all files that match szQuery. The string list identified by listFilesOrComponents must already have been initialized by a call to <a href="#">ListCreate</a> .

**Table 49** • FeatureFileEnum Parameters (cont.)

Parameter	Description
<b>nOption</b>	Pass NO_SUBDIR in this parameter. Note that nOptions is ignored when querying the list of components.

## Return Values

**Table 50** • FeatureFileEnum Return Values

Return Value	Description
<b>0</b>	FeatureFileEnum was successful.
<b>&lt;0</b>	FeatureFileEnum failed.

## FeatureFileEnum Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the FeatureFileEnum function.
 *
 * Notes: To run this example script, create a project with the
 *        following features (f), subfeatures (sf), and
 *        components (c):
 *
 *        (f) Program_Files
 *            (c) Program_DLLS
 *            (c) Program_EXEs
 *        (f) Example_Files
 *            (sf) Graphics
 *                (c) Graphic_Examples
 *
 *        Be sure to assign files to the components so there
 *        is something to enumerate.
 *
 \*-----*/

#define FEAT1      "Program Files"
#define FEAT2      "Example Files"
#define SUBFEAT1   "Graphics"
#define EXECCOMP   "Program Executable Files"
#define GRAPHCOMP  "Graphic Examples"
#define SDSHOWTITLE "FeatureFileEnum Results"

#define SDSHOWMSG1  FEAT1 + " enumerated files:"
#define SDSHOWMSG2  FEAT2 + " enumerated files:"

prototype HandleFeatureError (NUMBER);

NUMBER nResult;
LIST  listList1, listList2;

```



```

program

    // Create two lists to store file names.
    listList1 = ListCreate (STRINGLIST);
    listList2 = ListCreate (STRINGLIST);

    // Build a list of the program files.
    nResult = FeatureFileEnum (MEDIA, FEAT1, EXECCOMP + "\\*.\"",
                              listList1, INCLUDE_SUBDIR);

    HandleFeatureError (nResult);

    // Build a list of the graphic files.
    nResult = FeatureFileEnum (MEDIA, FEAT2 + "\\\" + SUBFEAT1,
                              GRAPHCOMP+"\\*.\"", listList2, INCLUDE_SUBDIR);

    HandleFeatureError (nResult);

    // Display the program files.
    SdShowInfoList (SDSHOWTITLE, SDSHOWMSG1, listList1);

    // Display the graphic files.
    SdShowInfoList (SDSHOWTITLE, SDSHOWMSG2, listList2);

    // Release the lists from memory.

    ListDestroy (listList1);
    ListDestroy (listList2);

endprogram

/*-----*\
*
* Function:  HandleFeatureError
*
* Purpose:  This function evaluates the value returned by a feature
*           function and if the value is less than zero, displays the error
*           number and aborts the setup.
*
*\-----*/

function HandleFeatureError( nResult )

    NUMBER  nvError;
    STRING  svFeatureSource, svFeature, svComponent, svFile;

begin
    if (nResult < 0) then

        FeatureError (svFeatureSource, svFeature, svComponent, svFile, nvError);

        SprintfBox (INFORMATION, "Data Transfer Error Information",
                    "FeatureError returned the following data transfer error.\n" +
                    "Setup will now abort.\n\n" +
                    "Media Name: %s\nFeature: %s\nComponent: %s\n" +

                    "File: %s\nError Number: %ld",
                    svFeatureSource, svFeature, svComponent, svFile, nvError);
    end if
end function

```

```
        abort;  
    endif;  
end;
```

## FeatureFileInfo



---

**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

*Note that this function does not work with files in InstallScript Object projects.*

The **FeatureFileInfo** retrieves information on a file in the file media that is referenced by szFeatureSource (MEDIA).



---

**Note** • This function cannot be used with script-created features.

### Syntax

```
FeatureFileInfo ( szFeatureSource, szFeature, szFile, nInfo, nvResult, svResult );
```

## Parameters

**Table 51** • FeatureFileInfo Parameters

Parameter	Description
<b>szFeatureSource</b>	Pass MEDIA in this parameter.
<b>szFeature</b>	Specifies the feature containing the component in which the file is found.
<b>szFile</b>	<p>Specifies the component and the file name, with or without a path. Delimit tokens in the szFile expression with double backslashes and enclose the expression in double quotation marks.</p> <ul style="list-style-type: none"> <li>If there are two tokens in szFile, the first is the component name and the second is the file name:</li> </ul> <p><b>"component\\file name"</b></p> <ul style="list-style-type: none"> <li>If there are more than two tokens in szFile, the first is the component name, the last is the file name, and the middle tokens form the path:</li> </ul> <p><b>"component\\path\\file name"</b></p> <p>For example:</p> <ul style="list-style-type: none"> <li>"Shared_Files\\Is5.dll"—Shared_Files is a component and Is5.dll is a file name.</li> <li>"Shared_Files\\dev\\myapp\\dlls\\Is5.dll"—Shared_Files is a component, \\dev\\myapp\\dlls is a path, and Is5.dll is a file name.</li> </ul>



**Note** • *InstallShield lets you add entire file and folder structures to your components by dragging and dropping them in the Files view.*

**Table 51 • FeatureFileInfo Parameters (cont.)**

Parameter	Description
<b>nInfo</b>	<p>Specifies the type of information to be retrieved. Pass a constant in this parameter. The available constants apply to either a specified component, or a specified file in a component.</p> <p>The available file-specific constants are:</p> <ul style="list-style-type: none"> <li>• <b>FEATURE_INFO_ATTRIBUTE</b>—Attribute values for the specified file. For more information about the <b>File</b> table attributes that are returned, see the Windows Installer Help. Only InstallScript MSI projects support this constant; InstallScript projects do not support it.</li> <li>• <b>FEATURE_INFO_DATE</b>—Date value for the specified file in the format mm-dd-yy. Only InstallScript projects support this constant; InstallScript MSI projects do not support it. If this constant is used with <b>FeatureFileInfo</b> in an InstallScript MSI installation, the function returns failure.</li> <li>• <b>FEATURE_INFO_DATE_EX</b>—Date value for the specified file in the 2000-compliant format mm-dd-yyyy. Only InstallScript projects support this constant; InstallScript MSI projects do not support it. If this constant is used with <b>FeatureFileInfo</b> in an InstallScript MSI installation, the function returns failure.</li> <li>• <b>FEATURE_INFO_TIME</b>—Time value for the specified file in the format hh:mm, using 24-hour time. Only InstallScript projects support this constant; InstallScript MSI projects do not support it. If this constant is used with <b>FeatureFileInfo</b> in an InstallScript MSI installation, the function returns failure.</li> <li>• <b>FEATURE_INFO_VERSIONLS</b>—The minor version number or numbers in string form.</li> <li>• <b>FEATURE_INFO_VERSIONMS</b>—The major version number or numbers in string form.</li> <li>• <b>FEATURE_INFO_VERSIONSTR</b>—The entire version number in string form.</li> <li>• <b>FEATURE_INFO_MD5_SIGNATURE</b>—Returns the MD5 signature of the file specified by szFile. Note that this information is generated during the media build, so calling this function with this parameter does not cause the MD5 signature to be generated.</li> </ul>



**Note** • Raw MD5 signature data for a particular file consists of 16 generated numeric values of 16 bit each (between 0x00 and 0xFF). These values are usually stored in a string of unsigned characters. However, the InstallScript language does not support unsigned characters, so instead of returning the raw MD5 file data, each of the 16 numeric values are converted to their string equivalent and placed in the resulting string. This results in a string of 32 characters, where each set of two characters represent a single numeric value. This is sometimes referred to as a MD5 hex string.

**Table 51** • FeatureFileInfo Parameters (cont.)

Parameter	Description
<b>nInfo (cont.)</b>	<ul style="list-style-type: none"> <li>● <b>FEATURE_INFO_ORIGSIZE_HIGH</b>—Returns the upper 31 bits of size of the original file. If the original size of the file is more than 2 GB, the size of the file is equal to the value returned for FEATURE_INFO_ORIGSIZE_HIGH multiplied by 2 GB added to the value returned for FEATURE_INFO_ORIGSIZE_LOW. Only InstallScript installations support this constant.</li> <li>● <b>FEATURE_INFO_ORIGSIZE_LOW</b>—Returns the lower 31 bits of size of the original file. If the original size of the file is more than 2 GB, the size of the file is equal to the value returned for FEATURE_INFO_ORIGSIZE_HIGH multiplied by 2 GB added to the value returned for FEATURE_INFO_ORIGSIZE_LOW.</li> <li>● <b>FEATURE_INFO_COMPsize_LOW</b>—Returns the lower 31 bits of size of the compressed file. If the compressed size of the file is more than 2 GB, the size of the file is equal to the value returned for FEATURE_INFO_COMPsize_HIGH multiplied by 2 GB added to the value returned for FEATURE_INFO_COMPsize_LOW.</li> <li>● <b>FEATURE_INFO_COMPsize_HIGH</b>—Returns the upper 31 bits of size of the compressed file. If the original size of the file is more than 2 GB, the size of the file is equal to the value returned for FEATURE_INFO_ORIGSIZE_HIGH multiplied by 2 GB added to the value returned for FEATURE_INFO_ORIGSIZE_LOW. Only InstallScript installations support this constant.</li> </ul>



**Note** • You can use the **ConvertSizeToUnits** function to convert these values into a single value of KBYTES, MBYTES, GBYTES, or TBYTES.

**Table 51 • FeatureFileInfo Parameters (cont.)**

Parameter	Description
<b>nInfo (cont.)</b>	<p>The available component-specific constants are:</p> <ul style="list-style-type: none"> <li>• <b>FEATURE_INFO_COMPONENT_FLAGS</b>—General flags that are set for the specified component. Only InstallScript projects support this constant; InstallScript MSI projects do not support it.</li> <li>• <b>FEATURE_INFO_DESTINATION</b>—Destination setting for the component. Note that any text substitutions that are included in the destination are not resolved in the returned string. To determine the component's destination with any text substitutions resolved, call <b>TextSubSubstitute</b> on the returned string.</li> <li>• <b>FEATURE_INFO_DOTNET</b>—.NET flags that are selected for the specified component. Only InstallScript projects support this constant; InstallScript MSI projects do not support it.</li> <li>• <b>FEATURE_INFO_FTPLOCATION</b>—FTP Location setting for the specified component. Only InstallScript projects support this constant; InstallScript MSI projects do not support it.</li> <li>• <b>FEATURE_INFO_HTTPLOCATION</b>—HTTP Location setting for the specified component. Only InstallScript projects support this constant; InstallScript MSI projects do not support it.</li> <li>• <b>FEATURE_INFO_LANGUAGE</b>—Languages setting for the specified component. Because this applies only to a component, szFile must be a single token expression identifying a component.</li> <li>• <b>FEATURE_INFO_MISC</b>—Miscellaneous setting for the specified component. Only InstallScript projects support this constant; InstallScript MSI projects do not support it.</li> <li>• <b>FEATURE_INFO_OS</b>—Operating System setting for the specified component. Because this applies only to a component, szFile must be a single token expression that identifies a component. Only InstallScript projects support this constant; InstallScript MSI projects do not support it.</li> <li>• <b>FEATURE_INFO_OVERWRITE</b>—Overwrite flags that are set for the specified component. Only InstallScript projects support this constant; InstallScript MSI projects do not support it.</li> <li>• <b>FEATURE_INFO_PLATFORM_SUITE</b>—Platform Suites setting for the specified component. Because this applies only to a component, szFile must be a single token expression that identifies a component. Only InstallScript projects support this constant; InstallScript MSI projects do not support it.</li> </ul>

**Table 51 • FeatureFileInfo Parameters (cont.)**

Parameter	Description
<b>nvResult</b>	<p>Specifies information that is dependent on the constant that is passed in nInfo.</p> <p>When nInfo is FEATURE_INFO_ORIGSIZE_HIGH, FEATURE_INFO_ORIGSIZE_LOW, or FEATURE_INFO_ATTRIBUTE, nvResult returns a number value.</p> <p>When nInfo is FEATURE_INFO_ATTRIBUTE, nvResult returns a numeric value that represents the file attributes from the Windows Installer <b>File</b> table. To determine the file attribute settings, use file attribute constants in bitwise OR operations with nvResult. For an example of how to test nvResult, see <a href="#">FeatureFileInfo Example</a>. For more information about the File table attributes, see the Windows Installer Help.</p> <p>When nInfo is FEATURE_INFO_LANGUAGE or FEATURE_INFO_OS, the parameter position occupied by nvResult must specify a LIST variable to store the language or operating system IDs. You must call <a href="#">ListCreate</a> with the constant NUMBERLIST to create the list before you call <b>FeatureFileInfo</b> to determine which languages or operating systems are specified by the component:</p> <pre>listID = ListCreate(NUMBERLIST);  FeatureFileInfo(szFeatureSource, szFeature, szFile, FEATURE_INFO_LANGUAGE, listID, svResult);</pre> <p>When nInfo is FEATURE_INFO_LANGUAGE, <b>FeatureFileInfo</b> adds language IDs to the list; when nInfo is FEATURE_INFO_OS, <b>FeatureFileInfo</b> adds operating system IDs to the list.</p> <p>When nInfo is FEATURE_INFO_COMPONENT_FLAGS, nvResult returns one or more of the following values:</p> <ul style="list-style-type: none"> <li>● FEATURE_INFO_COMPONENT_FLAG_COMPRESSED—The component's Compressed setting is Yes.</li> <li>● FEATURE_INFO_COMPONENT_FLAG_DATAASFILES—The component's files are placed in a particular folder for a CD-ROM type of release. (This is determined by the CD-ROM Folder setting for the feature that contains the component.) Only InstallScript projects support this constant; InstallScript MSI projects do not support it.</li> <li>● FEATURE_INFO_COMPONENT_FLAG_DONTUNINSTALL—The component's Uninstall setting is No.</li> <li>● FEATURE_INFO_COMPONENT_FLAG_ENCRYPTED—The component is encrypted. (This is determined by whether the feature that includes the component has Yes selected for its Encryption setting.)</li> <li>● FEATURE_INFO_COMPONENT_FLAG_POTENTIALLYLOCKED—The component's Potentially Locked setting is Yes.</li> <li>● FEATURE_INFO_COMPONENT_FLAG_SELFREGISTERING—The component's Self-Register setting is Yes.</li> <li>● FEATURE_INFO_COMPONENT_FLAG_SHARED—The component's Shared setting is Yes.</li> </ul>

**Table 51** • FeatureFileInfo Parameters (cont.)

Parameter	Description
<b>nvResult (cont.)</b>	<p>When ninfo is FEATURE_INFO_OVERWRITE, nvResult returns one or more of the following values:</p> <ul style="list-style-type: none"> <li>• FEATURE_VALUE_ALWAYS—The component's Overwrite setting is Always.</li> <li>• FEATURE_VALUE_DATE_NEWER—The component's Overwrite setting is Newer Date.</li> <li>• FEATURE_VALUE_DATE_OLDER—The component's Overwrite setting is Older Date.</li> <li>• FEATURE_VALUE_DATE_SAME—The component's Overwrite setting is Same or Newer Date.</li> <li>• FEATURE_VALUE_NEVER—The component's Overwrite setting is Never.</li> <li>• FEATURE_VALUE_VERSION_NEWER—The component's Overwrite setting is Newer Version.</li> <li>• FEATURE_VALUE_VERSION_OLDER—The component's Overwrite setting is Older Version.</li> <li>• FEATURE_VALUE_VERSION_SAME—The component's Overwrite setting is Same or Newer Version.</li> </ul> <p>Note that nvResult could contain both date and version values.</p> <p>When ninfo is FEATURE_INFO_DOTNET, nvResult may return the following value:</p> <p>FEATURE_VALUE_LOCAL_ASSEMBLY—The component is marked as a local assembly.</p>
<b>svResult</b>	Returns the date, time, or version when nInfo specifies a corresponding information type.

## Return Values

**Table 52** • FeatureFileInfo Return Values

Return Value	Description
<b>0</b>	FeatureFileInfo was successful.
<b>&lt;0</b>	FeatureFileInfo failed.

## FeatureFileInfo Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the FeatureFileInfo function.

```



```

*
* This script calls FeatureFileInfo repeatedly to retrieve
* information about the file media. The information is stored
* in a list and then displayed.
*
* Note: To run this example script, create a project with the
*       following features (f), subfeatures (sf), and
*       components (c):
*
*       (f) Example_Files
*       (sf) Graphics
*       (c) Graphic_Examples
*
*       Be sure to assign at least one file to the Graphic_Examples
* component so there is something to enumerate.
*       Also, remember to specify the file name in the #define
*       FILE line.
*
\*-----*/

#define FEAT          "Example Files"
#define SUBFEAT       "Graphics"
#define COMP          "Graphic Examples"
#define FILE          "comdlg32.dll"
#define SDSHOWTITLE   "ComponentFileInfo Results"
#define SDSHOWMSG      FILE + " information:"

prototype HandleFeatureError (NUMBER);

NUMBER nReturn, nvResult;
STRING svResult;
LIST listID;

program

    // Create a list to store file media information.
    listID = ListCreate (STRINGLIST);

    // Get the original size of the specified file.
    nReturn = FeatureFileInfo (MEDIA, FEAT + "\\\" + SUBFEAT,
                              COMP + "\\\" + FILE,
                              FEATURE_INFO_ORIGSIZE , nvResult, svResult);

    HandleFeatureError (nvResult);

    // Convert the original size to a string.
    Sprintf (svResult, "%d", nvResult);

    // Add the string to the list.
    ListAddString (listID, "The Original size of the file is " +
                    svResult, AFTER);

    // Get the attributes of the specified file.
    nReturn = FeatureFileInfo (MEDIA, FEAT + "\\\" + SUBFEAT,
                              COMP + "\\\" + FILE,
                              FEATURE_INFO_ATTRIBUTE , nvResult, svResult);

    HandleFeatureError (nReturn);

```

```

// If no attributes are set, indicate normal attributes.
if (nvResult = FILE_ATTR_NORMAL) then
    svResult = "normal";

// If attributes are set, concatenate them for display.

else
    if (FILE_ATTR_ARCHIVED & nvResult) then
        svResult = "archived,";
    endif;

    if (FILE_ATTR_HIDDEN & nvResult) then
        svResult = svResult + "hidden,";
    endif;

    if (FILE_ATTR_READONLY & nvResult) then
        svResult = svResult + "read-only,";
    endif;

    if (FILE_ATTR_SYSTEM & nvResult) then
        svResult = svResult + "system,";
    endif;

    if (FILE_ATTR_DIRECTORY & nvResult) then
        svResult = svResult + "directory,";
    endif;

endif;

// Add the string of attributes to the list.
ListAddString (listID, "The attribute for the file is " +
    svResult, AFTER);

// Get the major file version of the specified file.
nReturn = FeatureFileInfo (MEDIA, FEAT + "\\" + SUBFEAT,
    COMP + "\\" + FILE,
    FEATURE_INFO_VERSIONMS , nvResult, svResult);
HandleFeatureError (nReturn);

// Add the major file version to the list.
ListAddString (listID, "The upper 32-bit version value " +
    svResult, AFTER);

// Get the minor file version of the specified file.
nReturn = FeatureFileInfo (MEDIA, FEAT + "\\" + SUBFEAT,
    COMP + "\\" + FILE,
    FEATURE_INFO_VERSIONLS , nvResult, svResult);
HandleFeatureError (nReturn);

// Add the minor file version to the list.

ListAddString (listID, "The lower 32-bit version value is " +
    svResult, AFTER);

// Get the complete file version of the specified file.
nReturn = FeatureFileInfo (MEDIA, FEAT + "\\" + SUBFEAT,
    COMP + "\\" + FILE,
    FEATURE_INFO_VERSIONSTR , nvResult, svResult);

```

```

    HandleFeatureError (nReturn);

    // Add the complete file version to the list.
    ListAddString (listID, "The version for the file is          " +

                    svResult, AFTER);

    // Display the list.
    SdShowInfoList (SDSHOWTITLE, SDSHOWMSG, listID);

    // Release the list from memory.
    ListDestroy(listID);

endprogram

/*-----*\
 *
 * Function:  HandleFeatureError
 *
 * Purpose:  This function evaluates the value returned by a feature
 *           function and if the value is less than zero, displays the error
 *           number and aborts the setup.
 *
 *-----*/
function HandleFeatureError (nvResult)

    NUMBER  nvError;
    STRING  svFeatureSource, svFeature, svComponent, svFile;

begin
    if (nvResult < 0) then
        FeatureError(svFeatureSource, svFeature, svComponent, svFile, nvError);
        SprintfBox(INFORMATION, "Data Transfer Error Information",
                    "FeatureError returned the following data transfer error.\n" +

                    "Setup will now abort.\n\n" +
                    "Media Name: %s\nFeature: %s\nComponent: %s\n" +
                    "File: %s\nError Number: %ld",
                    svFeatureSource, svFeature, svComponent, svFile, nvError);

        abort;
    endif;
end;

```

## FeatureFilterLanguage



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **FeatureFilterLanguage** function filters (excludes) files from file transfer based on language. By default, all languages included in the media build are unfiltered (included). You must call FeatureFilterLanguage for each language you wish to filter or unfilter.



**Note** • This function cannot be used with script-created feature sets.

## Filtering Language-Specific Components



### Task

#### To filter language-specific components during the installation:

1. Filter (exclude) all languages by calling `FeatureFilterLanguage` with `ISLANG_ALL` in the parameter `nLangID` and `bFiltered` set to `TRUE`.
2. For each language that you want to install, call `FeatureFilterLanguage` with the appropriate language constant in `nLangID` and with the parameter `bFiltered` parameter set to `FALSE`. Each call unfilters (includes) components for the language specified in `nLangID`.



**Note** • You cannot specify multiple language constants in the `nLangID` parameter by using the OR operator (`|`). Specifying multiple language constants causes the function to perform incorrectly.

## Supporting Different Languages

InstallShield allows you to designate components for any language or language subgroup that is supported by Windows. However, in order for the Release Wizard to build a language-specific component, you must have support for the language of that component. Your setup must also support the language of the component.

If your setup includes language-specific components that are designated as specific to a language not supported by InstallShield or by your setup, the components are filtered (not included) by the Release Wizard.

## Using FeatureFilterLanguage With GetSystemInfo

When using `FeatureFilterLanguage` in conjunction with the [GetSystemInfo](#) function, you must consider the following: The language constants that can be used to designate language-specific components are a small subset of the language constants that can be returned by `GetSystemInfo`.

If your setup includes language filtering based on these return values, you must use a switch statement to convert constants returned by this function into one of the constants supported for language filtering.

## Syntax

```
FeatureFilterLanguage ( szFeatureSource, nLangID, bFiltered );
```

## Parameters

**Table 53** • FeatureFilterLanguage Parameters

Parameter	Description
<b>szFeatureSource</b>	Specifies the media name of the file media.
<b>nLangID</b>	Specifies the ID of the language to filter or unfilter. Only <i>one</i> language constant can be specified for each function call. To filter all languages, pass ISLANG_ALL in this parameter. See <a href="#">Language IDs</a> to learn more.
<b>bFiltered</b>	Indicates whether to filter (exclude) or unfilter (include) the language specified in nLangID. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>TRUE</b>—Filter the language specified in nLangID—do not include it in the file transfer.</li> <li>• <b>FALSE</b>—Do not filter the language specified in nLangID—include it in the file transfer.</li> </ul>

## Return Values

**Table 54** • FeatureFilterLanguage Return Values

Return Value	Description
<b>0</b>	FeatureFilterLanguage was successful.
<b>&lt; 0</b>	FeatureFilterLanguage failed.

## FeatureFilterLanguage Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the FeatureFilterLanguage function.
 *
 * First, FeatureFilterLanguage is called to exclude all
 * languages. Next, GetSystemInfo is called to determine the
 * target computer's default language/locale.
 *
 * Then, FeatureFilterLanguage is called again to include the
 * language appropriate for the target computer. If language
 * support is not provided for the target computer, English is
 * used. Finally, the FeatureMoveData is called to create the
 * installation.
 *
 \*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

```

```

export prototype ExFn_FeatureFilterLanguage(HWND);

function ExFn_FeatureFilterLanguage(hMSI)
    STRING  szResult;
    NUMBER  nResult, nDisk;
begin

    // Filter out all language-specific file groups.
    FeatureFilterLanguage (MEDIA, ISLANG_ALL, TRUE);

    // Retrieve the target machine's default language/locale setting.
    GetSystemInfo (LANGUAGE, nResult, szResult);

    // Turn off filtering for file groups specific to the target
    // machine's default language/locale setting.

    switch (nResult)
        case ISLANG_FRENCH_CANADIAN:
            FeatureFilterLanguage (MEDIA, ISLANG_FRENCH_CANADIAN, FALSE);
        case ISLANG_FRENCH_STANDARD, ISLANG_FRENCH_BELGIAN, ISLANG_FRENCH_SWISS,
ISLANG_FRENCH_LUXEMBOURG:
            FeatureFilterLanguage (MEDIA, ISLANG_FRENCH_STANDARD, FALSE );
        case ISLANG_GERMAN_STANDARD, ISLANG_GERMAN_SWISS, ISLANG_GERMAN_AUSTRIAN,
ISLANG_GERMAN_LUXEMBOURG, ISLANG_GERMAN_LIECHTENSTEIN:
            FeatureFilterLanguage (MEDIA, ISLANG_GERMAN, FALSE);
        // Use English as a default.
        default:
            FeatureFilterLanguage (MEDIA, ISLANG_ENGLISH_UNITEDSTATES, FALSE);
    endswitch;

    // Transfer files for selected language.
    FeatureMoveData (MEDIA, nDisk, 0);

end;

```

## FeatureFilterOS



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **FeatureFilterOS** function filters components that are flagged for specified operating systems and suites.



**Note** • This function cannot be used with script-created feature sets.

Any filtering that is done when the installation is first run must also be done during maintenance mode. Be sure to call this function in code that is executed during both initial and maintenance installations. Do not call this function from the following event handlers: OnAppSearch, OnCCPSearch, OnFirstUIBefore, OnFirstUIAfter, OnMaintUIBefore, or OnMaintUIAfter.



---

**Project** • Components are not filtered by default in an InstallScript MSI installation.



In an InstallScript installation, **FeatureFilterOS** is called by the default *OnFilterComponents* event handler during both initial and maintenance installations. This event handler filters components by default that are specific to operating systems and suites other than those on the target system. An installation that is run in maintenance mode has no information about the filtering that was done during the initial installation.

## Syntax

```
FeatureFilterOS ( szMediaLibrary, nSuites, nOS, bFiltered );
```


## Parameters

**Table 55 • FeatureFilterOS Parameters**

Parameter	Description
<b>szMediaLibrary</b>	Specifies the media name of the file media library.
<b>nSuites</b>	<p>Specifies the operating system suite or suites that you want to filter. Choose from the following values. You can combine values using the bitwise OR operator ( ).</p> <div>  <p><b>Note •</b> The suites listed here are those that can be specified in the Windows API's OSVERSIONINFOEX data structure.</p> <ul style="list-style-type: none"> <li>• <b>ISOS_ST_ALL</b>—All Windows suites</li> <li>• <b>ISOS_ST_XP_PRO</b>—Windows XP Professional</li> <li>• <b>ISOS_ST_XP_HOME</b>—Windows XP Home Edition</li> <li>• <b>ISOS_ST_SERVER</b>—Windows Server</li> <li>• <b>ISOS_ST_WORKSTATION</b>—Windows Workstation</li> <li>• <b>ISOS_ST_BACKOFFICE</b>—Microsoft BackOffice</li> <li>• <b>ISOS_ST_DATACENTER</b>—Windows Server Datacenter</li> <li>• <b>ISOS_ST_ENTERPRISE</b>—Windows Server Enterprise</li> <li>• <b>ISOS_ST_SERVER2003_R2</b>—Microsoft Windows Server 2003 R2</li> <li>• <b>ISOS_ST_SMALLBUSINESS</b>—Microsoft Small Business Server</li> <li>• <b>ISOS_ST_SMALLBUSINESS_RESTRICED</b>—Microsoft Small Business Server with the restrictive client license</li> <li>• <b>ISOS_ST_TERMINAL</b>—Microsoft Terminal Services</li> <li>• <b>ISOS_ST_PROC_ARCH_32</b>—32-bit Processors</li> <li>• <b>ISOS_ST_PROC_ARCH_IA64</b>—Intel Itanium 64-bit Processor</li> <li>• <b>ISOS_ST_PROC_ARCH_AMD64</b>—AMD Athalon 64-bit Processor</li> <li>• <b>0 (zero)</b>—Specifies that FeatureFilterOS ignores components' Platform Suite properties</li> </ul> </div> <div>  <p><b>Project •</b> InstallScript MSI projects do not have support for platform suites. Therefore, in an InstallScript MSI project, you must specify the number 0 for nSuites. Otherwise, the function fails and the information that is specified in nOS is ignored.</p> </div>



**Table 55 • FeatureFilterOS Parameters (cont.)**

Parameter	Description
<b>nOS</b>	<p>Specifies the operating system or operating systems that you wish to filter. Choose from the following values. You can combine values using the bitwise OR operator (   ).</p> <ul style="list-style-type: none"> <li>• <b>ISOSL_ALL</b>—All Windows systems</li> <li>• <b>ISOSL_WINXP</b>—Windows XP Edition</li> <li>• <b>ISOSL_WINSERVER2003</b>—Windows Server 2003</li> <li>• <b>ISOSL_WINVISTA_SERVER2008 (or ISOSL_WINVISTA)</b>—Windows Vista or Windows Server 2008</li> <li>• <b>ISOSL_WIN7_SERVER2008R2</b>—Windows 7 or Windows Server 2008 R2</li> <li>• <b>ISOSL_WIN8</b>—Windows 8 or Windows Server 2012</li> <li>• <b>ISOSL_WIN81</b>—Windows 8.1 or Windows Server 2012 R2</li> <li>• <b>ISOSL_WIN10</b>—Windows 10</li> </ul> <p></p> <p><b>Note</b> • Several client and server versions of Windows use the same major and minor version numbers:</p> <ul style="list-style-type: none"> <li>• Windows 8.1 and Windows Server 2012 R2 use the same major and minor version numbers.</li> <li>• Windows 8 and Windows Server 2012 use the same major and minor version numbers.</li> <li>• Windows 7 and Windows Server 2008 R2 use the same major and minor version numbers.</li> <li>• Windows Vista and Windows Server 2008 have the same major and minor version numbers.</li> </ul> <p>Therefore, for these operating system versions, the installation considers the client versions to be the same as the equivalent server versions; thus, components that are marked for the client version are also installed on the server version. To distinguish between the client and server versions, you can check whether <code>SYSINFO.nOSProductType</code> is equal to <code>VER_NT_WORKSTATION</code>; on client versions, this is true. On server versions, this is false.</p>
<b>bFiltered</b>	<p>Specifies whether to filter (exclude) or unfilter (include) the operating systems specified in <code>nOS</code>. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>TRUE</b>—Filter the specified operating system—do not include them in the file transfer.</li> <li>• <b>FALSE</b>—Do not filter the specified operating system—include them in the file transfer.</li> </ul>

## Return Values

**Table 56 • FeatureFilterOS Return Values**

Return Value	Description
<b>0</b>	<b>FeatureFilterOS</b> was successful.
<b>&lt; 0</b>	<b>FeatureFilterOS</b> failed. Call <a href="#">FeatureError</a> for additional information.

## Additional Information

While both ISOSL\_ALL and ISOSL\_SUPPORTED specify all Windows systems when passed as the third argument to **FeatureFilterOS**, they have different numeric values. ISOSL\_ALL is zero (0) and ISOSL\_SUPPORTED is the value obtained by combining every operating system constant using the bitwise OR operator, that is, ISOSL\_WIN7\_SERVER2008R2 | ISOSL\_WINVISTA\_SERVER2008 | ... . In some cases, you may find it useful to perform bitwise operations on ISOSL\_SUPPORTED and other operating system constants.

## FeatureFilterOS Example

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the FeatureFilterOS function.
 *
 * To run this script, create a blank setup project.
 * Specify Windows 95, Windows 98, and Windows NT 4.0
 * (Intel) for operating systems in the Components view.
 * Use release flags to specify which platforms to include
 * in the build. Specify Windows 95, Windows 98, and Windows
 * NT 4.0 (Intel). When you run the Release Wizard, enter the
 * release flags in the Filtering Settings panel.
 *
 \*-----*/

// You can convert the following define statements into string
// entries to localize your setup.
#define COMPANY_NAME      "MultiLangOS Inc."
#define PRODUCT_NAME      "MultiLangOS"
#define PRODUCT_VERSION   "1.0"
#define PROGRAMFOLDER     "MultiLangOS"
#define PRODUCT_KEY       "Mlangos.exe"
#define DEINST_KEY        "MultiLangOS"
#define ASKDESTITLE       "Destination Location"
#define ASKDESTMMSG       "Select a destination location."
#define COMPERRTITLE       "Data Transfer Error Information"
#define COMPERRMSG1        "FeatureError returned the following error."
#define COMPERRMSG2        "Setup will now abort."
#define COMPERRMSG3        "Media Name:"
#define COMPERRMSG4        "Feature:"
#define COMPERRMSG5        "Component:"
#define COMPERRMSG6        "File:"
#define COMPERRMSG7        "Error Number:"
```

```

    prototype HandleFeatureError (NUMBER);

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_FeatureFilterOS(HWND);

function ExFn_FeatureFilterOS(hMSI)
    STRING  svResult, svDir, svLogFile;
    NUMBER  nvResult, nvDisk;
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Set up uninstallation and get destination location.
    InstallationInfo(COMPANY_NAME, PRODUCT_NAME, PRODUCT_VERSION, PRODUCT_KEY);

    INSTALLDIR = PROGRAMFILES ^ PROGRAMFOLDER;

    AskDestPath (ASKDESTITLE, ASKDESTMMSG, INSTALLDIR , 0 );

    DeinstallStart (INSTALLDIR, svLogFile, DEINST_KEY, 0);

    RegDBSetItem (REGDB_UNINSTALL_NAME, DEINST_KEY);

    // Get the operating system and call FeatureFilterOS
    // to filter the operating systems that are not present.
    GetSystemInfo (OS, nvResult, svResult);

    switch (nvResult)
    case IS_WINDOWSNT:
        GetSystemInfo (WINMAJOR, nvResult, svResult);
        if (nvResult = 4) then
            // It's NT 4.0, so filter Windows 95 and Windows 98.
            FeatureFilterOS (MEDIA, 0, ISOSL_WIN95, TRUE);
            FeatureFilterOS (MEDIA, 0, ISOSL_WIN98, TRUE);
        else
            MessageBox ("Target system OS not supported.", SEVERE);
            abort;
        endif;
    case IS_WINDOWS9X:
        // It's Windows 95 or 98, so filter Windows NT 4.0.
        FeatureFilterOS (MEDIA, 0, ISOSL_NT40, TRUE);

        // Determine if it's Windows 95 or Windows 98.
        GetSystemInfo (WINMINOR, nvResult, svResult);

        if (nvResult < 10) then
            // It's Windows 95, so filter Windows 98 too.
            FeatureFilterOS (MEDIA, 0, ISOSL_WIN98, TRUE);
        else
            // It's Windows 98, so filter Windows 95 too.
            FeatureFilterOS (MEDIA, 0, ISOSL_WIN95, TRUE);
        endif;
    default:
        MessageBox ("Target system OS not supported.", SEVERE);
        abort;
    endswitch;
endfunction;

```

```

// Transfer files to target system.
nvResult = FeatureMoveData (MEDIA, nvDisk, 0);

if (nvResult < 0) then
    HandleFeatureError (nvResult);
endif;

end;

/*-----*\
*
* Function:  HandleFeatureError
*
* Purpose:  This function evaluates the value returned by a feature
*           function and if the value is less than zero, displays the error
*           number and aborts the setup.
*
\*-----*/
function HandleFeatureError( nResult )
    NUMBER  nvError;
    STRING  svFeatureSource, svFeature, svComponent, svFile;
begin
    FeatureError (svFeatureSource, svFeature, svComponent, svFile, nvError);
    sprintfBox(INFORMATION, FEATERRTITLE,
        FEATERRMSG1 + "\n" + FEATERRMSG2 + "\n\n" +
        FEATERRMSG3 + " %s\n" + FEATERRMSG4 + " %s\n" +
        FEATERRMSG5 + " %s\n" + FEATERRMSG6 + " %s\n" +
        FEATERRMSG7 + " %ld",
        svFeatureSource, svFeature, svComponent, svFile, nvError);
    abort;
end;

```

## FeatureGetCost



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **FeatureGetCost** function is obsolete. Use the [FeatureGetCostEx](#) function instead.

The **FeatureGetCost** function determines the total space, in kilobytes (KB), required on the target drive for the feature that is specified by `szFeature`. When determining required drive space, this function takes into account whether the feature is currently selected, whether any components associated with the feature are currently filtered by operating system or language, and the cluster size on the target drive.

### Syntax

```
FeatureGetCost ( szFeatureSource, szFeature, szTargetDir, nvRequiredSpace );
```

## Parameters

**Table 57** • FeatureGetCost Parameters

Parameter	Description
<b>szFeatureSource</b>	Specifies the media name of the file media whose features have been specified for installation and uninstallation. Typically, you would pass the system variable <a href="#">MEDIA</a> in this parameter.
<b>szFeature</b>	Specifies the feature for which you want to determine the total space that is required on the target drive.
<b>szTargetDir</b>	Specifies the drive that is used, or the path whose drive is used, in determining required drive space. Typically, you would pass the system variable <a href="#">TARGETDIR</a> in this parameter.
<b>nvRequiredSpace</b>	Returns the required drive space, in KB.

## Return Values

**Table 58** • FeatureGetCost Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully determined the required drive space.
<b>&lt; 0</b>	Indicates that the function could not determine the required drive space.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

## FeatureGetCost Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the FeatureGetCost function.
 *
 * This example script displays a dialog that shows the user
 * the size of each top-level feature that they selected.
 *
 * Comments: To run this example script, create a project (or
 *           insert into a project) with several features
 *           and/or subfeatures with components containing
 *           files.
 *
 \*-----*/

#include "Ifx.h"

function OnBegin()

```

```

string svDir, svFeature, svFeatureInfo;
LIST listFeatures, listFeatureInfo;
number nListGetString, nvRequiredSpace;
begin
    svDir = TARGETDIR;
    SdFeatureTree ("", "", svDir, "", 1);

    listFeatures = ListCreate ( STRINGLIST );
    FeatureListItems ( MEDIA, "", listFeatures );
    nListGetString = ListGetFirstString ( listFeatures, svFeature );
    listFeatureInfo = ListCreate ( STRINGLIST );
    while nListGetString=0
        FeatureGetCost ( MEDIA, svFeature, svDir, nvRequiredSpace );
        Sprintf ( svFeatureInfo, "Selected feature %s has a size of %ld KB.\n",
            svFeature, nvRequiredSpace );
        ListAddString ( listFeatureInfo, svFeatureInfo, AFTER );
        nListGetString = ListGetNextString ( listFeatures, svFeature );
    endwhile;
    ListDestroy ( listFeatures );

    SdShowInfoList ( "", "", listFeatureInfo );
    ListDestroy ( listFeatureInfo );
end;

```

## FeatureGetCostEx



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **FeatureGetCostEx** function returns the cost of the specified feature (in bytes) using the `nvCostHigh` and `nvCostLow` parameters. If `szFeature` is "", the cost of the entire media is returned. This function replaces the **FeatureGetCost** and **FeatureGetTotalCost** functions.

### Syntax

```
FeatureGetCostEx ( szMediaSource, szFeature, szTarget, nvCostHigh, nvCostLow );
```

## Parameters

**Table 59** • FeatureGetCostEx Parameters

Parameter	Description
<b>szMediaSource</b>	The media source, typically MEDIA.
<b>szFeature</b>	Specifies the feature to get the cost of. If this value is "", the function returns the cost of the entire media.
<b>szTarget</b>	Specifies the target directory for the feature specified by szFeature.
<b>nvCostHigh</b>	Returns the upper 31 bits of the cost of the feature. Each cost unit returned represents 2GB of cost. For example, a value of 4 for this variable represents a cost of 8GB.
<b>nvCostLow</b>	Returns the lower 31 bits of the cost of the feature. The maximum value for this variable is 1 or 2GB of cost. Any cost greater than this is returned in the nvCostHigh variable.

## Return Values

**Table 60** • FeatureGetCostEx Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	Indicates that the function was successful.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was not successful.

# FeatureGetData



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **FeatureGetData** function retrieves information about a feature.

## Syntax

```
FeatureGetData ( szFeatureSource, szFeature, nInfo, nvResult, svResult );
```

## Parameters

**Table 61 • FeatureGetData Parameters**

Parameter	Description
<b>szFeatureSource</b>	Specifies the media name of the script-created feature set or (in InstallScript projects) file media library that contains the feature for which information is retrieved.
<b>szFeature</b>	Specifies the name of the feature from which information is retrieved. To learn how to refer to top-level features and subfeatures, see <a href="#">Specifying Features and Subfeatures in Function Calls</a> .
<b>nInfo</b>	<p>Specifies the type of information to retrieve. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>FEATURE_FIELD_CDROM_FOLDER</b>—For a CD-ROM media type, the location of the feature's data on the CD-ROM. Only InstallScript projects support this constant; InstallScript MSI projects do not. This constant is available for file media libraries, not for script-created feature sets.</li> <li>• <b>FEATURE_FIELD_DESCRIPTION</b>—The description displayed when the feature is selected in a feature selection dialog. This is the text from the feature's Description property.</li> <li>• <b>FEATURE_FIELD_DISPLAYNAME</b>—The feature name displayed in the selection dialogs. This is the value in the feature's Display Name setting.</li> <li>• <b>FEATURE_FIELD_ENCRYPT</b>—Specifies whether the feature is encrypted. Only InstallScript projects support this constant; InstallScript MSI projects do not. This constant is available for file media libraries, not for script-created feature sets.</li> <li>• <b>FEATURE_FIELD_FILENEED</b>—Defines how critical the feature files are for the installation. Only InstallScript projects support this constant; InstallScript MSI projects do not.</li> <li>• <b>FEATURE_FIELD_FLAGS</b>—Indicates the flags that are set for the feature. Only InstallScript projects support this constant; InstallScript MSI projects do not. This constant is available for file media libraries, not for script-created feature sets.</li> <li>• <b>FEATURE_FIELD_FTPLOCATION</b>—Specifies an FTP location. This value is stored in the feature's FTP Location setting.</li> <li>• <b>FEATURE_FIELD_GUID</b>—The GUID that is associated with the feature. Only InstallScript projects support this constant; InstallScript MSI projects do not. This constant is available for file media libraries, not for script-created feature sets.</li> </ul>



**Table 61** • FeatureGetData Parameters (cont.)

Parameter	Description
<b>nInfo (cont.)</b>	<ul style="list-style-type: none"> <li>• <b>FEATURE_FIELD_HANDLER_ONINSTALLED</b>—The name of the OnInstalled event for the feature. This constant is available for file media libraries, not for script-created feature sets.</li> <li>• <b>FEATURE_FIELD_HANDLER_ONINSTALLING</b>—The name of the OnInstalling event for the feature. This constant is available for file media libraries, not for script-created feature sets.</li> <li>• <b>FEATURE_FIELD_HANDLER_ONUNINSTALLED</b>—The name of the OnUninstalled event for the feature. This constant is available for file media libraries, not for script-created feature sets.</li> <li>• <b>FEATURE_FIELD_HANDLER_ONUNINSTALLING</b>—The name of the OnUninstalling event for the feature. This constant is available for file media libraries, not for script-created feature sets.</li> <li>• <b>FEATURE_FIELD_HTTPLOCATION</b>—Specifies an HTTP location. This value is stored in the feature's HTTP Location setting.</li> <li>• <b>FEATURE_FIELD_IMAGE</b>—Specifies the index of the icon to be displayed in nvResult. If no icon should be displayed for the feature, nvResult returns -1.</li> <li>• <b>FEATURE_FIELD_MISC</b>—Miscellaneous text. This field can be useful at run time because you can use it to flag or identify features using any information you want.</li> <li>• <b>FEATURE_FIELD_PASSWORD</b>—Specifies whether a password is associated with the feature. This constant is available for file media libraries, not for script-created feature sets. Only InstallScript projects support this constant; InstallScript MSI projects do not.</li> <li>• <b>FEATURE_FIELD_SELECTED</b>—Indicates whether the feature specified in szFeature is selected.</li> <li>• <b>FEATURE_FIELD_SIZE</b>—Total original file size for the feature specified in szFeature. This constant is available for file media libraries, not for script-created feature sets.</li> <li>• <b>FEATURE_FIELD_STATUS</b>—In InstallScript installations, this text is displayed in the progress indicator during file transfer. This constant is available for file media libraries, not for script-created feature sets. In InstallScript MSI installations, the feature's display name is returned.</li> <li>• <b>FEATURE_FIELD_VISIBLE</b>—Determines whether the feature that is specified in szFeature is visible in the selection dialog. This value is stored in the feature's Display setting.</li> </ul>

**Table 61** • FeatureGetData Parameters (cont.)

Parameter	Description
<b>nvResult</b>	<p>Returns a numeric value when nInfo produces a numeric result.</p> <p>When nInfo is FEATURE_FIELD_FILENEED, nvResult returns one of the following values:</p> <ul style="list-style-type: none"> <li>● <b>FEATURE_VALUE_CRITICAL</b>—This feature contains critical files.</li> <li>● <b>FEATURE_VALUE_HIGHLYRECOMMENDED</b>—This feature is highly recommended.</li> <li>● <b>FEATURE_VALUE_STANDARD</b>—This feature may or may not be included.</li> </ul> <p>When nInfo is FEATURE_FIELD_FLAGS, nvResult returns one of the following values:</p> <ul style="list-style-type: none"> <li>● <b>FEATURE_DATA_FLAG_PASSWORD</b>—The feature is password protected.</li> <li>● <b>FEATURE_DATA_FLAG_PASSWORD_VALIDATED</b>—The feature's password has been validated. (The installation is using <b>FeatureValidate</b> or the feature is not password protected.)</li> <li>● <b>FEATURE_DATA_FLAG_DATA_AS_FILES</b>—The feature's files are placed in a particular folder for a CD-ROM type of release.</li> <li>● <b>FEATURE_DATA_FLAG_SPLIT_AFTER</b>—This flag is set only for build-generated features.</li> <li>● <b>FEATURE_DATA_FLAG_SPLIT_BEFORE</b>—This flag is set only for build-generated features.</li> <li>● <b>FEATURE_DATA_FLAG_SPLIT_BEFORE_NOT_ALLOWED</b>—This flag is set only for build-generated features.</li> <li>● <b>FEATURE_DATA_FLAG_SPLIT_NOT_ALLOWED</b>—This flag is set only for build-generated features.</li> <li>● <b>FEATURE_DATA_FLAG_VISIBLE</b>—The feature is visible.</li> <li>● <b>FEATURE_DATA_FLAG_VOLATILE</b>—The feature is volatile.</li> <li>● <b>FEATURE_DATA_FLAG_ENCRYPTED</b>—The feature is encrypted.</li> </ul>

**Table 61** • FeatureGetData Parameters (cont.)

Parameter	Description
	<p>When <code>nInfo</code> is <code>FEATURE_FIELD_PASSWORD</code>, <code>nvResult</code> returns one of the following values:</p> <ul style="list-style-type: none"> <li>• <b>TRUE</b>—The feature is password protected. If the feature is password protected, you must get the correct password from the end user and validate it with <b>FeatureValidate</b> before calling <b>FeatureTransferData</b> to transfer the files in the file media library.</li> <li>• <b>FALSE</b>—The feature is not password protected.</li> </ul> <p>When <code>nInfo</code> is <code>FEATURE_FIELD_SELECTED</code>, <code>nvResult</code> returns one of the following values:</p> <ul style="list-style-type: none"> <li>• <b>TRUE</b>—This feature is selected.</li> <li>• <b>FALSE</b>—This feature is not selected.</li> </ul> <p>When <code>nInfo</code> is <code>FEATURE_FIELD_VISIBLE</code>, <code>nvResult</code> returns one of the following values:</p> <ul style="list-style-type: none"> <li>• <b>TRUE</b>—This feature is visible.</li> <li>• <b>FALSE</b>—This feature is not visible.</li> </ul>
<b>svResult</b>	Returns a string value when <code>nInfo</code> produces a string result.

## Return Values

**Table 62** • FeatureGetData Return Values

Return Value	Description
<b>0</b>	FeatureGetData was successful.
<b>&lt; 0</b>	<p>FeatureGetData failed. Call <a href="#">FeatureError</a> for additional information.</p> <p>You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a>.</p>

## FeatureGetData Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the FeatureGetData function.
 *
 * This example script demonstrates a function that retrieves
 * information about a specified feature.
 *
 * Comments: To run this example script, create a project (or
 *           insert into a project) with features and/or
 *           subfeatures with components containing files.
 *

```

```

\*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

// Include iswi.h for Windows Installer API function prototypes and constants,
// and to declare code for the OnBegin and OnEnd events.
#include "iswi.h"

    // The keyword export identifies MyFunction() as an entry-point function.
    // The argument it accepts must be a handle to the Installer database.
    export prototype MyFunction(HWND);

    // To Do: Declare global variables, define constants, and prototype user-
    //         defined and DLL functions here.

function MyFunction(hMSI)

    STRING  svDir, szTitle, szMsg, svResult;
    NUMBER  nvResult;

begin

    svDir   = INSTALLDIR;
    szTitle = "Select Features";
    szMsg   = "Select the features you want to install on your computer.";

    // Display available features.
    SdFeatureTree (szTitle, szMsg, svDir, "", 2);

    // Get the description property for Feature1.
    FeatureGetData (MEDIA, "Feature1", feature_FIELD_DESCRIPTION, nvResult, svResult);

    // Display the description for Feature1.
    MessageBox ("Feature1's description is: " + svResult, INFORMATION);

    // Determine whether or not Feature2 is selected.
    FeatureGetData (MEDIA, "Feature2", feature_FIELD_SELECTED, nvResult, svResult);

    // Display a message indicating whether or not Feature2 is selected.
    if nvResult=0 then
        MessageBox ("Feature2 is not selected.", INFORMATION);
    else
        MessageBox ("Feature2 is selected.", INFORMATION);
    endif;

end;

```

## FeatureGetItemSize



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **FeatureGetItemSize** function retrieves the size—in bytes—of a specified feature. The sizes of subfeatures are not included.



**Note** • This function cannot be used with script-created feature sets.

## Syntax

```
FeatureGetItemSize ( szFeatureSource, szFeature, nvSize );
```

## Parameters

**Table 63** • FeatureGetItemSize Parameters

Parameter	Description
<b>szFeatureSource</b>	Specifies the media name of the file media that contains the feature whose size is to be retrieved.
<b>szFeature</b>	Specifies the name of the feature whose size is to be retrieved.
<b>nvSize</b>	Returns the size in bytes of the specified feature. You can also call <a href="#">FeatureGetData</a> to get the total original file size of a specified feature. Call <a href="#">FeatureTotalSize</a> to determine the total size of all selected features and subfeatures.

## Return Values

**Table 64** • FeatureGetItemSize Return Values

Return Value	Description
<b>0</b>	FeatureGetItemSize was successful.
<b>&lt; 0</b>	FeatureGetItemSize failed. Call <a href="#">FeatureError</a> for additional information.

## FeatureGetItemSize Example

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the FeatureGetItemSize function.
*
* This example calls FeatureGetItemSize to get the size of
* a feature and a subfeature. The sizes are displayed
* in a dialog.
*
* Notes: To run this example script, create a project (or
*        insert into a project) with several features and/or
*        subfeatures with components containing files.
*        You must name one feature and one subfeature as
```

```

*      indicated in the #define statements in this example,
*      or change the #define statements to reference your
*      feature names.
*
\*-----*/

#define FEAT_NAME1 "Program Files"
#define FEAT_NAME2 "Example Files\\Graphics"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_FeatureGetItemSize(HWND);

function ExFn_FeatureGetItemSize(hMSI)
    NUMBER nvSize;
    STRING szString;
    LIST listInfo;
begin

    // Create a string list to store feature sizes.
    listInfo = ListCreate (STRINGLIST);

    // Get the size of FEAT_NAME1.
    FeatureGetItemSize (MEDIA, FEAT_NAME1, nvSize);

    // Convert the number to a string.
    NumToStr (szString, nvSize);

    // Put the string into the list.
    ListAddString (listInfo, "The size in bytes of " + FEAT_NAME1 +
        " is: " + szString, AFTER);

    // Get the size of FEAT_NAME2.
    FeatureGetItemSize (MEDIA, FEAT_NAME2, nvSize);

    // Convert the number to a string.
    NumToStr (szString, nvSize);

    // Put the string into the list.
    ListAddString (listInfo, "The size in bytes of " + FEAT_NAME2 +
        " is: " + szString, AFTER);

    // Display the list of feature sizes.
    SdShowInfoList ("Results of Calls to FeatureGetItemSize",
        "The feature sizes are:", listInfo);

    // Release the list from memory.
    ListDestroy (listInfo);

end;

```

## FeatureGetTotalCost



**Note** • This function is obsolete. Use the [FeatureGetCostEx](#) function instead.



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **FeatureGetTotalCost** function determines the total space, in kilobytes (KB), required on the target drive for the feature installations and uninstallations that have been specified (for example, by end user selections in feature or setup type dialogs). When determining required drive space, this function takes into account which features are currently selected, whether any components associated with the features are currently filtered by operating system or language, and the cluster size on the target drive.

## Syntax

```
FeatureGetTotalCost ( szFeatureSource, szTargetDir, nvTotalRequiredSpace );
```

## Parameters

**Table 65** • FeatureGetTotalCost Parameters

Parameter	Description
<b>szFeatureSource</b>	Specifies the media name of the file media whose features have been specified for installation and uninstallation. Typically, you would pass the system variable <a href="#">MEDIA</a> in this parameter.
<b>szTargetDir</b>	Specifies the drive that is used, or the path whose drive is used, in determining required drive space. Typically, you would pass the system variable <a href="#">TARGETDIR</a> in this parameter.
<b>nvTotalRequiredSpace</b>	Returns the required drive space, in KB.

## Return Values

**Table 66** • FeatureGetTotalCost Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully determined the required drive space.
<b>&lt; 0</b>	Indicates that the function could not determine the required drive space.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

# FeatureInitialize



**Project** • This information applies to *InstallScript* projects.

The **FeatureInitialize** function is supported only for compatibility with scripts created in earlier versions of InstallShield. It is recommended that you avoid using multiple file media libraries in InstallShield because they are no longer necessary and they can break your installation.

The **FeatureInitialize** function associates a media name with a file media library and prepares that media library for access.

## Syntax

```
FeatureInitialize ( szMediaLibrary, szMediaLibraryFile );
```

## Parameters

**Table 67 • FeatureInitialize Parameters**

Parameter	Description
<b>szMediaLibrary</b>	Specifies the media name to associate with the file media library whose files are to be transferred with <a href="#">FeatureMoveData Example</a> .  The media name Data is reserved for use with Data1.cab, the default file media library. You may not pass Data in the parameter szMediaLibrary.
<b>szMediaLibraryFile</b>	Specifies the file name of the file media library to be initialized. The file name must be in the format xxx1.cab; for example, second1.cab or wow1.cab. Do not specify a path; this file must reside in the installation's source folder (SRCDIR).

## Return Values

**Table 68 • FeatureInitialize Return Values**

Return Value	Description
<b>0</b>	FeatureInitialize was successful.
<b>&lt; 0</b>	FeatureInitialize failed.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

## Additional Information

This function cannot be used with script-created feature sets.

A file media library that is initialized with FeatureInitialize must be installed by calling **FeatureMoveData**. **FeatureTransferData** fails if called to install such a media library.

If an installation installs file media libraries that are initialized with **FeatureInitialize**, the uninstallation must be enabled by calling **DeinstallStart** rather than **MaintenanceStart**. (This is the case because files must be installed by calling **FeatureMoveData**, which does not support maintenance setups.)

The maintenance/uninstallation feature that is stored in szMediaLibraryFile is not installed by **FeatureMoveData**; only the first call to **FeatureMoveData**, before calling **FeatureInitialize**, can install a file media library's maintenance/uninstallation feature.



It is not necessary to call **FeatureInitialize** before accessing the default file media library (Data1.cab) using the default media name Data. The default media is initialized automatically during installation initialization. The file media library must reside in the installation's source folder. The name of this folder is assigned to the system variable SRCDIR during installation initialization.

## FeatureInitialize Example

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the FeatureInitialize function.
*
*\-----*/

NUMBER nResult, n;

program

    // Set a target directory.
    TARGETDIR = "C:\\temp" ^ MEDIA;

    // Select features to transfer.
    nResult = FeatureDialog ("", "", TARGETDIR, "");

    if (nResult < 0) then
        sprintfBox (SEVERE, "FeatureDialog ERROR", "%ld", nResult);
    endif;

    // Transfer files associated with data1.cab (Data Media).
    nResult = FeatureMoveData (MEDIA, n, 0);
    if (nResult < 0) then
        sprintfBox (SEVERE, "FeatureMoveData ERROR", "%ld", nResult);
    endif;

    // Set up for second media.
    MEDIA = "second";

    // Set a target directory.
    TARGETDIR = "C:\\temp" ^ MEDIA;

    // Associate new media with second1.cab cab file.
    nResult = FeatureInitialize (MEDIA, "second1.cab");
    if (nResult < 0) then
        sprintfBox (SEVERE, "FeatureInitialize ERROR", "%ld", nResult);
    endif;

    // Select features to transfer.
    nResult = FeatureDialog ("", "", TARGETDIR, "");
    if (nResult < 0) then
        sprintfBox (SEVERE, "FeatureDialog ERROR", "%ld", nResult);
    endif;

    // Reinitialize FeatureMoveData function.
    nResult = FeatureMoveData ("", n, 0);
    if (nResult < 0) then
        sprintfBox (SEVERE, "FeatureMoveData ERROR", "%ld", nResult);
    endif;
endprogram
```

```

endif;

// Transfer files associated with second1.cab (second Media).
nResult = FeatureMoveData (MEDIA, n, 0);
if (nResult < 0) then
    SprintfBox (SEVERE, "FeatureMoveData ERROR", "%ld", nResult);
endif;

endprogram

// Source file: Is5fn628.rul

```

## FeatureIsItemSelected



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

*Project-specific differences are noted where appropriate.*

In InstallScript projects, **FeatureIsItemSelected** determines the current selection state of the specified feature. In InstallScript MSI projects, **FeatureIsItemSelected** determines the current installed state of the specified feature. You can also use [FeatureGetData](#) to determine if a feature is selected.

**FeatureIsItemSelected** is typically called to perform feature-specific tasks before or after file transfer. To perform feature-specific tasks during file transfer in an InstallScript installation, it is recommended that you place code in feature event handler functions.

### Syntax

```
FeatureIsItemSelected ( szFeatureSource, szFeature );
```

## Parameters

**Table 69** • FeatureIsItemSelected Parameters

Parameter	Description
<b>szFeatureSource</b>	In InstallScript projects, specifies the media library for which the installed state or selection setting is determined.  In InstallScript MSI projects, specifies the media name of the script-created feature set.
<b>szFeature</b>	Specifies the name of the feature for which the installed state or selection setting is determined. To learn how to refer to top-level features and subfeatures, see <a href="#">Specifying Features and Subfeatures in Function Calls</a> .

## Return Values

**Table 70** • FeatureIsItemSelected Return Values

Return Value	Description
<b>TRUE (1)</b>	In InstallScript projects, szFeature is selected.  In InstallScript MSI projects, szFeature's installed state is INSTALLSTATE_LOCAL (feature was installed on the local drive).
<b>FALSE (0)</b>	In InstallScript projects, szFeature is deselected.  In InstallScript MSI projects, szFeature's installed state is INSTALLSTATE_ABSENT (feature was uninstalled).
<b>&lt; 0</b>	The function failed to determine if the feature was installed or selected. Call <a href="#">FeatureError</a> for additional information.

## FeatureIsItemSelected Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the FeatureIsItemSelected function.
 *
 * This example script displays a dialog that displays a list
 * of features in the setup that the user can install and the
 * amount of space that each feature occupies. When the user
 * selects features, the installed state of the features is
 * provided.
 *
 * Comments: To run this example script, create a project (or
 *           insert into a project) with several features and/or
 *           subfeatures with components containing files.
 *
\*-----*/

```

```

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

// Include iswi.h for Windows Installer API function prototypes and constants,
// and to declare code for the OnBegin and OnEnd events.
#include "iswi.h"

// The keyword export identifies MyFunction() as an entry-point function.
// The argument it accepts must be a handle to the Installer database.
export prototype MyFunction(HWND);

// To Do: Declare global variables, define constants, and prototype user-
// defined and DLL functions here.

function MyFunction(hMSI)

    STRING  szTitle, szMsg, svDir;
    NUMBER  nResult;

begin

    svDir   = INSTALLDIR;
    szTitle = "Select Features";
    szMsg   = "Select the features you want to install on your computer.";

    // Display available features.
    SdFeatureTree (szTitle, szMsg, svDir, "", 2);

    // Determine the installed state of Subfeature1.
    nResult = FeatureIsItemSelected (MEDIA, "Feature1\\Subfeature1");

    // Display message indicating the installed state of Subfeature1.
    if nResult = 1 then
        MessageBox ("Subfeature1 is installed locally.", INFORMATION);
    else
        MessageBox ("Subfeature1 is uninstalled.", INFORMATION);
    endif;

end;

```

## FeatureListItems



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **FeatureListItems** function lists all subfeatures under the feature specified in `szFeature`. The list of fully qualified subfeature names is stored in `listFeatures`. If `szFeature` has subfeatures, `listFeatures` returns an empty list.

### Syntax

```
FeatureListItems ( szFeatureSource, szFeature, listFeatures );
```

## Parameters

**Table 71** • FeatureListItems Parameters

Parameter	Description
<b>szFeatureSource</b>	Specifies the media name of the script-created feature set whose subfeatures are listed.
<b>szFeature</b>	Specifies the feature whose subfeatures are listed. Pass a null string ("") in this parameter to list all top-level features. To learn how to refer to top-level features and subfeatures, see <a href="#">Specifying Features and Subfeatures in Function Calls</a> .
<b>listFeatures</b>	Returns the list of features. You must call <a href="#">ListCreate</a> before calling FeatureListItems to initialize the string list identified by listFeatures.

## Return Values

**Table 72** • FeatureListItems Return Values

Return Value	Description
<b>0</b>	FeatureListItems listed the features.
<b>&lt; 0</b>	FeatureListItems was unable to list the features. Call <a href="#">FeatureError</a> for additional information.

## FeatureListItems Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the FeatureListItems function.
 *
 * This example script demonstrates a function that lists all
 * subfeatures under a specified feature.
 *
 * Comments: To run this example script, create a project (or
 *           insert into a project) with several features and/or
 *           subfeatures with components containing files.
 *
 \*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

// Include iswi.h for Windows Installer API function prototypes and constants,
// and to declare code for the OnBegin and OnEnd events.
#include "iswi.h"

// The keyword export identifies MyFunction() as an entry-point function.
// The argument it accepts must be a handle to the Installer database.

```

```

    export prototype MyFunction(HWND);

// To Do:  Declare global variables, define constants, and prototype user-
//          defined and DLL functions here.

function MyFunction(hMSI)

    STRING  szTitle, szMsg;
    NUMBERlistID;

begin

    szTitle = "List MEDIA Features";
    szMsg   = "MEDIA contains the following top-level features:";

    // Initialize the string list.
    listID = ListCreate (STRINGLIST);

    // Create a list of top-level features in the specified media.
    FeatureListItems (MEDIA, "", listID);

    // Display the list of top-level features.
    SdShowInfoList (szTitle, szMsg, listID);

end;

```

## FeatureLoadTarget



**Project** • This information applies to *InstallScript* projects.

The **FeatureLoadTarget** function is called automatically during the initialization of any installation for which a valid log file exists.

### Syntax

```
FeatureLoadTarget( szReserved );
```

## Parameters

**Table 73** • FeatureLoadTarget Parameters

Parameter	Description
<b>szReserved</b>	Pass a null string ("") in this parameter. No other value is allowed.

## Return Values

**Table 74** • FeatureLoadTarget Return Values

Return Value	Description
<b>0</b>	This function always returns zero (0).

## Additional Information

FeatureLoadTarget reads the log file and retrieves the target directory of all filegroups in the setup. FeatureLoadTarget retrieves the values of all text substitutions used by the setup; this includes the target directory of all filegroups and the value of all target-directory-related system variables, as well as all other text substitution values.

# FeatureMoveData



**Project** • This information applies to InstallScript projects:

The **FeatureMoveData** function transfers and decompresses files associated with selected features in the file media referenced by szFeatureSource. The function automatically prompts the end user for the next disk when it is needed.

If you use this function to transfer files to [WINSYSDIR64](#), you must first disable file system redirection using [WOW64FSREDIRECTION](#). Otherwise, files being transferred to WINSYSDIR64 are incorrectly redirected to the 32-bit SysWOW64 folder. Since some Windows functionality that could be used by the installation requires that redirection be enabled to work, Windows documentation recommends that you disable redirection only for as long as necessary. It is recommended that you then enable file system redirection as soon as you have completed transferring the necessary files to WINSYSDIR64. To learn more, see [Targeting 64-Bit Operating Systems with InstallScript Installations](#).

When you call FeatureMoveData, Do (SELFREGISTRATIONPROCESS) is called automatically after the files are installed but before the call returns. Therefore, if your installation needs to perform additional actions after the file transfer but before self-registration, place these actions in the OnMoved event; the OnMoved event is called after the file transfer but before the batch self-registration occurs.

You can call FeatureMoveData more than once on the same media, but you must reset internal structures before the second and subsequent calls by calling FeatureMoveData with a null string ("") in the first parameter position. InstallShield automatically initializes the default media and internal structures before your first call to FeatureMoveData.



**Note** • This function cannot be used with script-created feature sets.

Syntax

FeatureMoveData ( szFeatureSource, nvReserved, nReserved );

Parameters

Table 75 • FeatureMoveData Parameters

Parameter	Description
szFeatureSource	Specifies the media name of the file media whose files are to be transferred.
nvReserved	Pass a number variable in this argument. No useful data is returned.
nReserved	Pass zero in this parameter. No other value is allowed.

Return Values

Table 76 • FeatureMoveData Return Values

Return Value	Description
0	FeatureMoveData was successful.
< 0	FeatureMoveData failed. Call <a href="#">FeatureError</a> for additional information.

FeatureMoveData Example

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the functions SdSetupTypeEx, SdFeatureDialog,
* FeatureMoveData, FeatureError, and PlaceWindow.
*
* Notes: To run this example script, create a project with
*        the following features (f), subfeatures (sf),
*        and components (c):
*
*        (f) Program_Files
*            (c) Program_DLLS
*            (c) Program_EXEs
*        (f) Example_Files
*            (sf) Small_Documents
*                (c) Small_Document_Examples
*            (sf) Books
*                (c) Book_Examples
*            (sf) Graphics
*                (c) Graphic_Examples
```



```

*          (f) Help_Files
*          (c) Help_Files
*          (f) Utilities
*          (sf) Grammar_Checker
*          (c) Grammar_Checker
*          (sf) Art_Studio
*          (c) Art_Studio
*          (f) Evaluation_Copy
*          (c) Evaluation_Copy
*          (c) Help_Files
*
* Insert "dummy" files into the components. Make sure you
* define the correct file name for the main EXE (MAIN_EXE,
* below) that you insert into the Program_EXEs component.
*
* This example script installs files, adds an icon to the
* Start Programs menu, and provides uninstallation
* functionality.
*
\*-----*/

// Define strings. In a real installation, you would define these in the String Editor
// view and precede each string identifier in the script with the at symbol (@).
#define FEAT_SELECT_TITLE "Select features"
#define FEAT_SELECT_MSG "Select features and subfeatures to install."
#define FEAT_PROGRAMFILES_DISPLAYNAME "Program Files"
#define PASSWORD_PROMPT "Please enter the password."
#define PASSWORD_ERRMSG "Password incorrect. Please enter again."
#define TITLE_MAIN "Word Processor"
#define TITLE_CAPTIONBAR "Word Processor Setup"
#define APPBASE_PATH "Your Company\\Word Processor"
#define COMPANY_NAME "Your Company"
#define PRODUCT_NAME "Word Processor"
#define PRODUCT_VERSION "1.0"
#define PRODUCT_KEY "Word Processor"
#define DEINSTALL_KEY "Word Processor"
#define UNINSTALL_NAME "Word Processor"
#define ADDINGICON "Adding program icon to the Start Programs menu..."
#define PROGRAMDIR "Program"
#define DEFAULT_FOLDER_NAME ""
#define APP_NAME "Word Processor"
#define COMPLETE_MSG "Setup is complete. You can run Word Processor from the Start Programs menu."
#define MAIN_EXE "WRITE.EXE"
#define SETUPTYPE_TITLE "Setup Type Selection"
#define SETUPTYPE_MSG "Please select a setup type."
#define SETUPTYPE_CUSTOM "Custom"

// Global variable declarations.

// Function declarations.
prototype SetUpFileTransfer ();
prototype HandleFeatureError (NUMBER);
prototype FinishSetup ();

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_FeatureMoveData(HWND);

```

```

function ExFn_FeatureMoveData(hMSI)
    STRING svData, svLogFile, szProgram, szFeature;
    STRING svResult, svSetupType, svDir;
    BOOL    bInitStepsDone, bPwdValid;
    NUMBER  nvData, nvDisk, nResult;
begin

    SetUpFileTransfer ();

    // Disable the Back button in installation dialogs.
    Disable (BACKBUTTON);

    // Get the setup type.
    svDir = INSTALLDIR;

    SdSetupTypeEx (SETUPTYPE_TITLE, SETUPTYPE_MSG, "", svSetupType, 0);

    // If user selected Custom setup type, display feature selection dialog.
    if (svSetupType = SETUPTYPE_CUSTOM) then
        SdFeatureDialog (FEAT_SELECT_TITLE, FEAT_SELECT_MSG, svDir, "");
    endif;

    // Enable the Back button in installation dialogs.
    Enable (BACKBUTTON);

    // Set up the progress indicator, including locations for
    // progress indicator, information gauges, and billboards.
    PlaceWindow (FEEDBACK, LOWER_LEFT, LOWER_LEFT, LOWER_LEFT);

    PlaceWindow (STATUSDLG, CENTERED, LOWER_RIGHT, LOWER_RIGHT);

    PlaceWindow (BILLBOARD, CENTERED, CENTERED, CENTERED);

    Enable (STATUSDLG);

    Enable (INDVFILESTATUS);

    // Indicate the final percentage the progress bar is to show when the
    // following file transfer operation is complete.
    StatusUpdate (ON, 95);

    // Transfer files to the target system. FeatureMoveData prompts
    // for next disk in a floppy disk installation.
    nResult=FeatureMoveData (MEDIA, nvDisk, 0);

    // See the FeatureError function in action.
    HandleFeatureError (nResult);

    FinishSetup();

end;

/*-----*\
*
* Function:  SetupFileTransfer()
*
* Purpose:  This function sets up file transfer. The main reason for
*           abstracting this process into this function is to make it

```

```

*           easy to see the function calls this sample script demonstrates.
*
/*-----*/
function SetUpFileTransfer()

begin
    // Set up the installation screen.
    Enable (FULLWINDOWMODE);
    SdProductName ( PRODUCT_NAME );
    SetTitle (TITLE_MAIN, 24, WHITE);
    SetTitle (TITLE_CAPTIONBAR, 0, BACKGROUNDCAPTION);
    Enable (BACKGROUND);

    // Welcome the user, check that the system meets minimum requirements,
    // and verify the destination location.
    bInitStepsDone = FALSE;

    while (!bInitStepsDone)
        Disable (BACKBUTTON);
        Welcome ("", 0);
        Enable (BACKBUTTON);
        INSTALLDIR = PROGRAMFILES ^ APPBASE_PATH;
        if (AskDestPath ("", "", INSTALLDIR, 0) != BACK) then
            bInitStepsDone = TRUE;
        endif;
    endwhile;

    // Set installation information required for registry entries and for
    // the following call to DeinstallStart.
    InstallationInfo (COMPANY_NAME, PRODUCT_NAME,
                     PRODUCT_VERSION, PRODUCT_KEY);

    // Initialize the uninstallation log file, including registry entry.
    svLogFile = "Uninst.isu";

    DeinstallStart (INSTALLDIR, svLogFile, DEINSTALL_KEY, 0);

    RegDBSetItem (REGDB_UNINSTALL_NAME, UNINSTALL_NAME);
end;

/*-----*\
*
* Function:  HandleFeatureError
*
* Purpose:  This function evaluates the value returned by a feature
*           function and if the value is less than zero, displays the error
*           number and aborts the installation.
*
/*-----*/
function HandleFeatureError (nResult)
    NUMBER  nvError;
    STRING  svFeatureSource, svFeature, svComponent, svFile;
begin
    if (nResult < 0) then
        FeatureError (svFeatureSource, svFeature, svComponent, svFile, nvError);
        sprintfBox (INFORMATION, "Data Transfer Error Information",
                    "FeatureError returned the " +
                    "following data transfer error.\n" +
                    "Setup will now abort.\n\n" +

```

```

        "Media Name: %s\nFeature: %s\nComponent: %s\n" +
        "File: %s\nError Number: %ld",
        svFeatureSource, svFeature, svComponent, svFile, nvError);
    abort;
endif;
end;

/*-----*\
*
* Function:  FinishSetup()
*
* Purpose:  This function finishes the installation. The main reason for
*           abstracting this process into this function is to make it
*           easy to see the function calls this sample script demonstrates.
*
\*-----*/
function FinishSetup()
begin
    // Indicate the final percentage the progress bar is to show when the
    // following file transfer operation is complete.
    StatusUpdate(ON, 99);

    // Increment progress bar to 99% for creation of Start Programs menu icon.
    SetStatusWindow (96 , ADDINGICON);

    // Add the APP_NAME icon to the DEFAULT_FOLDER_NAME folder.
    szProgram = INSTALLDIR ^ PROGRAMDIR ^ MAIN_EXE;

    LongPathToQuote (szProgram, TRUE);
    AddFolderIcon (DEFAULT_FOLDER_NAME, APP_NAME, szProgram,
        INSTALLDIR ^ PROGRAMDIR, "", 0, "", REPLACE);

    Delay (1);

    // Disable the progress indicator and its settings.
    Disable (INDVFILESTATUS);
    Disable (STATUSDLG);

    // Announce installation complete and offer to view Readme file.
    MessageBox (COMPLETE_MSG, INFORMATION);
end;

```

## FeaturePatch



**Project** • This information applies to InstallScript projects.

The **FeaturePatch** function is called only in installations that use differential media. (You can check the media format by calling [MediaGetData](#) with MEDIA\_FIELD\_MEDIA\_FLAGS as its second argument).

**FeaturePatch** causes the next call to [FeatureTransferData](#) or [FeatureMoveData](#) to reinstall all features that are already installed when **FeatureTransferData** is called, including all of the maintenance/uninstallation feature's files except for Data1.hdr, Data1.cab, and Layout.bin. (Note that the maintenance/uninstallation feature is automatically

placed in your disk image by the release builder and is not displayed in InstallShield.) A copy of the running file media library is stored on the target machine for use during subsequent maintenance operations, that is, modifying, repairing, or uninstalling.

## Syntax

```
FeaturePatch ( );
```

## Parameters

None.

## Return Values

**Table 77** • FeaturePatch Return Values

Return Value	Description
0	Indicates that the function succeeded.
< 0	Indicates that the function failed.

## Additional Information

**FeaturePatch** is called in the default code for the OnUpdateUIBefore event handler function.

# FeatureReinstall



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

In a setup that has already been run, after the **FeatureReinstall** function is called, the next call to [FeatureTransferData](#) performs the file transfer that was specified the last time the setup was run.

## Syntax

```
FeatureReinstall ( );
```

## Parameters

None

## Return Values

**Table 78** • FeatureReinstall Return Values

Return Value	Description
0	Indicates that the function successfully prepared the setup for reinstallation.
< 0	Indicates that the function was unable to prepare the setup for reinstallation.

## FeatureRemoveAll



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

**FeatureRemoveAll** is used during a maintenance install to force the removal of all features that were installed previously. FeatureRemoveAll is generally called when the user selects the Remove option in the [SdWelcomeMaint](#) dialog.

Calling FeatureRemoveAll causes all features to be deselected. As a result, when [FeatureTransferData](#) is called, any features that have already been installed (determined by reading the setup log file, if it exists) are removed (uninstalled).

Setup determines whether or not features have been previously installed by reading the setup log file. If no valid log file is found during setup initialization, all features are considered *not installed*. In this case, calling **FeatureRemoveAll** clears all feature selections, but does not force uninstallation of the features when **FeatureTransferData** is called.



**Note** • **FeatureRemoveAll** also deselects all internal features, including the features containing the Disk 1 setup files that are automatically installed to the DISK1TARGET location.

### Syntax

```
FeatureRemoveAll ( );
```

### Parameters

None

## Return Values

**Table 79** • FeatureRemoveAll Return Values

Return Value	Description
0	Indicates that the function successfully deselected all feature selections.
< 0	Indicates that the function was unable to deselect all feature selections.

# FeatureRemoveAllInLogOnly



**Project** • This information applies to InstallScript projects.

The **FeatureRemoveAllInLogOnly** function is called during an update installation to force the removal of all features that are not in the current media but were installed previously, as recorded in the setup log file.

## Syntax

```
FeatureRemoveAllInLogOnly ( );
```

## Parameters

None.

## Return Values

**Table 80** • FeatureRemoveAllInLogOnly Return Values

Return Value	Description
0	Indicates that the function successfully deselected the feature selections.
< 0	Indicates that the function was unable to deselect the feature selections.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

## Additional Information

Calling **FeatureRemoveAllInLogOnly** causes all features that are not in the current media but were installed previously to be deselected. As a result, when [FeatureTransferData](#) is called, those features will be removed (uninstalled).



**Caution** • Do not call **FeatureRemoveAllInLogOnly** in an "add-on" installation—that is, an installation with the same **INSTANCE\_GUID** as a application already on the system that includes a subset of the features in the earlier installation. The result would be the uninstallation (removal) of all features that were contained in the earlier installation, but not the features in the current installation.

If no valid log file is found during installation initialization, calling `FeatureRemoveAllInLogOnly` has no effect.

## FeatureRemoveAllInMedia



**Project** • This information applies to *InstallScript* projects.

The **FeatureRemoveAllInMedia** is used during a maintenance installation to force the removal of all features that are in the current media and were installed previously. This function is generally called when the user selects the Remove option in the [SdWelcomeMaint](#) dialog.

When you call `FeatureRemoveAllInMedia`, only the features listed in the media header are removed; however, in the case of an updated application in which features were removed but not uninstalled during the update—that is, the features did not exist in the update media but did exist in the original media (and the update did not call [FeatureRemoveAllInLogOnly](#))—whether or not these features are removed during uninstallation depends on whether the application was updated via a differential media or a full update media:

- If the application was updated via a full media, the full media header replaces the original media header; therefore, these features are not removed during uninstallation.
- If the application was updated via a differential media, both header files are present; therefore, `FeatureRemoveAllInMedia` removes these features.

To ensure that all installed features are uninstalled, an installation should call [FeatureRemoveAllInMediaAndLog](#). This ensures that all features are removed.

### Syntax

```
FeatureRemoveAllInMedia ( );
```

### Parameters

None.

### Return Values

**Table 81** • FeatureRemoveAllInMedia Return Values

Return Value	Description
0	Indicates that the function successfully deselected all feature selections.
< 0	Indicates that the function was unable to deselect all feature selections.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

### Additional Information

Calling `FeatureRemoveAllInMedia` causes all features that are in the current media to be deselected. As a result, when [FeatureTransferData](#) is called, any features that have already been installed (determined by reading the installation log file, if it exists) are removed (uninstalled).



The installation determines whether or not features have been previously installed by reading the installation log file. If no valid log file is found during installation initialization (MAINTENANCE is FALSE), all features are considered *not installed*. In this case, calling **FeatureRemoveAllInMedia** causes all features to be deselected and not installed. However, a subsequent call to **FeatureTransferData** does not uninstall anything (since the log file contains the information regarding what to uninstall). Calling **FeatureRemoveAllInMedia** in this case is not recommended.



**Note** • **FeatureRemoveAll** also deselects all internal features, including the maintenance/uninstallation feature.

## FeatureRemoveAllInMediaAndLog



**Project** • This information applies to InstallScript projects.

The **FeatureRemoveAllInMediaAndLog** function is called during an update installation to force the removal of all features that were installed previously—both those that are in the current media, and those that are not in the current media but are recorded in the setup log file.

### Syntax

```
FeatureRemoveAllInMediaAndLog ( );
```

### Parameters

None.

### Return Values

**Table 82** • FeatureRemoveAllInMediaAndLog Return Values

Return Value	Description
0	Indicates that the function successfully deselected all feature selections.
< 0	Indicates that the function was unable to deselect all feature selections.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <b>FormatMessage</b> .

### Additional Information

Calling **FeatureRemoveAllInMediaAndLog** is equivalent to calling both **FeatureRemoveAllInMedia** and **FeatureRemoveAllInLogOnly**.

Calling **FeatureRemoveAllInMediaAndLog** causes all features that were installed previously to be deselected. As a result, when **FeatureTransferData** is called, those features will be removed (uninstalled).

The installation determines whether or not features have been previously installed by reading the setup log file. If no valid log file is found during installation initialization (MAINTENANCE is FALSE), all features are considered *not installed*. In this case, calling **FeatureRemoveAllInMediaAndLog** will cause all features to be deselected and not

installed. However, a subsequent call to **FeatureTransferData** will not uninstall anything (since the log file contains the information regarding what to uninstall). Calling **FeatureRemoveAllInMediaAndLog** in this case is not recommended.



**Note** • *FeatureRemoveAllInMediaAndLog will also deselect all internal features, including the maintenance/uninstallation feature.*

## FeatureSaveTarget



**Project** • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

**FeatureSaveTarget** gets the current values of all text substitutions used by the setup project and stores them in the installation log file. This includes the current target directory of all components and the value of all target-directory-related system variables, as well as all other text substitution values.

When the setup initializes and a valid log file is found, any values previously stored with this function are automatically restored. This allows a maintenance setup to update previously installed components and install new components to the appropriate location.

### Syntax

```
FeatureSaveTarget ( szReserved );
```

### Parameters

**Table 83** • FeatureSaveTarget Parameters

Parameter	Description
<b>szReserved</b>	Pass a null string ("" ) in this parameter. No other value is allowed.

### Return Values

This function always returns zero (0).

## FeatureSelectItem



**Project** • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

The **FeatureSelectItem** function sets a feature's selection status to either selected or not selected. You can use **FeatureSelectItem** to change selection status before displaying features in feature dialogs, and you can use it to change or override selections afterward, depending on your installation's requirements.



**Note** • If you call a setup type selection function (such as **FeatureSetupTypeSet**, **SetupType2**, **SdSetupType**, or **SdSetupTypeEx**) after calling **FeatureSelectItem**, the feature selections set by **FeatureSelectItem** are overridden by the selections in the setup type. (This applies to features that you add to your installation project and does not affect internal features.) Be aware that **SetupType2** is called in the default code for the **OnFirstUIBefore** event handler function, which is called after the **OnBegin**, **OnCCPSearch**, and **OnAppSearch** event handler functions are called. You can override or customize the **OnFirstUIBefore** event handler.

If you use **FeatureSelectItem** to deselect a feature, note that the same rules that apply to deselecting a feature through the feature dialog apply. That is, a feature cannot be deselected if it is required by a currently selected feature. Therefore, to deselect a feature that is required by another feature, ensure to deselect the requiring feature before attempting to deselect the required feature.

- It is usually not necessary to call **FeatureSelectItem** during maintenance mode, since the previous feature selections are automatically loaded from the existing log file.

## Syntax

```
FeatureSelectItem ( szFeatureSource, szFeature, bSelect );
```

## Parameters

**Table 84 • FeatureSelectItem Parameters**

Parameter	Description
<b>szFeatureSource</b>	Specifies the media name of the script-created feature set or (in InstallScript projects) file media library containing the feature whose selection status is to be set.
<b>szFeature</b>	<p>Specifies the feature whose selection status is to be set.</p> <p>You can specify a null string ("") for this parameter. If you specify a null string, all features from szFeatureSource are either selected or deselected, depending on the value of bSelect.</p> <p>For more information about specifying features and subfeatures in InstallScript, see <a href="#">Specifying Features and Subfeatures in Function Calls</a>.</p>
<b>bSelect</b>	<p>Specifies whether the feature should be selected. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>TRUE</b>—Select the specified feature.</li> <li>• <b>FALSE</b>—Do not select the specified feature.</li> </ul>

## Return Values

**Table 85 • FeatureSelectItem Return Values**

Return Value	Description
<b>0</b>	<b>FeatureSelectItem</b> successfully set the feature's selection status.
<b>&lt; 0</b>	<b>FeatureSelectItem</b> was unable to set the feature's selection status. For additional information, call <a href="#">FeatureError</a> .

## FeatureSelectItem Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the FeatureSelectItem function.
 *
 * This example script demonstrates a function that sets a
 * feature's status to either selected or not selected.
 *
 * Comments: To run this example script, create a project (or
 *           insert into a project) with several features and/or
 *           subfeatures with components containing files.
 *
 \*-----*/

```

```
// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

// Include iswi.h for Windows Installer API function prototypes and constants,
// and to declare code for the OnBegin and OnEnd events.
#include "iswi.h"

// The keyword export identifies MyFunction() as an entry-point function.
// The argument it accepts must be a handle to the Installer database.
export prototype MyFunction(HWND);

// To Do: Declare global variables, define constants, and prototype user-
//         defined and DLL functions here.

function MyFunction(hMSI)

    STRING  svDir, szTitle, szMsg;

begin

    svDir   = INSTALLDIR;
    szTitle = "Select Features";
    szMsg   = "Select the features you want to install on your computer.";

    // Set the selection status of Subfeature2 to not selected.
    FeatureSelectItem (MEDIA, "Feature1\\Subfeature2", FALSE);

    // Display available features.
    SdFeatureTree (szTitle, szMsg, svDir, "", 2);

end;
```

## FeatureSelectNew



**Project** • This information applies to InstallScript projects.

The **FeatureSelectNew** function sets the selection status of all new features to either selected or unselected.

### Syntax

```
FeatureSelectNew (szFeatureSource, bSelect);
```

## Parameters

**Table 86** • FeatureSelectNew Parameters

Parameter	Description
<b>szFeatureSource</b>	Specifies the media name of the file media library containing the new features whose selection status is to be set. Typically, this argument is set equal to the system variable <a href="#">MEDIA</a> .
<b>bSelected</b>	Specifies whether the new features are selected or deselected. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>TRUE</b>—Selects the new features.</li> <li>• <b>FALSE</b>—Deselects the new features.</li> </ul>

## Return Values

**Table 87** • FeatureSelectNew Return Values

Parameter	Description
<b>0</b>	FeatureSelectNew successfully set the new features' selection status.
<b>&lt; 0</b>	FeatureSelectNew was unable to set the new features' selection status.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

## Additional Information

New features are features that exist in the installation's file media library but not in the existing log file. FeatureSelectNew is typically called in an update installation to select new features before displaying features in features dialogs. FeatureSelectNew is called in the default code for the OnUpdateUIBefore event handler function.



**Note** • If you call a setup type selection function after calling FeatureSelectNew, the feature selections set by FeatureSelectNew are overridden by the selections in the setup type.

It is usually unnecessary to call FeatureSelectNew during maintenance mode because the previous feature selections are automatically loaded from the existing log file.

# FeatureSetData



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **FeatureSetData** function sets properties and data for the specified feature. Most of the settings correspond to the properties in the Features view.

## Syntax

```
FeatureSetData ( szFeatureSource, szFeature, nInfo, nData, szData );
```

## Parameters

**Table 88 • FeatureSetData Parameters**

Parameter	Description
<b>szFeatureSource</b>	Specifies the media name of the script-created feature set or (in InstallScript projects) file media library that contains the feature for which properties and data are to be set.
<b>szFeature</b>	Specifies the name of the feature. For more information about specifying features and subfeatures in InstallScript, see <a href="#">Specifying Features and Subfeatures in Function Calls</a> .
<b>nInfo</b>	<p>Specifies the type of information to be set. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>FEATURE_FIELD_DESCRIPTION</b>—This text is displayed in the Description field of selection dialogs.</li> <li>• <b>FEATURE_FIELD_FTPLOCATION</b>—An FTP location.</li> <li>• <b>FEATURE_FIELD_HTTPLOCATION</b>—An HTTP location.</li> <li>• <b>FEATURE_FIELD_STATUS (not for script-created feature sets)</b>—This text is displayed in the progress indicator during file transfer.</li> <li>• <b>FEATURE_FIELD_VISIBLE</b>—Indicates whether or not the feature is visible in a feature selection dialog. The parameter nData can be one of the following: TRUE: The feature is visible. FALSE: The feature is not visible.</li> <li>• <b>FEATURE_FIELD_SELECTED</b>—Sets the selection status of the feature. This setting has the same effect as <a href="#">FeatureSelectItem</a>. The parameter nData can be one of the following: TRUE: Select the feature. FALSE: Do not select the feature.</li> <li>• <b>FEATURE_FIELD_SIZE (not for file media)</b>—The total original file size for the feature.</li> <li>• <b>FEATURE_FIELD_MISC</b>—Miscellaneous text.</li> <li>• <b>FEATURE_FIELD_DISPLAYNAME (not for object projects)</b>—Indicates the name displayed in feature selection dialogs for the feature specified in szFeature.</li> <li>• <b>FEATURE_FIELD_IMAGE</b>—Overrides a feature's default icon assignment. Pass the index of the icon to display in nData. To specify that no icon should be displayed for the feature, pass -1 in nData.</li> </ul>
<b>nData</b>	Specifies a numeric value to set when the information indicated by nInfo is numeric.
<b>szData</b>	Specifies a string value to set when the information indicated by nInfo is a string.



## Return Values

**Table 89** • FeatureSetData Return Values

Return Value	Description
0	FeatureSetData was successful.
< 0	FeatureSetData failed. Call <a href="#">FeatureError</a> for additional information.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

## FeatureSetData Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the FeatureSetData function.
 *
 * This example script demonstrates a function that sets data
 * and properties for a specified feature.
 *
 * Comments: To run this example script, create a project (or
 *           insert into a project) with several features and/or
 *           subfeatures with components containing files.
 *
\*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

// Include iswi.h for Windows Installer API function prototypes and constants,
// and to declare code for the OnBegin and OnEnd events.
#include "iswi.h"

// The keyword export identifies MyFunction() as an entry-point function.
// The argument it accepts must be a handle to the Installer database.
export prototype MyFunction(HWND);

// To Do: Declare global variables, define constants, and prototype user-
//        defined and DLL functions here.

function MyFunction(hMSI)

    STRING  svDir, szTitle, szMsg, szData;
    NUMBER  nData;

begin

    svDir   = INSTALLDIR;
    szTitle = "Select Features";
    szMsg   = "Select the features you want to install on your computer.";
    szData  = "Required Feature";

```

```
// Hide Feature1 from the end user.
FeatureSetData (MEDIA, "Feature1", FEATURE_FIELD_VISIBLE, FALSE, szData);

// Set the display name for Feature2.
FeatureSetData (MEDIA, "Feature2", FEATURE_FIELD_DISPLAYNAME, nData, szData);

// Display available features.
SdFeatureTree (szTitle, szMsg, svDir, "", 2);

end;
```

## FeatureSetTarget



**Project** • This information applies to the following project types:

- InstallScript
- InstallScript MSI

The **FeatureSetTarget** function assigns the value of szLocation to the public property (in InstallScript MSI projects) or property variable (in InstallScript projects) that is specified by szPropertyVar. Properties or property variables can be used in the Destination field of the Components view and in the Target field of a shortcut in the Shortcuts view. Call **FeatureSetTarget** before calling [FeatureMoveData](#).



**Note** • This function cannot be used with script-created feature sets.

This function cannot set the value of an object's property variable. To set the value of an object's property variable, you must use the object's ScriptDefinedVar property; see the object's help page for details. (You can display an object's help page by selecting the object in the Objects view or, if the object is included in your project, the Features or Setup Design view.) For information on adding the ScriptDefinedVar property to an object that you have created, see [Add a ScriptDefinedVar Property to My Object](#).


In InstallScript MSI projects, this function was designed to set paths for public properties that are in the **Directory** table. You can create a new directory by clicking a component's destination or a shortcut's target, and creating a new directory. This newly created directory has TARGETDIR as its Directory\_Parent. Through script, you can use **FeatureSetTarget** to set this public property (directory).

### Syntax

```
FeatureSetTarget ( szFeatureSource, szPropertyVar, szLocation );
```

## Parameters

**Table 90 • FeatureSetTarget Parameters**

Parameter	Description
<b>szFeatureSource</b>	Specifies the media name of the file media library whose user-defined variable is to be set.
<b>szPropertyVar</b>	<p>Specifies the user-defined variable. In InstallShield, user-defined variables take the form &lt;variable name&gt;.</p> <p></p> <p><b>Project •</b> (InstallScript MSI only:) The property used in this parameter must be a public property. You do not need to create an entry in the Property Manager for this property. If you intend to use the property as a destination, you must use it as a component's destination, not a feature's destination.</p>
<b>szLocation</b>	Specifies the path expression to substitute for the user-defined variable. This string should not include extra quotation marks, even if it specifies a long path. The value of szLocation can be a complete path—including drive letter and colon—or a partial path, depending on how szPropertyVar is used.

## Return Values

This function always returns 0.

## FeatureSetTarget Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the FeatureSetTarget function. After adding this
 * to a script for an InstallScript project, you will see the
 * modified value of INSTALLDIR in the SdAskDestPath
 * dialog. In a project that had any components with OTHERLOC as
 * their destination, that would also be changed.
 *
 \*-----*/

function OnBegin( )

begin
// change values of directory properties INSTALLDIR and OTHERLOC for a first-time
// installation
if (!MAINTENANCE) then
    FeatureSetTarget(MEDIA, "<INSTALLDIR>", "C:\\RightHere");
    FeatureSetTarget(MEDIA, "<OTHERLOC>", "C:\\SomewhereElse");
endif;

end;

```

# FeatureSetupTypeEnum



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **FeatureSetupTypeEnum** function enumerates all setup types associated with the specified media. These setup types are defined by you in the IDE and stored in the file media. You must create the listSetupTypes string list using the [ListCreate](#) function.



**Note** • This function cannot be used with script-created feature sets.

## Syntax

```
FeatureSetupTypeEnum ( szFeatureSource, listSetupTypes );
```

## Parameters

**Table 91** • FeatureSetupTypeEnum Parameters

Parameter	Description
<b>szFeatureSource</b>	Specifies the media name of the file media whose setup types are to be enumerated.
<b>listSetupTypes</b>	Returns a list of all setup types in the specified media. The string list identified by listSetupTypes must already have been initialized by a call to ListCreate.

## Return Values

**Table 92** • FeatureSetupTypeEnum Return Values

Return Value	Description
<b>0</b>	FeatureSetupTypeEnum was successful.
<b>&lt; 0</b>	FeatureSetupTypeEnum failed.

## FeatureSetupTypeEnum Example

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the FeatureSetupTypeEnum function,
 *
```

```

* This script enumerates the setup types in the media.
*
* Note: To run this example script, create a project (or
*       insert into a project) with several setup types
*       defined. The value of DEFTYPE must be the name
*       of one of your setup types.
*
\*-----*/

#define SDSHOWTITLE "Setup Type Enumeration"
#define SDSHOWMSG   MEDIA + " media's enumerated setup types are:"
#define SETUPTITLE  "Setup Type Selection"
#define SETUPMSG    "Select a setup type."
#define DEFTYPE     "Typical"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_FeatureSetupTypeEnum(HWND);

function ExFn_FeatureSetupTypeEnum(hMSI)
    LIST listID;
    STRING svSetupType;
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Create a list to store setup types.
    listID = ListCreate ( STRINGLIST );

    // Get the setup type names from the media into the list.
    if (FeatureSetupTypeEnum( MEDIA, listID) < 0 ) then
        MessageBox ("FeatureSetupTypeEnum failed.", WARNING);
    endif;

    // Display the setup types.
    SdShowInfoList (SDSHOWTITLE, SDSHOWMSG, listID);

    // Now show setup types in a selection dialog.
    svSetupType = DEFTYPE;
    SdSetupTypeEx (SETUPTITLE, SETUPMSG, "", svSetupType, 0);

    // Release the list from memory.
    ListDestroy (listID);

end;

```

## FeatureSetupTypeGetData



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **FeatureSetupTypeGetData** function retrieves data associated with a specified setup type. You can then use this data for any purpose.

A typical application of FeatureSetupTypeGetData might be to display setup type information in a custom setup type-related dialog. You would call FeatureSetupTypeGetData inside the switch-case statement following the call to [WaitOnDialog](#) that displays the custom dialog.



---

**Note** • *This function cannot be used with script-created feature sets.*

## Syntax

```
FeatureSetupTypeGetData ( szFeatureSource, szSetupType, nInfo, nvResult, svResult );
```

## Parameters

**Table 93** • FeatureSetupTypeGetData Parameters

Parameter	Description
<b>szFeatureSource</b>	Specifies the media name of the file media from which data associated with a setup type is to be retrieved.
<b>szSetupType</b>	Specifies the setup type name. This name must be specified exactly as it appears in the InstallShield interface—for example, "Typical".
<b>nInfo</b>	Specifies the information to retrieve. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>SETUPTYPE_INFO_DESCRIPTION</b>—Retrieves the description of the specified setup type. The description is returned in svResult.</li> <li>• <b>SETUPTYPE_INFO_DISPLAYNAME</b>—Retrieves the display name of the setup type. The name is returned in svResult.</li> </ul>
<b>nvResult</b>	Returns a NUMBER or LONG value when nInfo specifies information of that type.
<b>svResult</b>	Returns a STRING value when nInfo specifies information of that type.

## Return Values

**Table 94** • FeatureSetupTypeGetData Return Values

Return Value	Description
<b>0</b>	FeatureSetupTypeGetData was successful.
<b>&lt; 0</b>	FeatureSetupTypeGetData failed. Call FeatureError for additional information.

## FeatureSetupTypeGetData Example

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the functions FeatureSetupTypeGetData,
* FeatureGetData, FeatureSetData, SdFeatureDialog2,
* and FeatureSelectItem.
*
* Notes: To run this example script, create a project with
*        the following features (f), subfeatures (sf),
*        and components (c):
*
*        (f) Program_Files
*        (c) Program_DLLS

```

```

*           (c) Program_EXEs
*       (f) Example_Files
*           (sf) Small_Documents
*           (c) Small_Document_Examples
*           (sf) Books
*           (c) Book_Examples
*           (sf) Graphics
*           (c) Graphic_Examples
*       (f) Help_Files
*           (c) Help_Files
*       (f) Utilities
*           (sf) Grammar_Checker
*           (c) Grammar_Checker
*           (sf) Art_Studio
*           (c) Art_Studio
*
* Be sure to enter descriptions into the Description fields of
* the feature properties sheets for the Program_Files and
* Example_Files features and their subfeatures.
*
\*-----*/

// Define strings. In a real installation, you would define these in the String Editor
// view and precede each string identifier in the script with the at symbol (@).
#define FEAT_SELECT_TITLE           "Select Features"
#define FEAT_SELECT_MSG1           "IMPORTANT! Note the various feature"
#define FEAT_SELECT_MSG2           "and subfeature names, descriptions,"
#define FEAT_SELECT_MSG3           "and selection settings."
#define FEAT_SELECT_MSG4           "IMPORTANT! Note the CHANGED feature"
#define FEAT_SELECT_MSG5           "and subfeature name, description, "
#define FEAT_SELECT_MSG6           "and selection settings."
#define FEAT_PROGRAMFILES_DISPLAYNAME "Program Files"
#define FEAT_EXAMPLEFILES_DISPLAYNAME "Example Files"
#define FEAT_SMALLDOCUMENTS_DISPLAYNAME "Small Documents"
#define FEAT_BOOKS_DISPLAYNAME      "Books"
#define FEAT_GRAPHICS_DISPLAYNAME   "Graphics"
#define SETUP_TYPE                  "Typical"

// Global variable declarations.
// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_FeatureSetupTypeGetData(HWND);

function ExFn_FeatureSetupTypeGetData(hMSI)
    STRING  svInfo, szInfo, szFeature;
    NUMBER  nvInfo, nInfo, nResult;
begin

    // Get the description field data for the SETUP_TYPE setup type.
    FeatureSetupTypeGetData (MEDIA, SETUP_TYPE, SETUPTYPE_INFO_DESCRIPTION,
                             nvInfo, svInfo );

    sprintfBox (INFORMATION, "FeatureSetupTypeGetData demo",
               "FeatureSetupTypeGetData got the following " +
               "value from the " + SETUP_TYPE + " description field:\n\n%s",
               svInfo);

    // Get the description field data for the FEAT_PROGRAMFILES_DISPLAYNAME

```



```

// feature using FeatureGetData.
szFeature = FEAT_PROGRAMFILES_DISPLAYNAME;

nResult = FeatureGetData (MEDIA, szFeature, FEATURE_FIELD_DESCRIPTION,
                          nvInfo, svInfo);

sprintfBox (INFORMATION, "FeatureGetData demo",
            "FeatureGetData got the following value " +
            "from the " + FEAT_PROGRAMFILES_DISPLAYNAME +
            " description field:\n\n%s", svInfo);

// Disable the Back button in setup dialogs.
Disable (BACKBUTTON);

// Show the original description field values
// in the feature selection dialog.
SdFeatureDialog2 (FEAT_SELECT_TITLE, FEAT_SELECT_MSG1 + FEAT_SELECT_MSG2 +
                  FEAT_SELECT_MSG3, INSTALLDIR, "");

// Change the displayed names for the Program_Files feature and the
// Example_Files feature and subfeatures.
szInfo = "CHANGED Feature Name!";

nResult = FeatureSetData (MEDIA, szFeature, FEATURE_FIELD_DISPLAYNAME,
                          nInfo, szInfo);

szFeature = FEAT_EXAMPLEFILES_DISPLAYNAME;

nResult = FeatureSetData (MEDIA, szFeature, FEATURE_FIELD_DISPLAYNAME,
                          nInfo, szInfo);

szFeature = FEAT_EXAMPLEFILES_DISPLAYNAME + "\\\" +
            FEAT_SMALLDOCUMENTS_DISPLAYNAME;

nResult = FeatureSetData (MEDIA, szFeature, FEATURE_FIELD_DISPLAYNAME,
                          nInfo, szInfo);

szFeature = FEAT_EXAMPLEFILES_DISPLAYNAME + "\\\" +
            FEAT_BOOKS_DISPLAYNAME;

nResult = FeatureSetData (MEDIA, szFeature, FEATURE_FIELD_DISPLAYNAME,
                          nInfo, szInfo);

szFeature = FEAT_EXAMPLEFILES_DISPLAYNAME + "\\\" + FEAT_GRAPHICS_DISPLAYNAME;

nResult = FeatureSetData (MEDIA, szFeature, FEATURE_FIELD_DISPLAYNAME,
                          nInfo, szInfo);

// Change the descriptions displayed for the Program_Files feature
// and the Example_Files feature and subfeatures.
szFeature = FEAT_PROGRAMFILES_DISPLAYNAME;
szInfo = "CHANGED description field value!";

nResult = FeatureSetData (MEDIA, szFeature, FEATURE_FIELD_DESCRIPTION,
                          nInfo, szInfo);

szFeature = FEAT_EXAMPLEFILES_DISPLAYNAME;

nResult = FeatureSetData (MEDIA, szFeature, FEATURE_FIELD_DESCRIPTION,

```

```

        nInfo, szInfo);

    szFeature = FEAT_EXAMPLEFILES_DISPLAYNAME + "\\\" +
        FEAT_SMALLDOCUMENTS_DISPLAYNAME;

    nResult = FeatureSetData (MEDIA, szFeature, FEATURE_FIELD_DESCRIPTION,
        nInfo, szInfo);

    szFeature = FEAT_EXAMPLEFILES_DISPLAYNAME + "\\\" + FEAT_BOOKS_DISPLAYNAME;

    nResult = FeatureSetData (MEDIA, szFeature, FEATURE_FIELD_DESCRIPTION,
        nInfo, szInfo);

    szFeature = FEAT_EXAMPLEFILES_DISPLAYNAME + "\\\" + FEAT_GRAPHICS_DISPLAYNAME;

    nResult = FeatureSetData (MEDIA, szFeature, FEATURE_FIELD_DESCRIPTION,
        nInfo, szInfo);

    // Deselect the Program_Files and Example_Files features (and all their
    // subfeatures, by extension).
    FeatureSelectItem (MEDIA, FEAT_PROGRAMFILES_DISPLAYNAME, FALSE);
    FeatureSelectItem (MEDIA, FEAT_EXAMPLEFILES_DISPLAYNAME, FALSE);

    // Display the features again, noting the changed names, descriptions,
    // and selection settings.

    SdFeatureDialog2 (FEAT_SELECT_TITLE, FEAT_SELECT_MSG4 + FEAT_SELECT_MSG5 +
        FEAT_SELECT_MSG6, INSTALLDIR, "");

end;

```

## FeatureSetupTypeSet



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **FeatureSetupTypeSet** function sets the specified setup type in the file media referenced by szFeatureSource. You can use FeatureSetupTypeSet to override the selection made in a setup type dialog, such as [SdSetupTypeEx](#).



**Note** • This function cannot be used with script-created feature sets.

### Syntax

```
FeatureSetupTypeSet ( szFeatureSource, szSetupType );
```

## Parameters

**Table 95** • FeatureSetupTypeSet Parameters

Parameter	Description
<b>szFeatureSource</b>	Specifies the media name of the file media whose setup type is to be set.
<b>szSetupType</b>	Specifies which setup type to set.

## Return Values

**Table 96** • FeatureSetupTypeSet Return Values

Return Value	Description
<b>0</b>	FeatureSetupTypeSet was successful.
<b>&lt; 0</b>	FeatureSetupTypeSet failed.

## FeatureSetupTypeSet Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the FeatureSetupTypeSet function.
 *
 * Note: To run this example script, create a project (or
 *       insert into a project) with several setup types and
 *       features. Make sure you specify the setup types'
 *       default feature selections. If you do not include
 *       a Compact setup type, then define one of your setup
 *       types as SETUP_TYPE in the #define SETUP_TYPE line
 *       below.
 *
 \*-----*/

#define SETUP_TYPE    "Compact"
#define SDSETUPTITLE "Setup Type Selection"
#define SDSETUPMSG    "Select a setup type other than " + SETUP_TYPE + "."
#define SDFEATTITLE   "Feature Selection"
#define SDFEATMSG1    "Feature selection before FeatureSetupTypeSet."
#define SDFEATMSG2    "Feature selection after FeatureSetupTypeSet."
#define MSG1          "FeatureSetupTypeSet will now select all\n"
#define MSG2          "features in the " + SETUP_TYPE + " setup type."

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_FeatureSetupTypeSet(HWND);

function ExFn_FeatureSetupTypeSet(hMSI)

```

```

    STRING szSetupType, svSetup, svDir;
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Select a setup type other than SETUP_TYPE to show default
    // selection settings.
    SdSetupTypeEx (SDSETUPTITLE, SDSETUPMSG, "", svSetup, 0);

    // Display the feature selections for the selected setup type.
    svDir = INSTALLDIR;
    SdFeatureMult (SDFEATTITLE, SDFEATMSG1, svDir, "");

    MessageBox (MSG1 + MSG2, INFORMATION);

    // Now change/override the previous setup type selection by
    // selecting SETUP_TYPE's features. All others are deselected.
    szSetupType = SETUP_TYPE;
    if (FeatureSetupTypeSet (MEDIA, szSetupType) < 0) then
        MessageBox ("FeatureSetupTypeSet failed.", SEVERE);
    endif;

    // Display the new feature selections.
    SdFeatureMult (SDFEATTITLE, SDFEATMSG2, svDir, "");

end;

```

## FeatureSpendCost



**Project** • This information applies to InstallScript projects:

The **FeatureSpendCost** function tells the setup that the progress bar should be updated for a certain amount of cost that has been spent by an event external to the setup. This function has the effect of updating the progress bar appropriately as if the setup itself had spent the corresponding amount of cost. This function can only be called during a standard file operation event, such as an event resulting from calling **FeatureTransferData** or **FeatureMoveData**. Typically, this function is called during the OnInstalling or OnInstalled event as a result of an external event occurring that spends cost.



**Note** • It is not necessary to call this function for file transfer operations that the setup engine carries out, such as *XCopyFile* or *CopyFile*. These mechanisms automatically update the status bar during operation. However, *FeatureAddCost* must be called in this case (before file transfer operations begin) so the installation is aware of the additional cost.

### Syntax

```
FeatureSpendCost ( nCostHigh, nCostLow );
```

## Parameters

**Table 97** • FeatureSpendCost Parameters

Parameter	Description
<b>nCostHigh</b>	Specifies the upper 31 bits of the installation cost that should be spent.
<b>nCostLow</b>	Specifies the lower 31 bits of the installation cost that should be spent.

## Return Values

**Table 98** • FeatureSpendCost Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	Indicates that the function successfully updated the progress bar.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was unable to update the progress bar.

# FeatureSpendUninstallCost



**Project** • This function applies to InstallScript projects.

The **FeatureSpendUninstallCost** function is used to tell the setup that the progress bar should be updated for a certain amount of uninstall 'cost' that has been spent by an event external to the setup. This function has the effect of updating the progress bar appropriately as if the setup itself had spent the corresponding amount of cost. This function can only be called during a standard file operation event, such as an event resulting from calling **FeatureTransferData** or **FeatureMoveData**. Typically, this function is called during the OnInstalling or OnInstalled event as a result of an external event occurring that spends cost.



**Note** • It is not necessary to call this function for file transfer operations that are listed in the uninstall log file, such as *XCopyFile* or *CopyFile* operations. The status bar is updated automatically for items listed in the log file, note also that you do not need to call *FeatureAddUninstallCost* for items listed in the log file, as the setup automatically accounts for all items listed in the log file. However, *FeatureAddCost* must be called in this case so the file transfer operation is aware of the additional cost.

## Syntax

```
FeatureSpendUninstallCost ( nOps, nOpType);
```

## Parameters

**Table 99 • FeatureSpendUninstallCost Parameters**

Parameter	Description
<b>nOps</b>	The number of operations to spend.
<b>nOpType</b>	<p>Indicates the type of operation. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>FEATURE_OPCOST_UNINSTALL_FILE</b>—Specifies the operation as uninstalling a file. You can also specify this constant using the value 4096. This enables you to experiment with different sizes to determine what size to use for your custom operation.</li> <li>• <b>FEATURE_OPCOST_UNINSTALL_REGORINI</b>—Specifies the operation as uninstalling a registry file. You can also specify this constant using the value 2048. This enables you to experiment with different sizes to determine what size to use for your custom operation.</li> <li>• <b>FEATURE_OPCOST_UNINSTALL_UNREGFILE</b>—Specifies the operation as unregistering a file. You can also specify this constant using the value 16384. This enables you to experiment with different sizes to determine what size to use for your custom operation.</li> </ul>



**Note** • You can also pass a numeric value in this parameter to indicate an operation of a specific cost. You may need to experiment with different values to determine the appropriate value for a specified custom operation. Note that the total uninstall cost added is *nOps* multiplied by *nOpType*.

## Return Values

**Table 100 • FeatureSpendUninstallCost Parameters**

Return Value	Description
<b>ISERR_SUCCESS</b>	Indicates that the function successfully updated the progress bar.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was unable to update the progress bar.

# FeatureStandardSetupTypeSet



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **FeatureStandardSetupTypeSet** function sets the current setup type to the standard setup type specified by *nSetupType*. This function attempts to set the setup type by calling the **FeatureSetupTypeSet** function with the appropriate string to set the setup type.

## Syntax

```
FeatureStandardSetupTypeSet ( szFeatureSource, nSetupType );
```

## Parameters

**Table 101** • FeatureStandardSetupTypeSet Parameters

Parameter	Description
<b>szFeatureSource</b>	Specifies the media name of the file media whose setup type is to be set.
<b>nSetupType</b>	<p>Specifies which setup type to set. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• TYPICAL</li> <li>• COMPLETE</li> <li>• COMPACT</li> <li>• CUSTOM</li> </ul> <p>You can customize the string setup type value associated with each numeric constant by customizing the global predefined script variables listed below. The function sets the setup or installation type by trying the appropriate script variable, then tries the setup type name in English and Japanese (see below). Also, when the COMPLETE setup type is specified, the function tries all TYPICAL values after checking the values for COMPLETE. If the setup type cannot be set to any of these values, the function returns failure. The following global predefined script variables can be customized:</p> <ul style="list-style-type: none"> <li>• <b>SETUPTYPE_STR_TYPICAL</b>—Defines the setup type that will be set by default if TYPICAL is specified.</li> <li>• <b>SETUPTYPE_STR_COMPLETE</b>—Defines the setup type that will be set by default if COMPLETE is specified.</li> <li>• <b>SETUPTYPE_STR_COMPACT</b>—Defines the setup type that will be set by default if COMPACT is specified.</li> <li>• <b>SETUPTYPE_STR_CUSTOM</b>—Defines the setup type that will be set by default if CUSTOM is specified.</li> </ul>

The following table shows the values the FeatureStandardSetupTypeSet function uses when each constant is specified in the nSetupType parameter.

**Table 102** • nSetupType Constants

Constant	Strings Used
<b>TYPICAL</b>	SETUPTYPE_STR_TYPICAL Typical
<b>COMPLETE</b>	SETUPTYPE_STR_COMPLETE Complete

**Table 102** • nSetupType Constants (cont.)

Constant	Strings Used
<b>COMPACT</b>	SETUPTYPE_STR_COMPACT Compact
<b>CUSTOM</b>	SETUPTYPE_STR_CUSTOM Custom

## Return Values

**Table 103** • FeatureStandardSetupTypeSet Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	FeatureStandardSetupTypeSet was able to set the specified setup type.
<b>&lt; ISERR_SUCCESS</b>	FeatureStandardSetupTypeSet was unable to set the specified setup type.

# FeatureTotalSize



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **FeatureTotalSize** function returns the total size—in bytes—of the features referenced by szFeature.

- To include subfeatures in the size calculation, set bIncludeSubcomp to TRUE.
- To get the total size of all the features in the specified media, set szFeature to a null string ("" ) and set bIncludeSubcomp to TRUE.



**Note** • There might be a difference between the value returned by FeatureTotalSize and the space required values displayed in the SdFeatureDialog, SdFeatureDialog2, SdFeatureDialogAdv, and SdFeatureMult dialogs. This difference is due to the fact that this function does not take into account whether any components associated with the feature are currently filtered by operating system or language, or the cluster size on the target drive. To obtain a drive space calculation that takes these factors into account, call FeatureGetCost.

## Syntax

```
FeatureTotalSize ( szFeatureSource, szFeature, bIncludeSubcomp, bTargetSize );
```



## Parameters

**Table 104** • FeatureTotalSize Parameters

Parameter	Description
<b>szFeatureSource</b>	Specifies the name of the file media from which the total size of selected features is to be returned.
<b>szFeature</b>	Specifies the name of the feature whose size is to be retrieved. To retrieve the size of the entire media, pass a null string ("") in this parameter.
<b>bIncludeSubcomp</b>	Indicates whether to include selected subfeatures of szFeature. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>TRUE</b>—Include selected subfeatures in the size calculation.</li> <li>• <b>FALSE</b>—Do not include subfeatures in the size calculation.</li> </ul>
<b>bTargetSize</b>	Indicates whether to retrieve the original, uncompressed size or the size in the media library. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>TRUE</b>—Retrieve the original, uncompressed size.</li> <li>• <b>FALSE</b>—Retrieve the size in the media library.</li> </ul>

## Return Values

**Table 105** • FeatureTotalSize Return Values

Return Value	Description
<b>XXXX</b>	The total size, in bytes, of the selected features.
<b>&lt; 0</b>	FeatureTotalSize failed. Call <a href="#">FeatureError</a> for additional information.

## FeatureTotalSize Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the FeatureListItems, SdFeatureMult, and
 * FeatureTotalSize functions.
 *
 * Comments: To run this example script, create a project with
 *           the following features (f), subfeatures (sf),
 *           and components (c):
 *
 *           (f) Program_Files
 *               (c) Program_DLLS
 *               (c) Program_EXEs
 *           (f) Example_Files
 *               (sf) Small_Documents

```

```

*                (c) Small_Document_Examples
*            (sf) Books
*                (c) Book_Examples
*            (sf) Graphics
*                (c) Graphic_Examples
*        (f) Help_Files
*            (c) Help_Files
*        (f) Utilities
*            (sf) Grammar_Checker
*                (c) Grammar_Checker
*            (sf) Art_Studio
*                (c) Art_Studio
*
\*-----*/

#define FEAT_SELECT_TITLE    "Select Features"
#define FEAT_SELECT_MSG      "Select features and subfeatures to install."
#define FEATTOTSIZEMSG1      "Want to change feature selections and see\n"
#define FEATTOTSIZEMSG2      "size change reflected in FeatureTotalSize call?"

    // Global variable declarations.

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_FeatureTotalSize(HWND);

function ExFn_FeatureTotalSize(hMSI)
    STRING  szDir, svString;
    NUMBER  nResult, nDone;
    LIST    listCompList, listTemp;
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Create a string list of all top-level features.
    listCompList = ListCreate (STRINGLIST);
    FeatureListItems (MEDIA, "", listCompList);

    // Display the string list of top-level features.
    SdShowInfoList ("List MEDIA Features", "MEDIA contains " +
        "the following top-level features:", listCompList);

    // Get each top-level feature in listCompList, in turn, and
    // list and display all of its subfeatures, if any.
    nResult = ListGetFirstString ( listCompList, svString );
    while ( nResult != END_OF_LIST )
        listTemp = ListCreate (STRINGLIST);
        FeatureListItems (MEDIA, svString, listTemp);
        SdShowInfoList ("Subfeature Listing", svString + " contains " +
            "the following subfeatures:", listTemp);
        ListDestroy (listTemp);
        nResult = ListGetNextString (listCompList, svString);
    endwhile;

    // Show feature selection dialog and total size of all selected
    // features.  Loop to change selections and see total size change
    // reflected in the call to FeatureTotalSize.

```

```

nDone = YES;
while (nDone = YES)
    szDir = INSTALLDIR;
    SdFeatureMult (FEAT_SELECT_TITLE, FEAT_SELECT_MSG, szDir, "" );

    nResult = FeatureTotalSize(MEDIA, "", TRUE, TRUE);
    sprintfBox (INFORMATION, "", "Total size of all files " +
        "in SELECTED features:\n\n%d", nResult);
    nDone = AskYesNo (FEATTOTSIZMSG1 + FEATTOTSIZMSG2, YES);
endwhile;

ListDestroy (listComplList);

end;

```

## FeatureTransferData



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*—if the *InstallScript* user interface (UI) style is the traditional style (which uses the *InstallScript* engine as an external UI handler)

*This information does not apply to InstallScript MSI projects in which the InstallScript UI style is the new style (which uses the InstallScript engine as an embedded UI handler). To learn more, see Using the InstallScript Engine as an External vs. Embedded UI Handler for InstallScript MSI Installations.*

If you use an event-based script, the **FeatureTransferData** function is called automatically after the OnFirstUIBefore event and interacts with the OnMoving, OnMoved, and feature events. FeatureTransferData installs or uninstalls features appropriately based on their selection state and whether they are currently installed.

**FeatureTransferData** does the following:

- **FeatureTransferData** installs features that are selected (for example, by end user selections in feature or setup type dialogs) and are currently not installed.
- **FeatureTransferData** uninstalls features that are not selected and are currently installed.

The installation determines whether a feature is currently installed by reading the existing log file during installation initialization. If no valid log file is found during installation initialization, all features are considered to be not installed.

When you call **FeatureTransferData**, Do(SELFREGISTRATIONPROCESS); is called automatically after the files are installed but before the call returns. Therefore, if your installation needs to perform additional actions after the file transfer but before self-registration, place these actions in the OnMoved event; the OnMoved event is called after the file transfer but before the batch self-registration occurs.

### Syntax

```
FeatureTransferData ( szFeatureSource );
```

## Parameters

**Table 106** • FeatureTransferData Parameters

Parameter	Description
<b>szFeatureSource</b>	Specifies the media name of the file media whose features have been specified for installation and uninstallation. Typically, you would pass the system variable <a href="#">MEDIA</a> in this parameter.

## Return Values

**Table 107** • FeatureTransferData Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully performed the feature installations and uninstallations.
<b>&lt; 0</b>	Indicates that the function could not perform the feature installations and uninstallations.

## Additional Information

You can call one of the following functions before calling **FeatureTransferData** to affect the result of the call:

- **FeatureReinstall**—Reinstalls all features that are currently selected when **FeatureTransferData** is called.
- **FeatureRemoveAll**—Removes (uninstalls) all features when **FeatureTransferData** is called.

# FeatureUpdate



**Project** • This information applies to InstallScript project.

The **FeatureUpdate** function causes the next call to [FeatureTransferData](#) (or [FeatureMoveData](#)) to reinstall all features that are already installed when FeatureTransferData is called, except the maintenance/uninstallation feature. (Note that this feature is automatically placed in your .cab files by the media builder and is not displayed in InstallShield.)

## Syntax

```
FeatureUpdate ( szReserved );
```

## Parameters

**Table 108** • FeatureUpdate Parameters

Parameter	Description
<b>szReserved</b>	Pass a null string ("") in this parameter. No other value is allowed.

## Return Values

**Table 109** • FeatureUpdate Return Values

Return Value	Description
<b>0</b>	Indicates that the function succeeded.
<b>&lt; 0</b>	Indicates that the function failed.

## Additional Information

Call FeatureUpdate in an installation whose maintenance/uninstallation feature you do not want to be used during subsequent maintenance operations (modifying, repairing, or uninstalling). FeatureUpdate is typically called in an update or add-on installation, in which the installation media does not contain all the elements of the previous media and thus should not be used for subsequent maintenance operations.

FeatureUpdate is similar to [FeatureReinstall](#), but FeatureReinstall also reinstalls the maintenance/uninstallation feature.



**Note** • If you call both *FeatureUpdate* and *FeatureReinstall*, the call to *FeatureUpdate* will not have any effect; so if you call *FeatureUpdate*, do not call *FeatureReinstall* in the same installation.

Call *FeatureUpdate* to update only applications that were installed with the same version of *InstallShield* as the update installation. If you call *FeatureUpdate* to update an application that was installed with a previous version of *InstallShield*, the maintenance/uninstallation files will not be updated, so any subsequent maintenance operations will use the maintenance/uninstallation files that were created with the previous version.

The following information applies in this situation: A previously released version of your application was installed by an installation that aborts if a newer version of the application is present. In this case, *FeatureUpdate* should not be called in an update installation that changes the version number of the installed application. Otherwise, after the update installation runs, the currently installed application cannot be uninstalled.

# FeatureValidate



**Project** • This information applies to *InstallScript* projects.



**Note** • This function cannot be used with script-created feature sets.

The **FeatureValidate** function validates the password of the file media library or of a specified feature.

## Syntax

```
FeatureValidate ( szMediaLibrary, szFeature, szPassword );
```

## Parameters

**Table 110** • FeatureValidate Parameters

Parameter	Description
<b>szMediaLibrary</b>	Specifies the name of the file media library whose password is to be validated.
<b>szFeature</b>	Specifies the name of the feature. If this parameter is a null string (""), the entire media library is assumed.
<b>szPassword</b>	Specifies the password to be validated.

## Return Values

**Table 111** • FeatureValidate Return Values

Return Value	Description
<b>0</b>	FeatureValidate was successful.
<b>&lt; 0</b>	FeatureValidate failed. Call <a href="#">FeatureError</a> for additional information.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

## FeatureValidate Example

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the functions SdSetupTypeEx, SdFeatureDialog,
* FeatureIsItemSelected, FeatureGetData, and FeatureValidate.
*
* Notes: To run this example script, create a project with
*        the following features (f), subfeatures (sf),
*        and components (c):
*
*          (f) Program Files
*              (c) Program DLLS
*              (c) Program EXEs
*          (f) Example Files
*              (sf) Small Documents
*                  (c) Small Document Examples
*              (sf) Books
*                  (c) Book Examples
```

```

*           (sf) Graphics
*           (c) Graphic Examples
*       (f) Help Files
*           (c) Help Files
*       (f) Utilities
*           (sf) Grammar Checker
*           (c) Grammar Checker
*           (sf) Art Studio
*           (c) Art Studio
*       (f) Evaluation Copy
*           (c) Evaluation Copy
*           (c) Help Files
*
* Insert "dummy" files into the components. Make sure you
* define the correct file name for the main EXE (MAIN_EXE,
* below) that you insert into the Program EXEs components.
* Run the setup with and without a password assigned to the
* Program Files feature (remember to rebuild your media
* each time).
*
* You can also create billboards (name them Bbrd1.bmp,
* Bbrd2.bmp, and Bbrd3.bmp) and add them to the project in
* the Support Files/Billboards's Language Independent folder.
*
* This example script installs files, adds an icon to the
* Start Programs menu, and provides uninstallation
* functionality.
*
\*-----*/

// Define strings. In a real installation, you would define these in the String Editor
// view and precede each string identifier in the script with the at symbol (@).
#define FEATURE_SELECT_TITLE           "Select Features"
#define FEATURE_SELECT_MSG             "Select features and subfeatures to install."
#define FEATURE_PROGRAMFILES_DISPLAYNAME "Program Files"
#define PASSWORD_PROMPT                "Please enter the password."
#define PASSWORD_ERRMSG                "Password incorrect. Please enter again."

// Global variable declarations.
STRING  svData, svLogFile, szProgram, szFeature, svResult, svSetupType,
svDir;
BOOL    bInitStepsDone, bPwdValid;
NUMBER  nvData, nResult;

#include "ifx.h"

function OnFirstUIBefore()
begin
    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Get the setup type.
    svDir = TARGETDIR;
    SdSetupTypeEx (SETUPTYPE_TITLE, SETUPTYPE_MSG, "", svSetupType, 0);

    // If user selected Custom setup type, display feature selection dialog.
    if (svSetupType = SETUPTYPE_CUSTOM) then
        SdFeatureDialog (FEATURE_SELECT_TITLE, FEATURE_SELECT_MSG, svDir, "");

```

```

endif;

// Enable the Back button in setup dialogs.
Enable (BACKBUTTON);

// If the Program Files feature is selected and there is a password
// associated with it, the user must input the password and
// it must be validated.
nResult = FALSE;
nvData = FALSE;
nResult = FeatureIsItemSelected (MEDIA, FEATURE_PROGRAMFILES_DISPLAYNAME);
FeatureGetData (MEDIA, FEATURE_PROGRAMFILES_DISPLAYNAME,
    FEATURE_FIELD_PASSWORD, nvData, svData);
if (nResult && nvData) then
    bPwdValid = FALSE;
    Disable (BACKBUTTON); // Back button not needed or supported here.
    while (!bPwdValid)
        AskText (PASSWORD_PROMPT, "", svResult);
        nResult = FeatureValidate (MEDIA, FEATURE_PROGRAMFILES_DISPLAYNAME,
            svResult);
        if (nResult = 0) then
            bPwdValid = TRUE;
        else
            MessageBox (PASSWORD_ERRMSG, SEVERE);
        endif;
    endwhile;
    Enable (BACKBUTTON); // Restore Back button's default status.
endif;
end;

```

## FileCompare

The **FileCompare** function compares the size, modify dates, contents, or versions of two files.

### Syntax

```
FileCompare ( szFileName1, szFileName2, nCompareFlag );
```



## Parameters

**Table 112 • FileCompare Parameters**

Parameter	Description
<b>szFileName1</b>	Specifies the name of the first file to compare. The installation uses the system variable <b>SRCDIR</b> as the path of the file.
<b>szFileName2</b>	Specifies name of the second file to compare. The installation uses one of the following system variables as the path of the file: <ul style="list-style-type: none"> <li>• <b>TARGETDIR</b> (in an InstallScript installation)</li> <li>• <b>INSTALLDIR</b> (in an Basic MSI or InstallScript MSI installation)</li> </ul>
<b>nCompareFlag</b>	Specifies comparison options. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>COMPARE_SIZE</b>—Compares the size of the two files.</li> <li>• <b>COMPARE_DATE</b>—Compares the modify dates of the two files.</li> <li>• <b>COMPARE_VERSION</b>—Compares the version resource of the two files.</li> <li>• <b>COMPARE_MD5_SIGNATURE</b>—Compares the MD5 signature of the two files. If the MD5 signatures match, the contents of the files are identical.</li> </ul>

## Return Values (when **COMPARE\_MD5\_SIGNATURE** is specified)

**Table 113 • FileCompare Return Values (COMPARE MD5\_SIGNATURE)**

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	The function successfully compared the files, specifically: <ul style="list-style-type: none"> <li>• <b>EQUALS</b>—The files MD5 signatures are the same (the files contents are the same).</li> <li>• <b>NOT_EQUALS</b>—The files MD5 signatures are not the same.</li> </ul>
<b>&lt;= ISERR_SUCCESS</b>	The function failed to compare the files.

## Return Values (when **COMPARE\_SIZE**, **COMPARE\_DATE**, or **COMPARE\_VERSION** is specified)

**Table 114 • FileCompare Return Values (COMPARE\_SIZE, COMPARE\_DATE, or COMPARE\_VERSION)**

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	The function successfully compared the files, specifically: <ul style="list-style-type: none"> <li>• <b>EQUALS</b>—The first file's date, size, or version is equal to that of the second file.</li> <li>• <b>GREATER_THAN</b>—The first file is newer or larger than the second file.</li> <li>• <b>LESS_THAN</b>—The first file is older or smaller than the second file.</li> </ul>

**Table 114** • FileCompare Return Values (COMPARE\_SIZE, COMPARE\_DATE, or COMPARE\_VERSION) (cont.)

Return Value	Description
<= ISERR_SUCCESS	The function failed to compare the files.

## FileCompare Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the FileCompare function.
 *
 * This example script calls FileCompare three times:
 *
 * -- The first call compares the version of FILE_COMP1 in SDIR
 *    to the version of FILE_COMP2 in TDIR.
 *
 * -- The second call checks if FILE_COMP1 was created earlier
 *    than FILE_COMP2.
 *
 * -- The third call checks if FILE_COMP1 is smaller than
 *    FILE_COMP2.
 *
 * Note: Before you run this script, set the preprocessor
 *       constants so that they reference existing files in
 *       existing directories on the target computer.
 *
 \*-----*/

#define SDIR      "C:\\\"
#define TDIR      "C:\\\"
#define FILE_COMP1 "Example1.dll"
#define FILE_COMP2 "Example2.dll"
#define MSG_TITLE "FileCompare Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_FileCompare(HWND);

function ExFn_FileCompare(hMSI)
    NUMBER nResult;
begin

    /*-----*\
     *
     * Compare the versions of the two files.
     *
     \*-----*/

```

```

nResult = FileCompare (SDIR ^ FILE_COMP1, TDIR ^ FILE_COMP2,
                      COMPARE_VERSION);

// Report an error and terminate if either file is not found.
if (nResult = FILE_NOT_FOUND) then
    sprintfBox (INFORMATION, MSG_TITLE, "%s and/or %s not found.",
              FILE_COMP1, FILE_COMP2);

    abort;
endif;

// Files are present so report the result of the comparison.
switch (nResult)
    case EQUALS:
        sprintfBox (INFORMATION, MSG_TITLE, "%s is the same version as %s.",
                  FILE_COMP1, FILE_COMP2);
    case GREATER_THAN:
        sprintfBox (INFORMATION, MSG_TITLE, "%s is a newer version than %s.",
                  FILE_COMP1, FILE_COMP2);
    case LESS_THAN:
        sprintfBox (INFORMATION, MSG_TITLE, "%s is an older version than %s.",
                  FILE_COMP1, FILE_COMP2);
    case OTHER_FAILURE:
        sprintfBox (INFORMATION, MSG_TITLE,
                  "Version information not available in %s and/or %s.",
                  FILE_COMP1, FILE_COMP2);

endswitch;

/*-----*\
*
* Compare the creation dates.
*
\*-----*/

nResult = FileCompare (SDIR ^ FILE_COMP1, TDIR ^ FILE_COMP2,
                      COMPARE_DATE);

switch (nResult)
    case LESS_THAN:
        sprintfBox (INFORMATION, MSG_TITLE,
                  "%s was created earlier than %s.",
                  FILE_COMP1, FILE_COMP2);
    case GREATER_THAN:
        sprintfBox (INFORMATION, MSG_TITLE,
                  "%s was created earlier than %s.",
                  FILE_COMP2, FILE_COMP1);
    case EQUALS:
        sprintfBox (INFORMATION, MSG_TITLE,
                  "%s and %s have the same creation date and time.",
                  FILE_COMP1, FILE_COMP2);
    default:
        sprintfBox (INFORMATION, MSG_TITLE,
                  "Unable to compare the creation date and time of %s and %s.",
                  FILE_COMP2, FILE_COMP1);

endswitch;

/*-----*\
*
* Compare the file sizes.
*

```

```

\*-----*/

nResult = FileCompare (SDIR ^ FILE_COMP1, TDIR ^ FILE_COMP2,
    COMPARE_SIZE);

switch (nResult)
    case LESS_THAN:
        sprintfBox (INFORMATION, MSG_TITLE,
            "%s is smaller than %s.",
            FILE_COMP1, FILE_COMP2);
    case GREATER_THAN:
        sprintfBox (INFORMATION, MSG_TITLE,
            "%s is smaller than %s.",
            FILE_COMP2, FILE_COMP1);
    case EQUALS:
        sprintfBox (INFORMATION, MSG_TITLE,
            "%s and %s are the same size.",
            FILE_COMP1, FILE_COMP2);
    default:
        sprintfBox (INFORMATION, MSG_TITLE,
            "Unable to compare the size of %s and %s.",
            FILE_COMP1, FILE_COMP2);
endswitch;

end;

```

## FileDeleteLine

The **FileDeleteLine** function deletes a range of lines (including the starting line and ending line) from a text file using a starting and ending line number. This function works on line-oriented text files—it does not work with binary files. You can use FileDeleteLine with the FileGrep function to search and delete text lines in a file.



**Caution** • The file specified by *szFileName* must not already have been opened by a call to *OpenFile*. If *szFileName* is already open, call *CloseFile* before calling *FileDeleteLine*.

### Syntax

```
FileDeleteLine ( szFileName, nStartLineNum, nEndLineNum );
```

## Parameters

**Table 115** • FileDeleteLine Parameters

Parameter	Description
<b>szFileName</b>	Specifies the unqualified name of a file located in the directory specified by the system variable SRCDIR, or the fully qualified path to the file. The lines specified by nStartLineNum and nEndLineNum will be deleted from that file if it exists.
<b>nStartLineNum</b>	Specifies the number of the first line to delete from the file. Note that line numbering begins at 0.
<b>nEndLineNum</b>	Specifies the number of the last line to delete from the file. Note that line numbering begins at 0. To delete from the line specified by nStartLineNum to the end of the file, pass the predefined constant DELETE_EOF in this parameter.

## Return Values

**Table 116** • FileDeleteLine Return Values

Return Value	Description
<b>0</b>	FileDeleteLine successfully deleted the specified lines from the file.
<b>&lt; 0</b>	FileDeleteLine failed because of one of the following conditions: <ul style="list-style-type: none"> <li>• <b>FILE_NOT_FOUND (-2)</b>—The installation could not find the file in szFileName.</li> <li>• <b>FILE_RD_ONLY (-5)</b>—The file is write-protected.</li> <li>• <b>LINE_NUMBER (-7)</b>—One of the line numbers you specified is less than zero, or the line number does not exist in the file.</li> <li>• <b>OUT_OF_DISK_SPACE (-6)</b>—There is insufficient space on the disk drive to complete the specified operation.</li> <li>• <b>OTHER_FAILURE (-1)</b>—Some other unspecified error has occurred.</li> </ul>

## FileDeleteLine Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the FileDeleteLine function.
*
* This script searches a file for the first line that includes
* the string "PATH". If a line with that string is found, it
```

```

* is deleted. Finally, a new line is added to the file at the
* position of the deleted line. If a line with the word
* "PATH" was not found, the new line is inserted at the top of
* the file.
*
* Note: Before running this script, create a batch file
*       named ISExempl.bat in the root of drive C. For
*       best effect, that file should include PATH command.
*
\*-----*/

#define SDIR          "C:\\\"
#define EXAMPLE_BAT  "ISExempl.bat"
#define TITLE        "FileDeleteLine Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_FileDeleteLine(HWND);

function ExFn_FileDeleteLine(hMSI)
    STRING szSearchStr, svReturnLine, szMsg;
    NUMBER nvResult, nvLineNum;
begin

    // Set up the search string parameter for FileGrep.
    szSearchStr = "PATH";

    // Find the search string in the specified file.
    nvResult = FileGrep (SDIR ^ EXAMPLE_BAT, szSearchStr,
                        svReturnLine, nvLineNum, RESTART);

    switch(nvResult)
    case FILE_NOT_FOUND:
        // Report error; then terminate.
        MessageBox (EXAMPLE_BAT + " not found.", WARNING);
        abort;
    case FILE_LINE_LENGTH:
        // Report error; then terminate.
        MessageBox (EXAMPLE_BAT + "lines too long.", WARNING);
        abort;
    case OTHER_FAILURE:
        // Report error; then terminate.
        MessageBox (EXAMPLE_BAT + "Unknown failure on call to FileGrep.",
                    WARNING);
        abort;
    case END_OF_FILE:
        // Report that the search string was not found.
        szMsg = "\"%s\" not found in %s.";
        sprintfBox (INFORMATION, TITLE, szMsg, szSearchStr, EXAMPLE_BAT);

        // Set the line number parameter for FileInsertLine.
        nvLineNum = 0;
    case 0:
        // Delete the line with the search string.
        if (FileDeleteLine (EXAMPLE_BAT, nvLineNum, nvLineNum) < 0) then
            MessageBox ("Failed on call to FileDeleteLine.", SEVERE);
            abort;
        else

```

```

        // Report the deletion.
        szMsg = "\"%s\" found in line %d of %s:\n\n%s\n\nLine deleted. ";
        sprintfBox (INFORMATION, TITLE, szMsg, szSearchStr, nvLineNum,
                    EXAMPLE_BAT, svReturnLine);
    endif;
endswitch;

// Set up the new string parameter for FileInsertLine.
szNewString = "PATH=C:\\Windows\\Bin;C:\\Bin;C:\\Ishield;";

// Insert the new string.
if (FileInsertLine (EXAMPLE_BAT, szNewString, nvLineNum, BEFORE) < 0) then
    // Report an error.
    MessageBox ("Failed on call to FileInsertLine.", SEVERE);
else
    // Report success.
    szMsg = "The following string was inserted as line %d of %s:\n\n%s";
    sprintfBox (INFORMATION, TITLE, szMsg, nvLineNum, EXAMPLE_BAT,
                szNewString);
endif;

end;

```

## FileGrep

The **FileGrep** function searches a text file for a specified string. If the string is found, the line containing that string is returned in `svReturnLine` and the number of the line is returned in `nvLineNumber`. The search is not case-sensitive. FileGrep works on line-oriented text files; it will not work with binary files.




---

**Note** • It is not necessary to open a file before searching it with FileGrep; nor do you need to close it after the call to FileGrep. File open and file close are performed automatically by FileGrep. Although FileGrep will perform successfully in most cases on a file that is already open as a result of a previous call to OpenFile, it will return `FILE_NOT_FOUND` if the file was opened in append mode.

### Syntax

```
FileGrep ( szFileName, szSearchStr, svReturnLine, nvLineNumber, nFlag );
```

## Parameters

**Table 117 • FileGrep Parameters**

Parameter	Description
<b>szFileName</b>	Specifies the fully qualified name of the file to search.
<b>szSearchStr</b>	Specifies the search string.
<b>svReturnLine</b>	Returns the line in which szSearchStr was found.
<b>nvLineNumber</b>	Return the number of the line in which szSearchStr was found. Line numbering starts at 0.
<b>nFlag</b>	Specifies search options. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>CONTINUE</b>—Retrieves the next occurrence (if any) of the search string.</li> <li>• <b>RESTART</b>—Retrieves the first instance of the search string.</li> </ul>

## Return Values

**Table 118 • FileGrep Return Values**

Return Value	Description
<b>0</b>	FileGrep found the specified string.
<b>&lt; 0</b>	FileGrep failed because of one of the following conditions: <ul style="list-style-type: none"> <li>• <b>END_OF_FILE (-4)</b>—The end of file was reached without finding the search string.</li> <li>• <b>FILE_NOT_FOUND (-2)</b>—InstallShield was unable to find the file in szFileName.</li> <li>• <b>FILE_LINE_LENGTH (-3)</b>—The line exceeds the maximum length allowed.</li> <li>• <b>OTHER_FAILURE (-1)</b>—An unspecified error has occurred.</li> </ul>

## FileGrep Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the FileGrep function.
*
* FileGrep is called to search a file for the first line that
* contains the word "PATH". The results are displayed in a
* message box. Note that the FileGrep function is not case
* sensitive.
```



```

*
* Note: Before running this script, create a batch file
*       named ISExempl.bat in the root of drive C. For
*       best effect, that file should include PATH command.
*
\*-----*/

#define SOURCE_DIR "C:\\\"
#define SOURCE_FILE "ISExempl.bat"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_FileGrep(HWND);

function ExFn_FileGrep(hMSI)
    STRING svLine, szNewString, svReturnLine, szMsg;
    NUMBER nvLineNumber, nvResult;
begin

    // Find the search string in the source file.
    nvResult = FileGrep (SOURCE_DIR ^ SOURCE_FILE, "PATH", svReturnLine,
                        nvLineNumber, RESTART);

    switch(nvResult)
    case FILE_NOT_FOUND:
        // Report error; then abort.
        MessageBox( SOURCE_FILE + " not found.", WARNING);
        abort;
    case FILE_LINE_LENGTH:
        // Report error; then abort.
        MessageBox (SOURCE_FILE + "lines too long.", WARNING);
        abort;
    case OTHER_FAILURE:
        // Report error; then abort.
        MessageBox (SOURCE_FILE + "Unknown failure on call to FileGrep.",
                    WARNING);
        abort;
    endswitch;

    // Loop until end of file.
    while (nvResult != END_OF_FILE)

        // Set up message string for sprintfBox.
        szMsg = "'PATH' found in line %d of %s:\n\n'%s'";

        // Report matching line from file.
        sprintfBox (INFORMATION, "FileGrep", szMsg, nvLineNumber, SOURCE_FILE,
                    svReturnLine);

        // Search again.
        nvResult = FileGrep (SOURCE_DIR ^ SOURCE_FILE, "PATH", svReturnLine,
                            nvLineNumber, CONTINUE);

    endwhile;

end;

```

# FileInsertLine

The **FileInsertLine** function inserts or replaces a line using line numbers. You can use FileInsertLine with FileGrep, which finds lines and returns their line numbers.

FileInsertLine works on line-oriented text files with lines that are no longer than 1,024 bytes. With InstallShield, a line longer than the maximum allowed length indicates a binary file and causes FileInsertLine to fail.



---

**Caution** • *The file specified by szFileName must not already have been opened by a call to OpenFile. If szFileName is already open, call CloseFile before calling FileInsertLine.*

## Syntax

```
FileInsertLine (szFileName, szInsertLine, nLineNumber, nInsertFlag);
```

## Parameters

**Table 119** • FileInsertLine Parameters

Parameter	Description
<b>szFileName</b>	Specifies the unqualified name of a file located in the directory specified by the system variable SRCDIR, or the fully qualified path to the file. The value of szInsertLine is inserted into that file if it exists.
<b>szInsertLine</b>	Specifies the string to insert into the file.
<b>nLineNumber</b>	Specifies the line number at which you want to insert szInsertLine. The first line number is 0.
<b>nInsertFlag</b>	Specifies an insertion location option. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>BEFORE</b>—Inserts the line before nLineNumber.</li> <li>• <b>AFTER</b>—Inserts the line after nLineNumber.</li> <li>• <b>APPEND</b>—Appends szInsertLine to the line indicated by nLineNumber.</li> <li>• <b>REPLACE</b>—Replaces the line indicated by nLineNumber with szInsertLine.</li> </ul>

## Return Values

**Table 120** • FileInsertLine Return Values

Return Value	Description
<b>0</b>	FileInsertLine successfully inserted the line into the specified file.
<b>&lt; 0</b>	FileInsertLine failed because of one of the following conditions: <ul style="list-style-type: none"> <li>• <b>FILE_LINE_LENGTH (-3)</b>—Indicates that the length of the line exceeds the maximum length allowed for text files.</li> <li>• <b>FILE_NOT_FOUND (-2)</b>—FileInsertLine was unable to find the file in szFileName.</li> <li>• <b>FILE_RD_ONLY (-5)</b>—Indicates that the file is write-protected.</li> <li>• <b>LINE_NUMBER (-7)</b>—Indicates that one of the line numbers you specified is less than zero, or the line number does not exist in the file.</li> <li>• <b>OUT_OF_DISK_SPACE (-6)</b>—Indicates that there is insufficient space on the disk drive to complete the specified operation.</li> <li>• <b>OTHER_FAILURE (-1)</b>—An unspecified error has occurred.</li> </ul>

## FileInsertLine Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the FileInsertLine function.
*
* The AskText dialog is displayed to obtain a line of text
* from the user. This text is then inserted as the first line
* in the text file specified by TARGET_FILE.
*
* FileInsertLine is then called again to append the same text
* to the first line, leaving two copies of the same text in
* the first line of the file.
*
* Note: Before running this script, create a batch file named
*       ISExempl.bat in the root of drive SRCDIR.
*
\*-----*/

#define TARGET_FILE  "ISExempl.bat"
#define TITLE        "FileInsertLine Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_FileInsertLine(HWND);

function ExFn_FileInsertLine(hMSI)
    STRING szMsg, svText;
begin

    // Set up message parameter for call to AskText.
    szMsg = "Please enter a line to insert into EXAMPLE.BAT.";

    // Get line to add to file.
    AskText (szMsg, "", svText);

    // Insert the text as the first line of the specified file.
    if (FileInsertLine (TARGET_FILE, svText, 0, BEFORE) < 0) then
        MessageBox ("FileInsertLine failed.", SEVERE);
    else
        // Set up message parameter for call to sprintfBox.
        szMsg = "'%s' successfully inserted as first line of %s.";

        // Display message.
        sprintfBox (INFORMATION, TITLE, szMsg, svText, TARGET_FILE);
    endif;

    // Append the same string to the same line.
    if (FileInsertLine(TARGET_FILE, svText, 0, APPEND) < 0) then
        MessageBox("FileInsertLine failed.", SEVERE);
    else
        // Set up message parameter for call to sprintfBox.
        szMsg = "'%s' successfully appended to first line of %s.";

```

```

        // Display message.
        sprintfBox(INFORMATION, TITLE, szMsg, svText, TARGET_FILE);
    endif;

end;

```

## FindAllDirs

The **FindAllDirs** function searches an entire hierarchical disk or directory structure starting with the specified directory, and it returns a string list of subdirectory names. You can use FindAllDirs to find either subdirectories of a certain directory, or you can use it to find all the directories on a disk.


If nOp is INCLUDE\_SUBDIR, the search starts at the directory specified by szDir and continues searching the subdirectory structure. If the specified directory is a root directory and nOp contains INCLUDE\_SUBDIR, all the directory names on the entire disk are returned.

### Syntax

```
FindAllDirs ( szDir, nOp, listDirs );
```

## Parameters

**Table 121 • FindAllDirs Parameters**

Parameter	Description
<b>szDir</b>	Specifies the name of the directory to search.   <b>Note</b> • If the directory is enclosed in quotation marks, FindAllDirs fails. To ensure that the folder name is not enclosed in quotation marks, call <a href="#">LongPathToQuote</a> (szPath, FALSE) before calling FindAllDirs.
<b>nOp</b>	Specifies whether or not to search all of the subdirectories below szDir. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"><li>• <b>EXCLUDE_SUBDIR</b>—Excludes subdirectories.</li><li>• <b>INCLUDE_SUBDIR</b>—Includes subdirectories.</li></ul>
<b>listDirs</b>	Returns a list of fully qualified directory names. The string list identified by listDirs must already have been initialized by a call to <a href="#">ListCreate</a> .

## Return Values

**Table 122 • FindAllDirs Return Values**

Return Value	Description
<b>0</b>	FindAllDirs successfully generated the list of subdirectory names.
<b>&lt; 0</b>	FindAllDirs function was unable to generate a list.

## FindAllDirs Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* This example demonstrates the FindAllDirs function.
*
* FindAllDirs is called to retrieve all directories located
* in a specified directory. Subdirectories are included at
* the user's discretion.
*
\*-----*/

#define TITLE "FindAllDirs Example"
```

```

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_FindAllDirs(HWND);

function ExFn_FindAllDirs(hMSI)
    LIST listDirs;
    STRING svSearchPath, szMsg;
    NUMBER nOp, nResult;
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Ask the user for a path.
    AskPath ("Enter an existing path.", "", svSearchPath);

    // Ask whether or not to include subdirectories.
    if (AskYesNo ("Include subdirectories?", YES) = YES) then
        nOp = INCLUDE_SUBDIR;
        szMsg = "Directories and Subdirectories";
    else
        nOp = EXCLUDE_SUBDIR;
        szMsg = "Directories only";
    endif;

    // Display a message while building the list.
    SdShowMsg ("Searching . . . please wait.", TRUE);

    // Create a STRING list for directory names.
    listDirs = ListCreate (STRINGLIST);

    // Find requested elements place them into the list.
    nResult = FindAllDirs (svSearchPath, nOp, listDirs);

    // Close the message box.
    SdShowMsg ("", FALSE);

    if ( nResult< 0) then
        // Report no matches.
        sprintfBox (INFORMATION, TITLE, "No directories in %s",
                    svSearchPath);
    else
        // Display the list.
        SdShowInfoList (TITLE, szMsg, listDirs);
    endif;

end;

```

## FindAllFiles

The **FindAllFiles** function searches an entire hierarchical subdirectory structure starting with the specified directory, and it returns the name of the first file with a particular file specification. If the specified directory is the root directory, InstallShield searches the entire disk. The function stops at the first matching file name it finds.



**Note** • If the argument passed to *nOp* is *RESET*, InstallShield starts searching at the directory specified in the parameter *szDir* and continues searching the subdirectory structure until it finds a file matching *szFileName*. If *nOp* equals *CONTINUE*, the search continues where it finished the last time the function was called. Call this function repeatedly to find all occurrences of files that match *szFileName*.

The first time you call this function to begin a new search, set *nOp* to *RESET*. You can continue to search for all other occurrences of the specified file by setting *nOp* to *CONTINUE* and placing the function call in a loop that ends when the *FindAllFiles* function fails.

## Finding Files that Match a File Specification



### Task

#### To find all files matching a file specification:

1. Assign to *szDir* the name of the folder to search.
2. Assign to *szFileName* the file specification, such as *IS5\*.txt*.
3. Call **FindAllFiles** with *nOp* set to *RESET*.
4. While the return code from **FindAllFiles** is 0, save the file name in *svResult*; then call **FindAllFiles** with *nOp* set to *CONTINUE*.
5. Call **FindAllFiles** with *nOp* set to *CANCEL*.



**Caution** • You cannot use the *XCOPYFile* function within a *FindAllFiles(..., RESET)*, and *FindAllFiles(..., CONTINUE)* loop. If you call *XCOPYFile* inside a *FindAllFiles* loop, the file name returned by *FindAllFiles(..., CONTINUE)* may be incorrect.

## Syntax

```
FindAllFiles ( szDir, szFileName, svResult, nOp );
```



## Parameters

**Table 123** • FindAllFiles Parameters

Parameter	Description
<b>szDir</b>	Specifies the name of the directory to search.
<b>szFileName</b>	Specifies the file specification to search for. Wild-card characters are allowed in this parameter.
<b>svResult</b>	Returns the fully qualified name of the first matching file if found. If <b>FindAllFiles</b> fails, svResult remains unchanged.
<b>nOp</b>	Indicates search options. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>● <b>CONTINUE</b>—Resumes the search at the location where the previous search stopped.</li> <li>● <b>RESET</b>—Starts the search at the beginning of the directory specified by szDir.</li> <li>● <b>CANCEL</b>—Frees all files and folders that were accessed by previous calls to <b>FindAllFiles</b>. Call <b>FindAllFiles</b> with this parameter to ensure the success of subsequent file and folder operations on files and folders that have been accessed by <b>FindAllFiles</b>.</li> </ul>

## Return Values

**Table 124** • FindAllFiles Return Values

Return Value	Description
<b>0</b>	Indicates that the function retrieved and returned a file that matched the specification.
<b>&lt; 0</b>	Indicates that the function was unable to find a file that matched the specifications.

## FindAllFiles Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the FindAllFiles function.
*
* This script gets a directory name and filespec from the
* user. Then it calls FindAllFiles repeatedly to build a
* list of files that are located in the specified directory
```

```

* and whose names match the filespec. Finally, the list of
* matching files is displayed in a listbox.
*
\*-----*/

#define TITLE_TEXT "FindAllFiles Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_FindAllFiles(HWND);

function ExFn_FindAllFiles(hMSI)
    STRING szMsg, svDir, svFileSpec, svMatchingFileName, svNumFiles;
    NUMBER nResult, nNumFiles;
    LIST listFiles;
begin

selectdir:

    // Set up parameters for call to SelectDir.
    szMsg = "Select a directory to search.";
    svDir = "";

    // Select a search directory.
    SelectDir (TITLE_TEXT, szMsg, svDir, FALSE);

askfile:
    szMsg = "Enter a file specification to search for in " + svDir + ":";

    // Get a file specification from the user.
    if (AskText (szMsg , "*.*", svFileSpec) = BACK) then
        goto selectdir;
    endif;

    // Create a string list for the file listing.
    listFiles = ListCreate (STRINGLIST);

    if listFiles = LIST_NULL then
        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Set the file count to zero.
    nNumFiles = 0;

    // Show a message while the file list is being built.
    SdShowMsg ("Searching . . . ", TRUE);

    // Get the first file that matches the file spec.
    nResult = FindAllFiles (svDir, svFileSpec, svMatchingFileName, RESET);

    while(nResult = 0)
        // Add the file to the list.
        if ListAddString (listFiles, svMatchingFileName, AFTER) < 0 then
            MessageBox ("Unable to build complete file list", WARNING);
            goto showmatches;
        endif;

```

```

        // Increment the file counter.
        nNumFiles = nNumFiles + 1;

        // Find the next matching file name.
        nResult = FindAllFiles(svDir, svFileSpec, svMatchingFileName, CONTINUE);
    endwhile;

showmatches:

    // Free all files and folders accessed by FindAllFiles. If your
    // setup does not target the Windows NT platform, this step is
    // not necessary.
    FindAllFiles(svDir, svFileSpec, svMatchingFileName, CANCEL);

    // Convert the file count to a string for display.
    NumToStr(svNumFiles, nNumFiles);

    // Clear the message that displayed while the file list was being built.
    SdShowMsg("", FALSE);

    // Display the files that match the file specification.
    szMsg = "Number of matching files : " + svNumFiles;
    if (SdShowInfoList(TITLE_TEXT, szMsg, listFiles) = BACK) then
        ListDestroy(listFiles);
        goto askfile;
    endif;

    // Remove the list from memory.
    ListDestroy(listFiles);

end;

```

## FindFile

The **FindFile** function searches a directory for a specified file. InstallShield returns the first matching file in the parameter svResult.




---


**Note** • This function searches only the specified subdirectory. It does not search an entire disk or directory tree.

### Syntax

```
FindFile ( szPath, szFileName, svResult );
```

## Parameters

**Table 125 • FindFile Parameters**

Parameter	Description
<b>szPath</b>	Specifies the name of the directory to search. Subdirectories beneath this directory are <i>not</i> searched.   <b>Note</b> • If the directory is enclosed in quotation marks, FindFile fails. To ensure that the folder name is not enclosed in quotation marks, call <a href="#">LongPathToQuote</a> (szPath, FALSE) before calling FindFile.
<b>szFileName</b>	Specifies the name of the file to search for. Wild-card characters are allowed in this parameter.
<b>svResult</b>	Returns the name of the first file that matches szFileName. This parameter contains the unqualified file name; that is, the drive designation and path are not included.

## Return Values

**Table 126 • FindFile Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully found and returned the specified file.
<b>&lt; 0</b>	Indicates that the function was unable to find the file.

## FindFile Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the FindFile function.
*
* FindFile is called to search for the file Config.sys in the
* root of drive C.
*
\*-----*/

#define FILE_SPEC "Config.sys"
#define SEARCH_DIR "C:\\\"
```

```

#define TITLE_TEXT "FindFile Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_FindFile(HWND);

function ExFn_FindFile(hMSI)
    STRING svResult;
begin

    if (FindFile (SEARCH_DIR, FILE_SPEC, svResult) < 0) then
        MessageBox ("FindFile failed.", SEVERE);
    else
        sprintfBox (INFORMATION, TITLE_TEXT, "Found: %s in %s.", svResult,
            SEARCH_DIR);
    endif;

end;

```

## FindWindow

The **FindWindow** function provides a way for advanced developers to get a handle to a window by specifying its window class and window name. If you know the class and window name of an application, you can get its handle. You can then use that handle to send messages directly to the window.




---

**Tip** • To find the class and name of a window, run the Microsoft Spy.exe program.

### Syntax

```
FindWindow ( szClassName, szWinName );
```

## Parameters

**Table 127 • FindWindow Parameters**

Parameter	Description
<b>szClassName</b>	Specifies the name of the class to which the window belongs. To specify “any class,” pass a null string in this parameter.
<b>szWinName</b>	Specifies the window caption (title). To return the handle of the topmost window in the specified class, pass a null string (“”) in this parameter.

## Return Values

**Table 128 • FindWindow Return Values**

Return Value	Description
<b>XXXX</b>	The handle of the window.
<b>NULL (0)</b>	FindWindow was unable to find a window with the specified name and class name.

## FindWindow Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the FindWindow and SendMessage functions.
*
* This script launches Windows Notepad and then calls
* FindWindow to locate the Notepad window. Next, it calls
* SendMessage to maximize the window; after a three-second
* delay, it calls SendMessage again to minimize the
* window. When the script ends, Windows NotePad remains
* open but minimized. Note that the parameters passed to
* SendMessage are Windows system messages whose values
* are defined as constants in this script.
*
* Note: Before running this script, set the preprocessor
*       constant NOTEPAD so that it references the fully-
*       qualified name of the Windows Notepad executable.
*
\*-----*/

#define NOTEPAD "C:\\Windows\\Notepad.exe"

// Include Ifx.h for built-in InstallScript function prototypes.
```

```

#include "Ifx.h"

export prototype ExFn_FindWindow(HWND);

function ExFn_FindWindow(hMSI)
    NUMBER nMsg, nwParam, nIParam;
    HWND nHwnd;
begin

    // Do not display the setup's background window.
    Disable (BACKGROUND);

    // Open the Windows Notepad.
    if (LaunchApp (NOTEPAD, "") < 0 ) then
        MessageBox ("Unable to launch Notepad.", SEVERE);
        abort;
    endif;

    // Wait three seconds so we can view the window before
    // it's maximized.
    Delay (3);

    // Retrieve the handle of the Notepad window. The first
    // parameter is the window class. A null string in the
    // second parameter specifies the topmost Notepad window.
    nHwnd = FindWindow ("Notepad", "");

    if (nHwnd = NULL) then
        MessageBox ("Unable to find the Notepad window.", SEVERE);
    else
        // Send system command to maximize the window.
        SendMessage (nHwnd, WM_SYSCOMMAND, SC_MAXIMIZE, 0);

        // Wait three seconds so we can view the window
        // before it's minimized.
        Delay (3);

        // Send system command to minimize the window.
        SendMessage (nHwnd, WM_SYSCOMMAND, SC_MINIMIZE, nIParam);
    endif;

end;

```

## FormatMessage

The **FormatMessage** function provides the error message text associated with a large error code returned by a built-in InstallScript function.

### Syntax

```
FormatMessage ( nErrorReturnCode );
```

## Parameters

**Table 129** • FormatMessage Parameters

Parameter	Description
<b>nErrorReturnCode</b>	Pass a large error code—for example, 2147024891 (0x80070005)—that was returned by a built-in InstallScript function. Passing the error code -1 will not produce useful results.

## Return Value

A string containing the error message text associated with the error code nErrorReturnCode.

## FormatMessage Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
//-----
//
//  InstallShield Example Script
//
//  Demonstrates the FormatMessage function.
//
//  To demonstrate, deliberately call XCopyFile with a nonexistent source
//  directory, which should cause the function to fail. To provide system
//  details about the error, call FormatMessage and display the message
//  string in a MessageBox.
//
//-----

function OnBegin( )
    NUMBER nReturn;
begin

    // call XCopyFile on a nonexistent directory
    nReturn = XCopyFile("C:\\no_such_directory", "C:\\destination", COMP_NORMAL);

    // when XCopyFile fails, display the error message and exit the installer
    if (nReturn < 0) then
        MessageBox(FormatMessage(nReturn), SEVERE);
        abort;
    endif;

end;
```



# GetAndAddAllFilesCost

The **GetAndAddAllFilesCost** function determines the cost of all of the files that are in szSrcDir and that match the wild-card pattern that is identified by szWildcard; the function adds this cost to the current value of nvCostHigh and/or nvCostLow. This cost can then be passed to **FeatureAddCost**.



---

**Note** • Note that **GetAndAddAllFilesCost** does not actually set any information to be used directly by the installation. You must call **FeatureAddCost** (as appropriate) after calling this function to add the additional cost to an existing feature.

## Syntax

```
GetAndAddAllFilesCost (szSrcDir, szWildcard, szTargetDir, nClusterSize, nvInstallCostHigh,  
    nvInstallCostLow, nvUninstallCost);
```

## Parameters

**Table 130** • GetAndAddAllFilesCost Parameters

Parameter	Description
<b>szSrcDir</b>	The fully qualified path of the source location containing the files to determine the cost of.
<b>szWildcard</b>	The wild card to match files against. Note that the function returns failure if no files that match szWildcard are found.
<b>szTargetDir</b>	If nClusterSize is 0, the fully qualified path that the file is being installed to. This path is used to determine the cluster size of the target drive. If nClusterSize is non-zero, this parameter is ignored.
<b>nClusterSize</b>	Specifies the cluster size of the target drive. If this parameter is 0, the function determines this information from szTargetDir.
<b>nvInstallCostHigh</b>	The upper 31 bits of the install cost (in bytes) of this file is added to the current value of this variable.
<b>nvInstallCostLow</b>	The lower 31 bits of the install cost (in bytes) of this file is added to the current value of this variable.
<b>nvUninstallCost</b>	The uninstall cost of these files is added to the current value of this variable. This value equals 1 per file to be uninstalled. Therefore, 100 files to be uninstalled returns a value of 100.

## Return Values

**Table 131** • GetAndAddAllFilesCost Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	Indicates that the function was successful.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was not successful.

# GetAndAddFileCost

The **GetAndAddFileCost** function determines the cost of the specified file and adds it to the current value of nvCostHigh and/or nvCostLow. This allows you to calculate and add up the cost of multiple files by calling the function multiple times in a loop. Set nvCostHigh and nvCostLow to zero before calling the function to determine the cost of a single file. This function is typically used when you need to determine the cost of an existing file on disk so this cost can then be passed to FeatureAddCost.



**Note** • Note that this function does not actually set any information to be used directly by the installation. You must call *FeatureAddCost* (as appropriate) after calling this function to add the additional cost to an existing feature.

## Syntax

```
GetAndAddFileCost ( szSrcFile, szTargetDir, nClusterSize, nvCostHigh, nvCostLow );
```

## Parameters

**Table 132** • GetAndAddFileCost Parameters

Parameter	Description
<b>szSrcFile</b>	The fully qualified path and file name of the existing file on disk to determine the cost of.
<b>szTargetDir</b>	If nClusterSize is 0, the target folder for the file. This path is used to determine the cluster size of the target drive. If nClusterSize is non-zero, this parameter is ignored.
<b>nClusterSize</b>	Specifies the cluster size of the target drive. If this parameter is 0, the function determines this information from szTargetDir.
<b>nvCostHigh</b>	The upper 31 bits of the installation cost (in bytes) of this file is added to the current value of this variable.
<b>nvCostLow</b>	The lower 31 bits of the installation cost (in bytes) of this file is added to the current value of this variable.

## Return Values

**Table 133** • GetAndAddFileCost Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	Indicates that the function was successful.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was not successful.

# GetArrayFromISArray

The **GetArrayFromISArray** function returns a pointer to an array of pointers that point to the actual data of the specified array. This function does not allocate any additional memory, but it returns a pointer to the data in the existing array. If vArray is a string array, the returned pointer can be passed to functions that take LPCWSTR\* or LPWSTR\* arguments; if vArray is a numeric array, the returned pointer can be passed to functions that take LPCDWORD\* or LPDWORD\* arguments.

## Syntax

```
GetCharArrayFromISArray (vArray);
```

## Parameters

**Table 134** • GetCharArrayFromISArray Parameters

Parameter	Description
<b>vArray</b>	Specifies the array to which you want a pointer.

## Return Values

**Table 135** • GetCharArrayFromISArray Return Values

Return Value	Description
<b>pointer</b>	A pointer to an array of pointers that point to the actual data of the specified array.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function failed.

## Additional Information

Be careful when using the returned pointer to modify strings that are contained in the array. The length of the strings that are contained in string arrays are managed internally by the installation. Therefore, if you change the length of a string, the installation is no longer able to manage the data that are contained in the string array.

# GetCharArrayFromISStringArray

The **GetCharArrayFromISStringArray** function returns a pointer to an array of pointers to ANSI character strings that corresponds to the wide character strings that are contained in the specified array.

## Syntax

```
GetCharArrayFromISStringArray ( vArray );
```

## Parameters

**Table 136** • GetCharArrayFromISStringArray Parameters

Parameter	Description
<b>vArray</b>	Specifies the string array to which you want a pointer.

## Return Values

**Table 137** • GetCharArrayFromISStringArray Return Values

Return Value	Description
<b>pointer</b>	A pointer to an array of pointers to ANSI character strings.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function failed.

## Additional Information

The **GetCharArrayFromISStringArray** function allocates additional memory for the array of pointers and the ANSI character strings. This pointer can be passed to functions that take LPCSTR\* or LPSTR\* arguments. After you are done with the newly created array, call [DeleteCharArray](#) to delete the array from memory.

If you call [CopyCharArrayToISStringArray](#) to write the data from the array of pointers back to the original string array, be careful when modifying strings that are contained in the array. Since the lengths of the strings that are contained in string arrays are managed internally by the installation, if you change the length of a string the entire string will not be copied back to the original array when you call **CopyCharArrayToISStringArray**.

# GetCurrentDialogName



**Project** • This information applies to InstallScript projects.

The **GetCurrentDialogName** function retrieves the name of the currently displayed dialog as it was specified in the call to [EzDefineDialog](#) when the dialog was defined.

This information can be used to close the dialog by calling [EndDialog](#).

## Syntax

```
GetCurrentDialogName ( svDialogName );
```

## Parameters

**Table 138** • GetCurrentDialogName Parameters

Parameter	Description
<b>svDialogName</b>	Returns the name of the currently displayed dialog, or a null string (""), if no dialog is currently displayed.

## Return Values

**Table 139** • GetCurrentDialogName Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	GetCurrentDialogName successfully retrieved the name of the currently displayed dialog.
<b>&lt; ISERR_SUCCESS</b>	No dialog is currently displayed.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

# GetCurrentDir

The **GetCurrentDir** function returns the current directory.



**Note** • When you are specifying a file in your script, always specify the full path (using the appropriate *InstallShield* system variable, for example, *SRCDIR*) rather than depend on the current folder having the appropriate value. The script internally executes code that can change the current folder, so its value may not be what you expect.

## Syntax

```
GetCurrentDir( svCurrentDir );
```

## Parameters

**Table 140** • GetCurrentDir Parameters

Parameter	Description
<b>svCurrentDir</b>	Returns the current directory.

## Return Values

**Table 141** • GetCurrentDir Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the function successfully retrieved the current directory.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was unable to retrieve the current directory.

# GetDir

The **GetDir** function removes the drive designation from the fully qualified path or file name passed in `szPath` and returns the remainder of the path or file name in `svDir`. The path must include a drive designation. It may be a UNC path.

In the following example, the fully qualified path `C:\Windows` is returned in `svDir` as `\Windows`.

```
GetDir("C:\\Windows", svDir);
```

In the next example, the UNC path `\\TheServer\\TheSharedDevice\\Programs` is returned in `svDir` as `\\Programs`.

```
GetDir("\\\\TheServer\\TheSharedDevice\\Programs", svDir);
```

## Syntax

```
GetDir ( szPath, svDir );
```

## Parameters

**Table 142** • GetDir Parameters

Parameter	Description
<b>szPath</b>	Specifies a path that includes a drive designation.
<b>svDir</b>	Returns the path without the drive designation. If szPath is a UNC path, GetDir returns the path without the server name and shared device name.

## Return Values

**Table 143** • GetDir Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully returned a path without a drive designation.
<b>&lt; 0</b>	Indicates that the function was unable to return a path without a drive designations.

## GetDir Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the GetDir function.
 *
 * This script gets a fully qualified directory name from the
 * user. Next, it calls GetDir to return the selected directory
 * name without the drive designation. The resulting path is
 * then displayed.
 *
 \*-----*/

#define TITLE_TEXT "GetDir example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_GetDir(HWND);

function ExFn_GetDir(hMSI)
    STRING szMsg, svSelectedDir, svDirNameOnly;
begin

```



```

// Get a fully qualified directory name from the user.
AskPath ("Select a directory.", INSTALLDIR, svSelectedDir);

// Get the directory name minus the drive designation.
if (GetDir (svSelectedDir, svDirNameOnly) < 0) then
    // Report the error.
    MessageBox ("GetDir failed.", SEVERE);
else
    // Display the directory name as it was returned by GetDir.
    sprintfBox (INFORMATION, TITLE_TEXT,
        "Selected directory without drive designation: %s",
        svDirNameOnly);
endif;

end;

```

## GetDisk

The **GetDisk** function extracts the disk drive designation from the fully qualified path or file name specified by szPath and returns it in svDisk.

### Syntax

```
GetDisk ( szPath, svDisk );
```

## Parameters

**Table 144** • GetDisk Parameters

Parameter	Description
<b>szPath</b>	Specifies a fully qualified path or file name that includes a drive designation. If a drive designation is not included, GetDisk will fail. The value passed in szPath may be a UNC path.
<b>svDisk</b>	Returns the drive designation (which includes the colon). If szPath is a UNC path, GetDisk returns the server name and shared device name in the format <b>\\server\sharedevice</b> .

## Return Values

**Table 145** • GetDisk Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully returned the drive designation.
<b>&lt; 0</b>	Indicates that the function was unable to return the drive designation.

## GetDisk Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the GetDisk function.
 *
 * This script gets a fully qualified directory name from
 * the user. Next, it calls GetDisk to return the disk drive
 * designation. The drive designation is then displayed.
 *
 \*-----*/

#define TITLE_TEXT "GetDisk example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_GetDisk(HWND);

function ExFn_GetDisk(hMSI)
    STRING svSelectedDir, svDisk;
begin

```

```

// Get a fully qualified directory name from the user.
AskPath ("Select a directory.", INSTALLDIR, svSelectedDir);

// Get the drive designation from the selected directory name.
if (GetDisk (svSelectedDir, svDisk) < 0) then
    // Report the error.
    MessageBox ("GetDir failed.", SEVERE);
else
    // Display the drive designation as it was returned by GetDisk.
    sprintfBox (INFORMATION, TITLE_TEXT, "Disk drive: %s", svDisk);
endif;

end;

```

## GetDiskInfo

The **GetDiskInfo** function gets information about a specified disk drive. By inspecting the values assigned to members of this variable, your script can determine the information about a disk drive.

### Syntax

```
GetDiskInfo ( _DISK_INFO& pdi );
```

### Parameters

**Table 146** • GetDiskInfo Parameters

Parameter	Description
<b>_DISK_INFO&amp; pdi</b>	Specifies a pointer to an existing _DISK_INFO structure (allocated by the installation before calling the function) that specifies the information to retrieve and contains the specified information after the function returns.

The following table shows the meaning of each \_DISK\_INFO member:

**Table 147** • \_DISK\_INFO Members

Member	Meaning
<b>szDiskPath[_MAX_PATH]</b>	Specifies the fully qualified path of the drive to return the information for.

**Table 147** • \_DISK\_INFO Members (cont.)

Member	Meaning
<b>nInfoToQuery</b>	<p>Specifies the information to query. Possible values are the following:</p> <ul style="list-style-type: none"> <li>• <b>DISK_INFO_QUERY_ALL</b>—Queries all information about the disk drive.</li> <li>• <b>DISK_INFO_QUERY_BYTES_PER_CLUSTER</b>—Queries the bytes per cluster (cluster-size) of the drive.</li> <li>• <b>DISK_INFO_QUERY_DISK_TOTAL_SPACE</b>—Queries the total space on the drive.</li> <li>• <b>DISK_INFO_QUERY_DISK_FREE_SPACE</b>—Queries the free space on the drive.</li> <li>• <b>DISK_INFO_QUERY_DRIVE_TYPE</b>—Queries the drive type.</li> </ul>
<b>nBytesPerCluster</b>	Returns the number of bytes per cluster if DISK_INFO_QUERY_BYTES_PER_CLUSTER was specified in nInfoToQuery.
<b>nTotalSpaceHigh</b>	Returns the upper 31 bits of the total space on the target drive.
<b>nTotalSpaceLow</b>	Returns the lower 31 bits of the total space on the target drive.
<b>nFreeSpaceHigh</b>	Returns the upper 31 bits of the free space on the target drive.
<b>nFreeSpaceLow</b>	Returns the lower 31 bits of the free space on the target drive.
<b>nDriveType</b>	<p>Returns the drive type using the same constants returned by the Windows API GetDriveType. For convenience, the following constants are predefined:</p> <ul style="list-style-type: none"> <li>• <b>DRIVE_UNKNOWN</b>—The drive type is unknown.</li> <li>• <b>DRIVE_NO_ROOT_DIR</b>—The drive type is defined as no root directory.</li> <li>• <b>DRIVE_REMOVABLE</b>—The drive type is defined as removable.</li> <li>• <b>DRIVE_FIXED</b>—The drive type is defined as fixed.</li> <li>• <b>DRIVE_REMOTE</b>—The drive type is defined as remote.</li> <li>• <b>DRIVE_CDROM</b>—The drive type is defined as a CD-ROM.</li> <li>• <b>DRIVE_RAMDISK</b>—The drive type is defined as a RAM DISK.</li> </ul>
<b>nResultDiskSpace</b>	<p>This member contains valid information on return when any of the following flags are specified:</p> <ul style="list-style-type: none"> <li>• DISK_INFO_QUERY_BYTES_PER_CLUSTER</li> <li>• DISK_INFO_QUERY_DISK_TOTAL_SPACE</li> <li>• DISK_INFO_QUERY_DISK_FREE_SPACE</li> </ul> <p>This member returns the result of querying the information.</p>

## Return Values

**Table 148** • GetDiskInfo Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	Indicates that the function was successful.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was unsuccessful. If a null pointer is passed to the function, this value is returned.

You can check the `nResultDiskSpace` member to determine whether the query was successful.

## GetDiskInfo Example



**Note** • To call this function in a Basic MSI installation, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the GetDiskInfo function.
 *
 * This script gets the amount of free disk space on a Windows drive
 * and displays that amount in a message box.
 *
 \*-----*/

function OnBegin( )
    _DISK_INFO di;
    NUMBER n; // to do: something with function return values
    NUMBER nvSizeTargetHigh, nvSizeTargetLow, nUnitsTarget;
begin

    // init. _DISK_INFO members: what drive, what info
    di.szDiskPath = WINDISK;
    di.nInfoToQuery = DISK_INFO_QUERY_DISK_FREE_SPACE;

    n = GetDiskInfo(&di);

    // change to desired unit for free space
    nUnitsTarget = MBYTES;

    n = ConvertSizeToUnits(
        di.nFreeSpaceHigh, di.nFreeSpaceLow, BYTES,
        nvSizeTargetHigh, nvSizeTargetLow, nUnitsTarget);

    sprintfBox(INFORMATION, "Free Space",
        "Free space: %d MB",
        nvSizeTargetLow);

end;

```

# GetDiskSpace

This function is obsolete. Use the [GetDiskInfo](#) function instead.

The **GetDiskSpace** function returns the amount of free space—in bytes—on the drive or path specified in szPath.

## Syntax

```
GetDiskSpace ( szDrive );
```

## Parameters

**Table 149** • GetDiskSpace Parameters

Parameter	Description
<b>szPath</b>	Specifies the path for which the function returns the amount of space available. The value specified can be a UNC path.

## Return Values

**Table 150** • GetDiskSpace Return Values

Return Value	Description
<b>&gt; 0</b>	The number of bytes free in the specified directory. The maximum value returned is 2 GB. Free disk space greater than that returns as 2 GB. Call GetDiskSpaceEx when your setup needs to check for free space greater than 2 GB.
<b>&lt; 0</b>	Indicates that GetDiskSpace was unable to obtain the amount of free space.

## GetDiskSpace Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the GetDiskSpace function.
*
* This script gets a fully qualified path from the end user.
* It then extracts the drive designation from the path, gets
* the amount of free space on that drive, and displays the
* amount of free space in a message box.
*
* A script-defined function is used to insert commas if
* necessary into the number before it's displayed.
*
```

```

\*-----*/

    // User-defined function to format a number in a string.
    prototype FormatIntString (BYREF STRING);

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_GetDiskSpace(HWND);

function ExFn_GetDiskSpace(hMSI)
    STRING svResultPath, svDrive;
    LONG   lFreeSpace;
    STRING svFreeSpace;
begin
    // Prompt for a target path; use C as the default.
    AskPath ("Select drive:", "C:\\", svResultPath);

    // Get the drive designation from the path.
    GetDisk (svResultPath, svDrive);

    // Get the amount of free disk space on that drive.
    lFreeSpace = GetDiskSpace (svDrive);

    if (lFreeSpace < 0) then
        // Handle an error from GetDiskSpace.
        MessageBox ("GetDiskSpace failed.", SEVERE);
    else
        // Convert the free disk space value to a string.
        NumToStr (svFreeSpace, lFreeSpace);

        // Insert commas into the number if necessary.
        FormatIntString (svFreeSpace) ;

        // Report the amount of free space.
        MessageBox (svFreeSpace + " bytes free on drive " + svDrive + ". ",
                    INFORMATION);
    endif;
end;

function FormatIntString(svInteger)
// Insert commas if necessary into an integer
// that's been stored in a string variable.
    INT nLen;
    STRING svSubStr, svTemp;
begin
    nLen = StrLength (svInteger);

    if nLen > 3 then
        nLen = nLen - 3;
        StrSub (svTemp , svInteger, nLen, 3);

        while nLen > 3
            nLen = nLen - 3;
            StrSub (svSubStr, svInteger, nLen, 3);

```

```
        svTemp = svSubStr + "," + svTemp;
    endwhile;

    StrSub (svSubStr, svInteger, 0,nLen);
    svInteger = svSubStr + "," + svTemp;
endif;
end;
```

## GetDiskSpaceEx

This function is obsolete. Use the [GetDiskInfo](#) function instead.

The **GetDiskSpaceEx** function returns the amount of free space on the specified path. The value passed in nUnits determines whether the value returned by GetDiskSpaceEx is a measure of bytes, kilobytes, megabytes, or gigabytes.

### Syntax

```
GetDiskSpaceEx ( szDrive, nUnits );
```



## Parameters

**Table 151** • GetDiskSpaceEx Parameters

Parameter	Description
<b>szPath</b>	Specifies the path for which the function returns the amount of space available. The value specified can be a UNC path.
<b>nUnits</b>	<p>Pass one of the following predefined constants to indicate the measurement unit:</p> <ul style="list-style-type: none"> <li>• <b>BYTES</b>—Indicates that GetDiskSpaceEx should return the number of free bytes.</li> <li>• <b>KBYTES</b>—Indicates that GetDiskSpaceEx should return the number of free kilobytes.</li> <li>• <b>MBYTES</b>—Indicates that GetDiskSpaceEx should return the number of free megabytes.</li> <li>• <b>GBYTES</b>—Indicates that GetDiskSpaceEx should return the number of free gigabytes.</li> </ul>

## Return Values

**Table 152** • GetDiskSpaceEx Return Values

Return Value	Description
<b>&gt; 0</b>	The number of free bytes, kilobytes, megabytes, or gigabytes in the specified directory, depending on the value of nUnits.
<b>&lt; 0</b>	Indicates that GetDiskSpaceEx was unable to obtain the amount of free space.

## Additional Information

The limit of GetDiskSpaceEx is 2 giga-units (2<sup>31</sup>) of the measurement unit you pass in nUnits. When you specify BYTES, the limit is 2 GB; when you specify KBYTES, the limit is 2 TB; and so on. You should specify KBYTES for most setups.

GetDiskSpaceEx returns the available space rounded down to the nearest single unit specified in nUnits. For example, if you specify GBYTES, a path that has 2.5 GB available will return 2. For setups that require more precise space information, call the Windows API function GetDiskFreeSpaceEx directly.

## GetDiskSpaceEx Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the GetDiskSpaceEx function.
 *
 * This script gets a fully qualified path from the end user.
 * It then extracts the drive designation from the path, gets
 * the amount of free space on that drive, and displays the
 * amount of free space in a message box.
 *
 \*-----*/

STRING    szPath;

          INT        iResult;

#include "ifx.h"

program

    // Prompt for a target path.

    szPath = PROGRAMFILES;
    iResult = SdAskDestPath
    ("Select Folder\nSelect the folder to check for available disk space.", " ", szPath, 0);

    if (iResult < 0) then
        MessageBox("Unable to display dialog.", SEVERE);
    endif;

    // Get the amount of free disk space (in kilobytes) for the specified path.

    iResult = GetDiskSpaceEx(szPath, KBYTES);

    if (iResult < 0) then
        MessageBox("Unable to get available disk space.", SEVERE);
    endif;

    // Display the amount of space available.

    SprintfBox(INFORMATION, "Space Available", "%d KB available on %s.", iResult, szPath);

endprogram

```

## GetEnvVar

The **GetEnvVar** function retrieves the current value of an environment variable.

### Syntax

```
GetEnvVar ( szParameter, svValue );
```

## Parameters

**Table 153** • GetEnvVar Parameters

Parameter	Description
<b>szParameter</b>	Specifies the name of the environment variable whose value is to be retrieved.
<b>svValue</b>	Returns the current value of the environment variable.

## Return Values

**Table 154** • GetEnvVar Return Values

Return Value	Description
<b>0</b>	Indicates that the function retrieved the value of the environment variable.
<b>&lt; 0</b>	Indicates that the function was unable to retrieve the value of the environment variable.

## GetEnvVar Example

This script is structured for use in a Basic MSI installation. To call this function in a Basic MSI installation, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release. To run this code in an InstallScript or InstallScript MSI installation, copy only the code that is in between the begin and end statements and paste it into your script's OnBegin event handler function.



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the GetEnvVar function.
*
* GetEnvVar is called to get the value of the environment
* variable named TEMP.
*
\*-----*/

#define TITLE_TEXT "GetEnvVar Example"

#define ENV_TEMP "TEMP"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_GetEnvVar(HWND);

```

```

function ExFn_GetEnvVar(hMSI)
    STRING svEnvVar;
begin

    // Get the value of the environment variable specified by ENV_TEMP.
    if (GetEnvVar (ENV_TEMP, svEnvVar) < 0) then
        // Report the error.
        MessageBox ("Environment variable " + ENV_TEMP + " not found.", SEVERE);
    else
        // Display the value of the environment variable.
        sprintfBox (INFORMATION, TITLE_TEXT, "%s = %s", ENV_TEMP, svEnvVar);
    endif;

end;

```

## GetExtendedErrInfo




---

**Project** • This information applies to InstallScript projects.

The **GetExtendedErrInfo** function returns the error information that was set by SetExtendedErrInfo. Many InstallScript functions call SetExtendedErrInfo(\_\_FILE\_\_, \_\_LINE\_\_, nError) internally.

### Syntax

```
GetExtendedErrInfo ( svScriptFile, nvLineNumber, nvError );
```

## Parameters

**Table 155** • GetExtendedErrInfo Parameters

Parameter	Description
<b>svScriptFile</b>	Returns the script file in which the error occurred.
<b>nvLineNumber</b>	Returns the line number in which the error occurred.
<b>nvError</b>	Returns the error code for the error.

## Return Values

**Table 156** • GetExtendedErrInfo Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	The function successfully retrieved the error information.
<b>&lt; ISERR_SUCCESS</b>	The function failed to retrieve the error information.

# GetExtents

The **GetExtents** function retrieves the dimensions of a screen—in pixels. The width of the screen is returned in nvDx and the height is returned in nvDy. For example, a standard VGA monitor returns 640 in nvDx and 480 in nvDy.



**Note** • The screen dimensions of the target machine can be gotten natively through the Windows Installer service by using the ScreenX and ScreenY properties.

## Syntax

```
GetExtents ( nvDx, nvDy );
```

## Parameters

**Table 157** • GetExtents Parameters

Parameter	Description
<b>nvDx</b>	Returns the width of the screen in pixels.
<b>nvDy</b>	Returns the height of the screen in pixels.

## Return Values

**Table 158** • GetExtents Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully retrieved the dimensions of a screen.
<b>&lt; 0</b>	Indicates that the function was unable to retrieve the values.

## GetExtents Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the GetExtents function.
*
* This script calls GetExtents to obtain the current video
* resolution of the target system. It then displays that
* information in a dialog.
*
\*-----*/

#define TITLE_TEXT "GetExtents example"
#define MSG_TEXT  "Video Resolution: %d by %d"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_GetExtents(HWND);

function ExFn_GetExtents(hMSI)
    NUMBER  nvDx, nvDy;
begin
    // Get the video resolution of the target system.
    if (GetExtents (nvDx, nvDy) < 0) then
        // Report the error.
```

```

        MessageBox ("GetExtents failed.", SEVERE);
    else
        // Display the information
        sprintfBox (INFORMATION, TITLE_TEXT, MSG_TEXT, nvDx, nvDy);
    endif;

end;

```

## GetFileInfo


Call the **GetFileInfo** function to determine a file's or directory's attributes, modify date, time, MD5 signature, or size. In each GetFileInfo statement, you can request only one of the data. For example, to obtain the date and time information for a file or directory, you must call GetFileInfo twice—once to obtain the date and once to obtain the time.

### Syntax

```
GetFileInfo ( szPathName, nType, nvResult, svResult );
```


## Parameters

**Table 159 • GetFileInfo Parameters**

Parameter	Description
<b>szPathName</b>	Specifies the fully qualified name of the file or directory about which you want to retrieve information. You can specify a valid Uniform Resource Locator (URL) in this parameter unless you pass FILE_ATTRIBUTE or FILE_SHARED_COUNT as the nType parameter, in which case the function fails and returns ISERR_INVALID_ARG. To check the validity of a URL, call Is(VALID_PATH, szURL).
<b>nType</b>	<p>Specifies the type of file or directory information to retrieve. If the information is a number, GetFileInfo places it in nvResult. If the information is a string, GetFileInfo places it in svResult. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>FILE_ATTRIBUTE</b>—The attribute of the file or directory is returned in nvResult.</li> <li>• <b>FILE_DATE</b>—The modify date of the file or directory in the format YYYY\MM\DD is returned in svResult.</li> <li>• <b>FILE_MD5_SIGNATURE</b>—Returns the MD5 signature of the file specified in svResult. Generating and returning an MD5 signature is an expensive operation that requires reading the contents of the entire file. Only use this parameter when absolutely necessary. FILE_MD5_SIGNATURE is not supported for URLs.</li> </ul> <p> <b>Note</b> • Raw MD5 signature data for a particular file consists of 16 generated numeric values of 16 bit each (between 0x00 abd 0xFF). These values are usually stored in a string of unsigned characters. However, the InstallScript language does not support unsigned characters, so instead of returning the raw MD5 file data, each of the 16 numeric values are converted to their string equivalent and placed in the resulting string. This results in a string of 32 characters, where each set of two characters represent a single numeric value. This is sometimes referred to as a MD5 hex string.</p> <ul style="list-style-type: none"> <li>• <b>FILE_SHARED_COUNT</b>—The file's reference count.</li> <li>• <b>FILE_SIZE</b>—The size in bytes is returned in nvResult (same as FILE_SIZE_LOW).</li> <li>• <b>FILE_SIZE_LOW</b>—The lower 31 bits of the file's size (in bytes) is returned in nvResult. If the size of the file is more than 2GB, the size of the file is equal to the value returned for FILE_SIZE_HIGH multiplied by 2GB added to the value returned for FILE_SIZE_LOW.</li> <li>• <b>FILE_SIZE_HIGH</b>—The upper 31 bits of the file's size (in bytes) is returned in nvResult. This value is 0 (zero) if the file is less than 2GB. If not, the size of the file is equal to the value returned for FILE_SIZE_HIGH multiplied by 2GB added to the value returned for FILE_SIZE_LOW.</li> <li>• <b>FILE_TIME</b>—The modify time of the file or directory in the format HH:MM:SS is returned in svResult.</li> </ul>



**Table 159** • GetFileInfo Parameters (cont.)

Parameter	Description
	 <p><b>Note</b> • After calling <code>GetFileInfo</code> with <code>FILE_ATTRIBUTE</code> as the second parameter (<code>nType</code>), use if-then-else logic to determine the file's or directory's attributes. If <code>nvResult = FILE_ATTR_NORMAL</code>, then no attributes are set. If <code>nvResult != FILE_ATTR_NORMAL</code>, test the result of bitwise AND (&amp;) operations on <code>nvResult</code> and one or more of the following file attribute constants to determine which attributes are set:</p> <ul style="list-style-type: none"> <li>• <code>FILE_ATTR_NORMAL</code>: The file is a normal file.</li> <li>• <code>FILE_ATTR_ARCHIVED</code>: The file is archived.</li> <li>• <code>FILE_ATTR_DIRECTORY</code>: The file is a directory.</li> <li>• <code>FILE_ATTR_HIDDEN</code>: The file is hidden.</li> <li>• <code>FILE_ATTR_READONLY</code>: The file is read-only.</li> <li>• <code>FILE_ATTR_SYSTEM</code>: The file is a system file.</li> </ul> <pre> if (nvResult = FILE_ATTR_NORMAL) then     //The file is NORMAL. else     if (FILE_ATTR_HIDDEN &amp; nvResult) then         //The file is HIDDEN.     endif;      if (FILE_ATTR_READONLY &amp; nvResult) then         //The file is READ-ONLY.     endif; endif; </pre>
<b>nvResult</b>	Returns numeric information.
<b>svResult</b>	Returns string information.

## Return Values

**Table 160** • GetFileInfo Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully retrieved the requested file information.
<b>ISERR_INVALID_ARG (-2)</b>	Indicates that an invalid argument was passed to the function.
<b>All other negative values</b>	Indicates that the function was unable to retrieve the requested information.

## GetFileInfo Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the GetFileInfo function.
*
* GetFileInfo is called to retrieve the time, date, and
* attributes of a file.
*
* Note: In order for this script to run correctly, you
*       must set the constant EXAMPLE_TXT to the name
*       of an existing file on the target system.
*
\*-----*/

#define EXAMPLE_FILE "C:\\IO.sys"
#define TITLE_TEXT  "GetFileInfo Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_GetFileInfo(HWND);

function ExFn_GetFileInfo(hMSI)
    STRING svResult, szAttributes;
    NUMBER nvResult;
    LIST listID;
begin

    // Create a list to store information about the file.
    listID = ListCreate (STRINGLIST);

    // Get the date the file was created or last updated into svResult.
    if (GetFileInfo (EXAMPLE_FILE, FILE_DATE, nvResult, svResult) = 0) then
        // Add the file date to the list.
```

```

        ListAddString (listID, "File date: " + svResult, AFTER);
    endif;

    // Get the time the file was created or last updated into svResult.
    if (GetFileInfo (EXAMPLE_FILE, FILE_TIME, nvResult, svResult) = 0) then
        // Add the file time to the list.
        ListAddString (listID, "File time: " + svResult, AFTER);
    endif;

    // Get the file attributes into nvResult.
    if (GetFileInfo (EXAMPLE_FILE, FILE_ATTRIBUTE, nvResult, svResult) = 0) then
        // Test for no attribute.
        if (nvResult = FILE_ATTR_NORMAL) then
            // No attributes are set. Add that info to the list
            ListAddString(listID, "File attributes: Normal", AFTER);
        else
            // Append attributes to this string.
            szAttributes = "File attributes: ";

            // Is it archived?
            if (FILE_ATTR_ARCHIVED & nvResult) then
                szAttributes = szAttributes + "archived ";
            endif;

            // Is it hidden?
            if (FILE_ATTR_HIDDEN & nvResult) then
                szAttributes = szAttributes + "hidden ";
            endif;

            // Is it read-only?
            if (FILE_ATTR_READONLY & nvResult) then
                szAttributes = szAttributes + "read-only ";
            endif;

            // Is it a system file?
            if (FILE_ATTR_SYSTEM & nvResult) then
                szAttributes = szAttributes + "system ";
            endif;

            // Is it a directory?
            if (FILE_ATTR_DIRECTORY & nvResult) then
                szAttributes = szAttributes + "directory ";
            endif;

        endif;

        // Add the file attributes to the list.
        ListAddString (listID, szAttributes, AFTER);

    endif;

    // Display the list.
    SdShowInfoList (TITLE_TEXT, EXAMPLE_FILE, listID);

    // Remove the list from memory.
    ListDestroy (listID);

end;

```

# GetFolderNameList

The **GetFolderNameList** function is used to enumerate all program item shortcuts and subfolders in a specified folder. This function can also be used to enumerate program item shortcuts and subfolders in the root folder.

## Syntax

```
GetFolderNameList ( szFolderName, listItemsID, listSubFoldersID );
```

## Parameters

**Table 161** • GetFolderNameList Parameters

Parameter	Description
<b>szFolderName</b>	<p>Specifies the name of the folder to be queried. You can specify a fully qualified path for szFolderName, such as:</p> <p><b>"C:\\Windows\\Start Menu\\Programs\\Accessories\\Games"</b></p> <p>If szFolderName is null, <b>GetFolderNameList</b> searches the default Programs directory. If you do not specify an absolute path (a path that includes a drive specification, for example, "C:\\Program Files\\AppName") for szFolderName, <b>GetFolderNameList</b> searches for a subfolder under the default Programs directory; the location depends on the value of the InstallScript variable ALLUSERS, as well as the version of Windows on the target system.</p> <p>You can also use an InstallScript system variable:</p> <ul style="list-style-type: none"> <li>● <b>FOLDER_DESKTOP</b>—Searches the Desktop folder.</li> <li>● <b>FOLDER_STARTUP</b>—Searches the Startup menu.</li> <li>● <b>FOLDER_STARTMENU</b>—Searches the Start menu.</li> <li>● <b>FOLDER_PROGRAMS</b>—Searches the Start Menu\\Programs folder.</li> </ul> <p>Or you could use a relative path, such as:</p> <p><b>FOLDER_PROGRAMS ^ "ACCESSORIES\\GAMES"</b></p>
<b>listItemsID</b>	<p>Returns a list with the names of the program item shortcuts in szFolderName. Note that if szFolderName includes both personal and common program items, the list returned in this parameter will contain either the common or personal program item shortcuts, but not both. The list identified by listItemsID must already have been initialized by a call to <b>ListCreate</b>.</p>
<b>listSubFoldersID</b>	<p>Returns a list with the names of the subfolders in szFolderName. Note that if szFolderName includes both personal and common folders, the list returned in this parameter will contain either the common or personal folders, but not both. The list identified by listSubFoldersID must already have been initialized by a call to <b>ListCreate</b>.</p>

## Return Values

**Table 162** • GetFolderNameList Return Values

Return Value	Description
<b>0</b>	<b>GetFolderNameList</b> successfully retrieved all the programs items and subfolder names.

**Table 162** • GetFolderNameList Return Values (cont.)

Return Value	Description
< 0	<p><b>GetFolderNameList</b> was unable to retrieve the program items and subfolder names.</p> <p>You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a>.</p>

## GetFolderNameList Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the GetFolderNameList function.
 *
 * GetFolderNameList is called to retrieve all of the folders
 * and program items on the desktop.
 *
 \*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_GetFolderNameList(HWND);

function ExFn_GetFolderNameList(hMSI)
    NUMBER    nResult;
    LIST      listItemsID, listFoldersID;
begin

    // Create lists for folders and program names.
    listItemsID = ListCreate (STRINGLIST);
    listFoldersID = ListCreate (STRINGLIST);

    if (listItemsID = LIST_NULL) || (listFoldersID = LIST_NULL) then
        MessageBox ("Unable to create lists.", SEVERE);
    else
        // Display a message box while building the list.
        SdShowMsg ("Searching . . . please wait.", TRUE);

        // Place the folder and program names into the lists.
        nResult = GetFolderNameList (FOLDER_DESKTOP, listItemsID, listFoldersID);

        // Close the message box.
        SdShowMsg ("", FALSE);

        if (nResult < 0) then
            MessageBox ("Unable to retrieve desktop folder and program names.",
                SEVERE);

```

```

else

    // Display the lists.
    repeat

        // Disable the Back button.
        Disable (BACKBUTTON);

        // Display the list of items.
        nResult = SdShowInfoList ("", "Items list:", listItemsID);

        // Enable the Back button.
        Enable (BACKBUTTON);

        // Display the list of folders.
        nResult = SdShowInfoList ("", "Folders list:", listFoldersID);

    until (nResult = NEXT);

    endif;
endif;

end;

```

## GetFont

The **GetFont** function builds a font and retrieves its handle. You can use the font handle to specify the font used by the controls in a custom dialog.




---

**Note** • *InstallShield deletes all fonts created with this function when the installation terminates. In addition, InstallShield releases all system resources upon termination.*

### Syntax

```
GetFont ( szFontName, nPointSize, nAttributes );
```

## Parameters

**Table 163** • GetFont Parameters

Parameter	Description
<b>szFontName</b>	Specifies the name of the font that you want to build.
<b>nPointSize</b>	Specifies the point size of the font that you want to build.
<b>nAttributes</b>	<p>Specifies the style of the font. Pass one or more of the following predefined constants in this parameter. Specify multiple styles by combining constants with the bitwise OR operator (   ).</p> <ul style="list-style-type: none"> <li>• <b>STYLE_BOLD</b>—Specifies a bold style font.</li> <li>• <b>STYLE_ITALIC</b>—Specifies that you want the font to be italicized.</li> <li>• <b>STYLE_NORMAL</b>—Specifies a normal style font.</li> <li>• <b>STYLE_UNDERLINE</b>—Specifies that you want the characters to be underlined.</li> </ul>

## Return Values

**Table 164** • GetFont Return Values

Return Value	Description
<b>XXXX</b>	The handle to the font.

If GetFont cannot find the font that is named in szFontName, the function builds the Arial font and returns a handle to that font.

## GetFont Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the GetFont and CtrlSetFont functions.
*
* This example script calls GetFont to retrieve the handles
* of four fonts. These handles are then passed to CtrlSetFont
* to set the font of the static text fields in a custom dialog
* box.
*
* The "custom" dialog used in this script is actually the
* InstallShield dialog that is displayed by the built-in
```



```

* function SetupType. Because this dialog is stored in
* the file _isres.dll, which is already compressed in
* the installation, it can be used in a script as a custom
* dialog.
*
\*-----*/

// Dialog and control IDs.
#define RES_DIALOG_ID      10203 // ID of the custom dialog
#define RES_PBUT_NEXT      1     // ID of Next button
#define RES_PBUT_CANCEL    9     // ID of Cancel button
#define RES_TEXT_1         202   // ID of first static text box

#define RES_TEXT_2         210   // ID of second static text box
#define RES_TEXT_3         220   // ID of third static text box
#define RES_TEXT_4         230   // ID of fourth static text box

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_GetFont(HWND);

function ExFn_GetFont(hMSI)
    STRING szDialogName;
    NUMBER nResult, nCmdValue;
    HWND   hFont1, hFont2, hFont3, hFont4, hwndDlg;
    BOOL    bDone;
begin

    // Get the handle of the fonts to use for the static text
    // that is displayed by the custom dialog.
    hFont1 = GetFont("Arial", 14, STYLE_BOLD);
    hFont2 = GetFont("Times New Roman", 11, STYLE_ITALIC);
    hFont3 = GetFont("Arial", 10, STYLE_BOLD);
    hFont4 = GetFont("Courier New", 9, STYLE_NORMAL);

    if (hFont1 = 0 || hFont2 = 0 || hFont3 = 0 || hFont4 = 0) then
        // Report an error; then terminate.
        MessageBox ("Unable to get all fonts. ", SEVERE);
        abort;
    endif;

    // Specify a name to identify the custom dialog in this installation.
    szDialogName = "CustomDialog";

    // Define the dialog. Pass a null string in the second parameter
    // to get the dialog from _isuser.dll or _isres.dll. Pass a null
    // string in the third parameter because the dialog is identified
    // by its ID in the fourth parameter.

    nResult = EzDefineDialog (szDialogName, "", "", RES_DIALOG_ID);

    if (nResult < 0) then
        // Report an error; then terminate.
        MessageBox ("Error in defining dialog.", SEVERE);
        abort;
    endif;

    // Initialize indicator used to control the while loop.

```

```

bDone = FALSE;

// Loop until done.
repeat

    // Display the dialog and return the next dialog event.
    nCmdValue = WaitOnDialog (szDialogName);

    // Respond to the event.
    switch (nCmdValue)
    case DLG_CLOSE:
        // The user clicked the window's Close button.
        Do (EXIT);
    case DLG_ERR:
        MessageBox ("Unable to display dialog. Setup canceled.", SEVERE);
        abort;
    case DLG_INIT:
        // Initialize the back, next, and cancel button enable/disable states
        // for this dialog and replace %P, %VS, %VI with
        // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
        // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
        hwndDlg = CmdGetHwndDlg(szDialogName);
        SdGeneralInit(szDialogName, hwndDlg, 0, "");

        // Set the font and text for static text box 1.
        if (CtrlSetFont (szDialogName, hFont1, RES_TEXT_1) = 0) then
            CtrlSetText (szDialogName, RES_TEXT_1,
                "This text is set in 14-point Arial bold.");
        else
            CtrlSetText (szDialogName, RES_TEXT_1,
                "Unable to set font for first static text box.");
        endif;

        // Set font and text for static text box 2.
        if (CtrlSetFont (szDialogName, hFont2, RES_TEXT_2) = 0) then
            CtrlSetText (szDialogName, RES_TEXT_2,
                "This text is set in 11-point Times New Roman italic.");
        else
            CtrlSetText (szDialogName, RES_TEXT_2,
                "Unable to set font for second static text box.");
        endif;

        // Set font and text for static text box 3.
        if (CtrlSetFont (szDialogName, hFont3, RES_TEXT_3) = 0) then
            CtrlSetText (szDialogName, RES_TEXT_3,
                "This text is set in 10-point Arial bold.");
        else
            CtrlSetText (szDialogName, RES_TEXT_3,
                "Unable to set font for third static text box.");
        endif;

        // Set font and text for static text box 4.
        if (CtrlSetFont (szDialogName, hFont4, RES_TEXT_4) = 0) then
            CtrlSetText (szDialogName, RES_TEXT_4,
                "This text is set in 9-point Courier New.");
        else
            CtrlSetText (szDialogName, RES_TEXT_4,
                "Unable to set font for fourth static text box.");
        endif;

```

```

        case RES_PBUT_NEXT:
            bDone = TRUE;
        case RES_PBUT_CANCEL:
            // The user clicked the Cancel button.
            Do (EXIT);
        endswitch;

    until bDone;

    // Close the dialog.
    EndDialog (szDialogName);

    // Free the dialog from memory.
    ReleaseDialog (szDialogName);

end;

```

## GetLine

The **GetLine** function reads a line of text from a text file opened in read-only mode. Before you call **GetLine**, you must first call **OpenFileMode** to set the file mode to read-only and then call **OpenFile** to open the file (which can be a file on the Internet). The first call to **GetLine** reads the first line of text from the file. After reading a line, **GetLine** repositions the file pointer to the next line. The second call to **GetLine** reads the second line, and so forth. **GetLine** strips the carriage return and line feed characters from the end of the line it returns.

When **GetLine** has read all the lines in a file, it returns an end-of-file error. If you open a file in append mode, the **GetLine** function fails if you call it because the file pointer is at the end of the file. The function also fails if the file specified by **nvFileHandle** was opened in a binary mode.

The maximum size of a line is 4,096 characters. To read multiple lines from a file, use a separate **GetLine** call for each line or place the **GetLine** statement in a loop.




---

**Tip** • To write to a text file, use the [WriteLine](#) function. *WriteLine* always produces lines that have a carriage return and line feed character combination at the end of the line.

### Syntax

```
GetLine ( nvFileHandle, svLine );
```

## Parameters

**Table 165** • GetLine Parameters

Parameter	Description
<b>nvFileHandle</b>	Specifies the handle of a file that has been opened by a call to OpenFile.
<b>svLine</b>	Returns a line of text from the file specified by nvFileHandle.

## Return Values

**Table 166** • GetLine Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully retrieved a line of text from an open text file.
<b>&lt; 0</b>	Indicates that the function failed due to an end-of-file error or another error condition. This condition also indicates GetLine has read all the lines in the file.

## GetLine Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the GetLine function.
*
* GetLine is called in this script to read a text file line by
* line.
*
* Note: In order for this script to run properly, you must set
*       the preprocessor constants so that they reference an
*       existing text file on the target system.
*
\*-----*/

#define EXAMPLE_FILE "Readme.txt"
#define EXAMPLE_DIR  "C:\\Windows"
#define TITLE_TEXT   "GetLine example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_GetLine(HWND);

function ExFn_GetLine(hMSI)
```

```

    STRING  szFileName, szPath, szText, svLine;
    NUMBER  nFlag, nFileHandle;
    LIST    listID;
begin

    // Create a list to store lines from the file.
    listID = ListCreate (STRINGLIST);

    // Set the file mode to normal.
    OpenFileMode (FILE_MODE_NORMAL);

    // Open the file for editing.
    OpenFile (nFileHandle, EXAMPLE_DIR, EXAMPLE_FILE);

    // Get lines from the file into the list.
    while (GetLine (nFileHandle, svLine) = 0)
        ListAddString (listID, svLine, AFTER);
    endwhile;

    // Close the file.
    CloseFile (nFileHandle);

    // Display the list.
    SdShowInfoList (TITLE_TEXT, EXAMPLE_FILE, listID);

    // Remove the list from memory.
    ListDestroy (listID);

end;

```

## GetMemFree

The **GetMemFree** function is obsolete and should not be used. This function always returns 1048576.

### Syntax

```
GetMemFree ( );
```




---

**Tip** • To determine the amount of actual physical memory available on the target system, call [GetSystemInfo](#).

## GetObject

The **GetObject** function initializes the szObjectName object and returns a reference that can be assigned to a variable of type OBJECT by using the set keyword. To get a reference to a new or existing COM object (as Visual Basic's GetObject function does), call CoGetObject.




---


**Tip** • To check whether the object was initialized successfully, call the *IsObject* function.

## Syntax

```
GetObject ( szObjectName );
```

## Parameters

**Table 167** • GetObject Parameters

Parameter	Description
<b>szObjectName</b>	<p>Specifies the name of the InstallShield object to be initialized as that name is displayed in the Features view, for example:</p> <pre>"New ATL 3.0"</pre>  <p><b>Note</b> • To ensure that the correct InstallShield object is initialized, give each InstallShield object in your project a unique name.</p> <p>If szObjectName is set to a null string (""), in an object project GetObject returns a reference to the object that is calling it (in a setup project GetObject returns an invalid reference).</p> <p>Setting szObjectName to a null string is useful for reading and writing object properties within the object itself. For example, the following code displays the current status description for the object:</p> <pre>OBJECT oThis;  set oThis = GetObject(""); if( IsObject( oThis ) ) then     MessageBox( oThis.Status.Description, INFORMATION ); endif;</pre>

## Return Values

A reference that can be assigned to a variable of type OBJECT by using the set keyword.

# GetObjectByIndex



**Project** • This information applies to InstallScript projects.

The **GetObjectByIndex** function finds the setup's or object's subobject that is specified by nIndex and returns a reference that can be assigned to a variable of type OBJECT by using the set keyword.

## Syntax

```
GetObjectByIndex ( nIndex );
```

## Parameters

**Table 168** • GetObjectByIndex Parameters

Parameter	Description
<b>nIndex</b>	Specifies the index of a subobject. Index numbers start at one (1); if an object passes zero (0) as the argument to GetObjectByIndex, the function returns a reference to the object or setup that is calling the function rather than any subobject. (This is the same as calling GetObject("").) If a setup calls GetObjectByIndex(0), the reference that is returned does not contain any meaningful information.

## Return Values

A reference that can be assigned to a variable of type OBJECT by using the set keyword.

## Additional Information

To check whether the object was initialized successfully, call the IsObject function.

To get the number of subobjects contained by the object or setup, call [GetObjectCount](#).

# GetObjectCount



**Project** • This information applies to InstallScript projects.

The **GetObjectCount** function returns the number of subobjects contained by the object or setup.

## Syntax

```
GetObjectCount ( );
```

## Parameters

None

## Return Values

**Table 169** • GetObjectCount Return Values

Return Value	Description
<b>&gt;= 0</b>	The number of subobjects.
<b>&lt; ISERR_SUCCESS</b>	GetObjectCount cannot determine the number of subobjects.

## Additional Information

To get a reference to a subobject that is specified by its index within the setup or object, call [GetObjectByIndex](#).

# GetProfInt

The **GetProfInt** function retrieves an integer from an .ini file. GetProfInt works like the Windows API GetPrivateProfileInt with the nDefault parameter specified as 0.

## Syntax

```
GetProfInt ( szFileName, szSectionName, szKeyName, nvValue );
```

## Parameters

**Table 170** • GetProfInt Parameters

Parameter	Description
<b>szFileName</b>	Specifies the name of the .ini file from which to get the current integer value of a key. If szFileName is unqualified (that is, if a drive designation and path are <i>not</i> included), InstallShield searches for the file in the Windows folder.
<b>szSectionName</b>	Specifies the name of the .ini file section to search for szKeyName. The section name should not be enclosed within delimiting brackets ( [ ] ). The search for this name is not case-sensitive.
<b>szKeyName</b>	Specifies the key whose integer value is to be returned in nvValue. The search for this key is not case-sensitive.
<b>nvValue</b>	Returns an integer value currently assigned to szKeyName. Due to limitations of the GetPrivateProfileInt function, this function can return only a 16-bit value from the profile. Therefore, the maximum value that can be returned is 65,535; larger values may not be returned correctly. If you need to return a larger value, use the general file handling functions, such as FileGrep and FileInsertLine, and then convert the returned string into an integer by calling StrToNum.

## Return Values

GetProfInt always returns 0.

## Additional Information

- As with the Windows API function **GetPrivateProfileInt**, no error is returned if an error occurs because the file, section, or key name cannot be found. Instead, nvValue will contain 0. For that reason, it is not possible to distinguish between an error and a returned value of zero. To distinguish between zero and an error, call **GetPrivateProfileInt** directly and specify an alternate default value.
- Some calls to **GetPrivateProfileInt**—and therefore to **GetProfInt**—are mapped automatically to the Windows registry instead of the profile.



## GetProfInt Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the GetProfInt function.
 *
 * GetProfInt is called to retrieve the value of a key in
 * the file specified by EXAMPLE_INI.
 *
 * Note: Before running this script, create a file called
 *       ISExample.ini in the root of drive C. Include the
 *       following lines in that file.
 *
 *       [ISExample]
 *       ISKey=100
 *
 \*-----*/

#define EXAMPLE_INI "C:\\ISExample.ini"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_GetProfInt(HWND);

function ExFn_GetProfInt(hMSI)
    STRING szSectionName, szKeyName;
    NUMBER nvValue;
begin

    szSectionName = "ISExample";
    szKeyName     = "ISKey";

    // Get the value of szKeyName under the szSectionName section.
    GetProfInt (EXAMPLE_INI, szSectionName, szKeyName, nvValue);

    sprintfBox (INFORMATION, "GetProfInt Example",
               "The value of %s is: %d.", szKeyName, nvValue);

end;
```

## GetProfSectionKeyCount



**Project** • This information applies to InstallScript projects.

The **GetProSectionKeyCount** function returns the number of keys in the section specified by szSectionName in the initialization file specified by szFilename.

## Syntax

```
GetProfSectionKeyCount ( szFilename, szSectionName );
```

## Parameters

**Table 171** • GetProfSectionKeyCount Parameters

Parameter	Description
<b>szFilename</b>	Specifies the fully qualified name of the .ini file in which to count the keys.
<b>szSectionName</b>	Specifies the name of the .ini file section in which to count the keys. The section name should not be enclosed within delimiting brackets ( [ ] ). The search for this name is not case-sensitive.

## Return Values

**Table 172** • GetProfSectionKeyCount Return Values

Return Value	Description
<b>X</b>	The number of keys in the specified section of the specified file.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function could not determine the number of keys.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

# GetProfString

The **GetProfString** function retrieves a profile string from the specified .ini file. GetProfString works like the Windows API GetPrivateProfileString.





**Note** • *GetProfString uses the functions provided by your operating environment's API to access the .ini file. Because of this, InstallShield's functionality might be limited by the operating environment.*

## Syntax

```
GetProfString ( szFileName, szSectionName, szKeyName, svResult );
```

## Parameters

**Table 173** • GetProfString Parameters

Parameter	Description
<b>szFileName</b>	Specifies the name of the .ini file from which to get the current value of a key. If szFileName is unqualified (that is, if a drive designation and path are <i>not</i> included), InstallShield searches for the file in the Windows folder.
<b>szSectionName</b>	<p>Specifies the name of the .ini file section to search for szKeyName. The section name should not be enclosed within delimiting brackets ( [ ] ). The search for this name is not case-sensitive. To obtain a list of all section names in the initialization file, pass a null string ("" ) in this parameter.</p> <p> <b>Tip</b> • To obtain a list of all section names in the initialization file, pass a null string ("" ) in szSectionName. The section names, delimited by null characters, are returned in the parameter svResult. svResult must be long enough to accept all of the section names. Use the StrGetTokens function to extract the individual section names from this string.</p>
<b>szKeyName</b>	<p>Specifies the key whose value is to be returned in svResult. The search for this key is not case-sensitive. To obtain a list of all key names in the section, pass a null string ("" ) in this parameter.</p> <p> <b>Tip</b> • To obtain a list of all key names in the section specified by szSectionName, pass a null string ("" ) in szKeyName. The key names, delimited by null characters, are returned in the parameter svResult. svResult must be long enough to accept all of the key names. Use the StrGetTokens function to extract the individual key names from this string.</p>
<b>svResult</b>	If szSectionName specifies a section name and szKeyName specifies a key name, the value of that key is returned in this parameter. If szSectionName specifies a null string ("" ), all of the section names are returned in svResult. If szKeyName specifies a null string ("" ), all of the key names in the section specified szSectionName are returned in svResult.

## Return Values

**Table 174** • GetProfString Return Values

Return Value	Description
<b>0</b>	GetProfString successfully returned the value of the profile string.
<b>&lt; 0</b>	GetProfString was unable to return the value.

**Table 174** • GetProfString Return Values (cont.)

Return Value	Description
-2	The length of the key's value exceeded 2048 characters, which is the maximum number of characters that can be returned in svResult by GetProfString.

## GetProfString Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the functions AddProfString and GetProfString.
*
* This script adds a profile string to a file; then it
* retrieves and displays the string that was added.
*
* Note: The first time you run this script, it will create a
*       file named ISExempl.ini in the root of drive C. You
*       may delete that file when you have finished analyzing
*       this script.
*
\*-----*/

#define EXAMPLE_INI "C:\\ISExempl.ini"

// The new section, key, and value to add to the file.
#define NEW_SECTION "New Section"
#define NEW_KEY     "New Key"
#define NEW_VALUE   "Test"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_GetProfString(HWND);

function ExFn_GetProfString(hMSI)
    STRING svResult;
begin

    // Add the profile string to the file.
    if (AddProfString (EXAMPLE_INI, NEW_SECTION, NEW_KEY, NEW_VALUE) != 0) then
        // Display an error message if the string could not be added.
        MessageBox ("AddProfString failed.", SEVERE);
    else
        // Retrieve the value of a key from the file.
        if (GetProfString (EXAMPLE_INI, NEW_SECTION, NEW_KEY, svResult) != 0) then
            // Display an error message if the string could not be retrieved.
            MessageBox ("GetProfString failed.", SEVERE);
        else

```

```
        // Display the key and its current value.  
        MessageBox (NEW_KEY + "=" + svResult, INFORMATION);  
    endif;  
endif;  
  
end;
```

## GetProfStringList

The **GetProfStringList** function retrieves lists of key names and string values from the specified section of the specified initialization file.

### Syntax

```
GetProfStringList ( szFileName, szSectionName, listKeyNames, listValues );
```

## Parameters

**Table 175** • GetProfStringList Parameters

Parameter	Description
<b>szFileName</b>	Specifies the name of the .ini file from which to get the key names and string values. If szFileName is unqualified (that is, if a drive designation and path are not included), InstallShield searches for the file in the Windows folder.
<b>szSectionName</b>	Specifies the name of the .ini file section to search for the key names and string values. The section name should not be enclosed within delimiting brackets ( [ ] ). The search for this name is not case-sensitive.
<b>listKeyNames</b>	Returns a list of key names. The string list identified by listKeyNames must already have been initialized by a call to ListCreate.
<b>listValues</b>	Returns a list of string values. The string list identified by listValues must already have been initialized by a call to ListCreate.

## Return Values

**Table 176** • GetProfStringList Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS (0)</b>	Indicates that the function successfully read the section and inserted key names and string values into the specified lists.
<b>&lt; ISERR_SUCCESS (0)</b>	Indicates that the function could not read the section or insert key names and string values into the specified lists.

## Additional Information

**GetProfStringList** calls the Windows API **GetPrivateProfileSection** and specifies a 32 KB buffer for the data. Therefore, the function returns only the first 32 KB of data from the section. An installation that needs to handle sections with more than 32 KB of data should call the Windows API **GetPrivateProfileSection** directly and specify a larger buffer size (and parse the returned information manually).

## GetProfStringList Example

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the function GetProfStringList.
*
* This script retrieves keys and their values from an
* initialization file.
*-----*/
```

```

#define EXAMPLE_INI "C:\\ISExempl.ini"

// The new section, key, and value to add to the file.
#define SECTION "InstallShield"
#define KEY1     "Key1"
#define KEY2     "Key2"
#define KEY3     "Key3"
#define KEY4     "Key4"
#define KEY5     "Key5"
#define VALUE1   1
#define VALUE2   2
#define VALUE3   3
#define VALUE4   4
#define VALUE5   5

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_GetProfString(HWND);

function ExFn_GetProfString(hMSI)
    STRING svResult, svKeyName, svKeyVal;
    LIST listKeyNames, listKeyValues;
    NUMBER nVal;
begin

    // Add the profile strings to the file.
    WriteProfInt (EXAMPLE_INI, SECTION, KEY1, VALUE1);
    WriteProfInt (EXAMPLE_INI, SECTION, KEY2, VALUE2);
    WriteProfInt (EXAMPLE_INI, SECTION, KEY3, VALUE3);
    WriteProfInt (EXAMPLE_INI, SECTION, KEY4, VALUE4);
    WriteProfInt (EXAMPLE_INI, SECTION, KEY5, VALUE5);

    // Create a list to hold the key names;
    listKeyNames = ListCreate(STRINGLIST);

    // If an error occurred, report it; then terminate.
    if (listKeyNames = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
        //Add your custom error handling code here.
        abort;
    endif;

    // Create a list to hold the key values;
    listKeyValues = ListCreate(STRINGLIST);

    // If an error occurred, report it; then terminate.
    if (listKeyValues = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
        //Add your custom error handling code here.
        abort;
    endif;

    // Retrieve the keys from the specified section of the file.
    nVal = GetProfStringList (EXAMPLE_INI, SECTION,
                             listKeyNames, listKeyValues);

    if (nVal = 0) then
        nVal = ListGetFirstString (listKeyNames, svKeyName);

```

```

if (nVal = END_OF_LIST) then
    MessageBox("No keys found in [" + SECTION + "]", WARNING);
else
    ListGetFirstString (listKeyValues, svKeyVal);

    repeat
        // Display the keys and their values.
        MessageBox(svKeyName + "=" + svKeyVal, INFORMATION);

        nVal = ListGetNextString (listKeyNames, svKeyName);
        if !(nVal = END_OF_LIST) then
            ListGetNextString (listKeyValues, svKeyVal);
        endif;

    until nVal = END_OF_LIST;

endif;
endif;

end;

```

## GetShortcutInfo

The **GetShortcutInfo** function checks for the existence of a specific shortcut or subfolder name. If the InstallScript engine finds the shortcut or subfolder, **GetShortcutInfo** returns its attributes. The attributes include the product's command line, working directory, icon path, shortcut key, and minimize flag.

To use **GetShortcutInfo**, enter information in the parameters szShortcutFolder and szName. The InstallScript engine fills the remaining parameters with the shortcut's or subfolder's attributes.

### Syntax

```

GetShortcutInfo (szShortcutFolder, szName, svCmdLine, svWrkDir, svIconPath, nvIconIndex, svShortCutKey,
    nvMinimizeFlag);

```



## Parameters

**Table 177** • GetShortcutInfo Parameters

Parameter	Description
<b>szShortcutFolder</b>	<p>Specify the name of the folder that contains the shortcut or subfolder. You can specify a fully qualified path for szShortcutFolder, such as:</p> <pre>"C:\\ProgramData\\Microsoft\\Windows\\Start Menu\\Programs\\Games"</pre> <p>If szShortcutFolder is null, <b>GetShortcutInfo</b> searches the default Programs directory. If you do not specify an absolute path (a path that includes a drive specification, for example, "C:\\Program Files\\AppName") for szShortcutFolder, <b>GetShortcutInfo</b> searches for a subfolder under the default Programs directory; the location depends on the value of the InstallScript variable ALLUSERS, as well as the version of Windows on the target system.</p> <p>You can also use an InstallScript system variable:</p> <ul style="list-style-type: none"> <li>● <b>FOLDER_DESKTOP</b>—Queries items in the Desktop folder.</li> <li>● <b>FOLDER_STARTUP</b>—Queries items in the Startup menu.</li> <li>● <b>FOLDER_STARTMENU</b>—Queries items in the Start menu.</li> <li>● <b>FOLDER_PROGRAMS</b>—Queries items in the Start\\Programs menu.</li> </ul> <p>Or you could use a relative path, such as:</p> <pre>FOLDER_PROGRAMS ^ "ACCESSORIES\\GAMES"</pre>
<b>szName</b>	Specify the name of the shortcut or subfolder for which you are looking.
<b>svCmdLine</b>	The function returns either the command line of the item's executable file or the complete path to the subfolder.
<b>svWrkDir</b>	The function returns the full path of the working directory of the program item. (Not applicable if szName is a subfolder.)
<b>svIconPath</b>	The function returns the full path and file name of the .ico file or .exe file. (Not applicable if szName is a subfolder.)
<b>nvIconIndex</b>	The function returns the index of the icon that is used for the shortcut. (Not applicable if szName is a subfolder.)
<b>svShortCutKey</b>	The function returns the item's shortcut key. (Not applicable if szName is a subfolder.)

**Table 177** • GetShortcutInfo Parameters (cont.)

Parameter	Description
<b>nvMinimizeFlag</b>	<p>The function returns one of the following constants, indicating whether an application window is minimized when first displayed:</p> <ul style="list-style-type: none"> <li>• <b>NULL</b>—Indicates that the application's window is not minimized upon startup.</li> <li>• <b>RUN_MINIMIZED</b>—Indicates that the application's window is minimized upon startup.</li> </ul> <p>(Not applicable if szName is a subfolder.)</p>

## Return Values

**Table 178** • GetShortcutInfo Return Values

Return Value	Description
<b>IS_ITEM (0)</b>	Indicates szName is a shortcut in szShortcutFolder.
<b>IS_FOLDER (1)</b>	Indicates szName is a subfolder in szShortcutFolder.
<b>&lt; 0</b>	<p>Indicates that the function was unable to find the shortcut or subfolder name.</p> <p>You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a>.</p>

## Additional Information

The location of the Start menu is different under different languages. The InstallScript engine automatically selects the correct path.

## GetShortcutInfo Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the GetShortcutInfo function.
*
* GetShortcutInfo is called to find the attributes of a target
* file or subfolder.
*
* Note: Before running this script, set the defined constants
*       FOLDER_NAME and SHORTCUT so that they reference an
*       existing folder name and shortcut.
*

```

```

\*-----*/
// Define constants to reference the shortcut or folder name.
#define FOLDER_NAME "C:\\Windows\\Start Menu\\Programs"
#define SHORTCUT "InstallShield"

function OnFirstUIAfter()
    STRING svCmdLine, svWrkDir, svIconPath;
    STRING svShortCutKey, svGroupPath, szTitle, szMsg, szInfo, svMinFlag;
    STRING svMinimizeFlag;
    NUMBER nvIconIndex, nvMinimizeFlag, nResult, nvMinFlag;
    LIST listInfo, listID;
begin

    // Search for an item in the FOLDER_NAME folder.
    nResult = GetShortcutInfo (FOLDER_NAME, SHORTCUT, svCmdLine, svWrkDir,
                             svIconPath, nvIconIndex, svShortCutKey,
                             nvMinimizeFlag);

    // Create string list.
    listInfo = ListCreate (STRINGLIST);

    // Error check GetShortcutInfo.
    if (nResult < 0) then
        // Report the error; then abort.
        MessageBox ("GetShortcutInfo failed.", SEVERE);
        abort;
    // Check if the item is an application.
    elseif (nResult = IS_ITEM) then
        // Add the command line to the string list.
        Sprintf (szInfo, "The command line of %s: %s", SHORTCUT, svCmdLine);
        ListAddString (listInfo, szInfo, AFTER);

        // Add the working directory to string list.
        Sprintf (szInfo, "The working directory of %s: %s", SHORTCUT, svWrkDir);
        ListAddString (listInfo, szInfo, AFTER);

        // Add the icon path to string list.
        Sprintf (szInfo, "The icon path of %s: %s", SHORTCUT, svIconPath);
        ListAddString (listInfo, szInfo, AFTER);

        // Add icon index to string list.
        Sprintf (szInfo, "The index of the icon: %d", nvIconIndex);
        ListAddString (listInfo, szInfo, AFTER);

        // Add shortcut key to string list.
        Sprintf (szInfo, "The shortcut key of %s: %s", SHORTCUT,
                svShortCutKey);
        ListAddString (listInfo, szInfo, AFTER);

        // Check if the item is a folder.
    elseif (nResult = IS_FOLDER) then
        // Add a message to string list.
        Sprintf (szInfo, "The item is a subfolder. GetShortcutInfo does not " +
                "retrieve very much information about subfolders.");
        ListAddString (listInfo, szInfo, AFTER);
    endif;

    // Display the string list.
    szTitle = "GetShortcutInfo Example";

```

```

    szMsg  = "The following are attributes of the item:";
    SdShowInfoList (szTitle, szMsg, listInfo);

    // Destroy the list.
    ListDestroy (listID);

end;

```

## GetStatus



**Project** • The **GetStatus** function applies to InstallScript Object projects.

The **GetStatus** function retrieves the current status of the object; that is, the current value of Status.Number.



**Tip** • To retrieve additional information or to retrieve the status of an object from the installation that contains the object, use the object's Status object.

### Syntax

```
number GetStatus();
```

### Parameters

**GetStatus** takes no parameters.

### Return Values

**GetStatus** returns the status of the object.

## GetSystemInfo

The **GetSystemInfo** function retrieves information about the target system. **GetSystemInfo** collects the information that it returns by using the Windows API.

### Syntax

```
GetSystemInfo ( nItem, nvResult, svResult );
```

## Parameters

**Table 179** • GetSystemInfo Parameters




Parameter	Description
<b>nlItem</b>	Specifies the type of information to retrieve.
<b>nvResult</b>	Returns system information in the form of numeric data.
<b>svResult</b>	Returns system information in the form of string data.

The following table contains a list of constants that you can pass in the `nlItem` parameter to retrieve system information. When using certain constants (such as `DISK_TOTALSPACE_EX`), you must specify additional information in the parameters `nvResult` and/or `svResult` before calling the function.


**Table 180** • nlItem Options

nlItem Option	nvResult Returns	svResult Returns
<b>BOOTUPDRIVE</b>	The ID of the bootup drive, where 1 = A:, 2 = B:, 3 = C:. It is possible to convert this number to the appropriate drive letter by adding 64 (DECIMAL) to the value and then setting a string variable to this value. Use the following syntax to convert:  <code>svResult[0] = 64 + nvResult;</code>	Returns the drive designation (the drive letter followed by a colon) of the bootup drive.
<b>CDROM</b>	TRUE or FALSE Indicates whether a CD-ROM is available.	Not applicable
<b>COLORS</b>	Returns the number of colors available on the user's system. The result is retrieved from the video driver on the target system, rather than from the monitor card. If the card can support 256 colors but the driver can handle only 16 colors, the number of colors returned is 16.	Not applicable


**Table 180 •** nltem Options (cont.)

nltem Option	nvResult Returns	svResult Returns
<b>CPU</b>	 <p><b>Note •</b> This parameter is deprecated. Use the structure members of <a href="#">SYSPROCESSORINFO</a> to determine the processor type.</p> <p>One of the following constants is returned:</p> <ul style="list-style-type: none"> <li>• <b>IS_UNKNOWN</b>—The user's CPU is unknown.</li> <li>• <b>IS_386</b>—The user has a 386 processor.</li> <li>• <b>IS_486</b>—The user has a 486 processor.</li> <li>• <b>IS_PENTIUM</b>—The user has a PENTIUM processor.</li> <li>• <b>IS_ALPHA</b>—The user has an ALPHA processor.</li> </ul>	Not applicable
<b>DATE</b>	Not applicable	The current system date in the format MM-DD-YYYY. Leading zeroes are suppressed in the month and day fields.
<b>DISK_TOTALSPACE</b>	 <p><b>Note •</b> This parameter is obsolete. Use the parameters available with the <a href="#">GetDiskInfo</a> function instead.</p> <p>Returns the total capacity of the disk drive specified in svResult. The maximum value returned is 2 GB. Total disk space greater than that still returns as 2 GB.</p>	The letter of the drive. Note that this parameter is passed to the function; that is, you must assign a value to svResult <i>before</i> calling GetSystemInfo. Also note that you must include the colon (:) after the drive letter; otherwise the function will fail. You can also specify a UNC path in this parameter.
<b>DISK_TOTALSPACE_EX</b>	 <p><b>Note •</b> This parameter is obsolete. Use the parameters available with the <a href="#">GetDiskInfo</a> function instead.</p> <p>Specifies the measurement unit; pass one of the following predefined constants in this parameter: BYTES, KBYTES, MBYTES or GBYTES. Returns the total capacity of the disk drive specified in svResult.</p>	The letter of the drive. Note that this parameter is passed to the function; that is, you must assign a value to svResult <i>before</i> calling GetSystemInfo. Also note that you must include the colon (:) after the drive letter; otherwise the function will fail. You can also specify a UNC path in this parameter.

**Table 180 •** nltem Options (cont.)

nltem Option	nvResult Returns	svResult Returns
<b>DRIVE</b>	 <p><b>Note •</b> <i>This parameter is obsolete. Use the parameters available with the <code>GetDiskInfo</code> function instead.</i></p> <p>Returns the type of the drive specified in svResult. One of the following constants is returned:</p> <ul style="list-style-type: none"> <li>• <b>IS_UNKNOWN</b>—Target drive is unknown.</li> <li>• <b>IS_REMOVABLE</b>—Target drive is a floppy drive.</li> <li>• <b>IS_FIXED</b>—Target drive is a fixed drive.</li> <li>• <b>IS_CDROM</b>—Target drive is a CD-ROM drive.</li> <li>• <b>IS_REMOTE</b>—Target drive is a network drive.</li> </ul>	<p>The letter of the drive followed by a colon (:). Note this parameter is passed to the function; that is, you must assign a value to svResult <i>before</i> calling <code>GetSystemInfo</code>.</p> <p>Due to operating system limitations, UNC paths are not supported for svResult. If you pass a UNC path in svResult, the function returns IS_UNKNOWN.</p>
<b>EXTENDEDMEMORY</b>	<p>Returns the <i>total</i> amount of memory installed on the machine. Due to operating system limitations, the value returned may be slightly different than the actual amount of physical memory installed on the system. This value will normally be within 100K (0.1 MB) of the actual value. Note that the value returned is a measurement in kilobytes.</p> <p><code>GetSystemInfo (EXTENDEDMEMORY, ...)</code> accurately returns the amount of extended memory up to 2 terabytes (TB). If a system has more than 2 TB of extended memory, 2 TB is returned.</p>	Not applicable
<b>LANGUAGE</b>	The InstallScript language constants for the target system are returned in this parameter. For more information, see <i>Creating Multilingual Installations</i> .	The equivalent language name string for the language constant returned in nvResult is returned in this parameter.

**Table 180 • nltem Options (cont.)**

nltem Option	nvResult Returns	svResult Returns
<b>OS</b>	 <p><b>Note •</b> This parameter is deprecated. Use the <code>WIN9X.bWin9X</code> or <code>WINNT.bWinNT</code> members of the <code>SYSINFO</code> structure variable to obtain operating system information.</p> <p>You can conditionally install some or all of your product based upon operating system by using conditional logic for components, features, and custom actions. For more information, see Building Conditional Statements.</p>	Not applicable
<b>OSMAJOR</b>	This parameter is deprecated. Use the <code>nOSMajor</code> member of the <code>SYSINFO</code> structure variable to obtain operating system information.	Not applicable
<b>OSMINOR</b>	This parameter is deprecated. Use the <code>nOSMinor</code> member of the <code>SYSINFO</code> structure variable to obtain operating system information.	Not applicable
<b>PARALLEL</b>	Returns the number of physical parallel ports available.	Not applicable
<b>SERIAL</b>	Returns the number of physical serial ports available.	Not applicable
<b>SYSTEM_DPI</b>	Returns the system DPI value.	Not applicable
<b>SYSTEM_DPI_SCALING</b>	Returns the system DPI scaling value.	Not applicable
<b>TIME</b>	Not applicable	Returns current system time in HH:MM:SS format.



**Table 180 •** nltem Options (cont.)

nltem Option	nvResult Returns	svResult Returns
<b>VIDEO</b>	<p>Returns the type of video adapter installed. (InstallShield cannot detect CGA or monochrome video drivers.) One of the following constants is returned:</p> <ul style="list-style-type: none"> <li>• <b>IS_UNKNOWN</b>—The user's video is unknown.</li> <li>• <b>IS_EGA</b>—EGA resolution.</li> <li>• <b>IS_VGA</b>—VGA resolution.</li> <li>• <b>IS_SVGA</b>—Super VGA (800 x 600) resolution.</li> <li>• <b>IS_XVGA</b>—XVGA (1024 x 768) resolution.</li> <li>• <b>IS_UVGA</b>—Greater than 1024 x 768 resolution.</li> </ul>	Not applicable
<b>VIRTUAL_MACHINE_TYPE</b>	<p>Not applicable</p> <p>This constant returns its value through the function return.</p> <p>For more information, see <a href="#">Detecting Whether the Installation Is Being Run on a Virtual Machine</a>.</p>	Not applicable
<b>VOLUMELABEL</b>	Not applicable	<p>Pass the drive designation—the drive letter followed by a colon—of the disk whose volume label you want to retrieve. The volume label of the specified drive is then returned in this parameter. If the drive has no volume label, a null string ("") is returned.</p>
<b>WINMAJOR</b>	<p>This parameter is deprecated. Use the nWinMajor member of the <a href="#">SYSINFO</a> structure variable to obtain operating system information.</p>	Not applicable
<b>WINMINOR</b>	<p>This parameter is deprecated. Use the nWinMinor member of the <a href="#">SYSINFO</a> structure variable to obtain operating system information.</p>	Not applicable

## Return Values

**Table 181** • GetSystemInfo Return Values

Return Value	Description
<b>IS_VM_TYPE_HYPERV</b>	<p>The installation is running on a Microsoft Hyper-V machine.</p> <p><b>GetSystemInfo</b> may return this value if VIRTUAL_MACHINE_TYPE was passed to nltem. For more information, see Detecting Whether the Installation Is Being Run on a Virtual Machine.</p>
<b>IS_VM_TYPE_NONE</b>	<p>No virtual machine has been detected.</p> <p><b>GetSystemInfo</b> may return this value if VIRTUAL_MACHINE_TYPE was passed to nltem. For more information, see Detecting Whether the Installation Is Being Run on a Virtual Machine.</p>
<b>IS_VM_TYPE_VMWARE</b>	<p>The installation is running on a VMware product such as VMware Player, VMware Workstation, or VMware Server.</p> <p><b>GetSystemInfo</b> may return this value if VIRTUAL_MACHINE_TYPE was passed to nltem. For more information, see Detecting Whether the Installation Is Being Run on a Virtual Machine.</p>
<b>IS_VM_TYPE_VIRTUALPC</b>	<p>The installation is running on a Microsoft Virtual PC machine.</p> <p><b>GetSystemInfo</b> may return this value if VIRTUAL_MACHINE_TYPE was passed to nltem. For more information, see Detecting Whether the Installation Is Being Run on a Virtual Machine.</p>
<b>IS_VM_TYPE_UNKNOWN</b>	<p>The type of virtual machine is not known.</p> <p><b>GetSystemInfo</b> may return this value if VIRTUAL_MACHINE_TYPE was passed to nltem. For more information, see Detecting Whether the Installation Is Being Run on a Virtual Machine.</p>
<b>0</b>	Indicates that the function successfully returned the specified information.
<b>&lt; 0</b>	Indicates that the function was unable to return the specified information.

## GetSystemInfo Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the GetSystemInfo function.
*
```

```

* This script uses many of the constants available for
* GetSystemInfo and tests for all possible return values.
* The results are displayed in a dialog.
*
\*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_GetSystemInfo(HWND);

function ExFn_GetSystemInfo(hMSI)
    STRING    szTitle, szMsg, svResult, szInfo;
    NUMBER    nvResult;
    LIST      listInfo;
begin

    // Create a list for system information.
    listInfo = ListCreate (STRINGLIST);

    // Get the amount of extended memory.
    if (GetSystemInfo (EXTENDEDMEMORY, nvResult, svResult) < 0) then
        szInfo = "Couldn't get EXTENDEDMEMORY info.";
    else
        Sprintf(szInfo, "Extended memory: %d K", nvResult);
    endif;

    // Add the information to the list.
    ListAddString(listInfo, szInfo, AFTER);

    // Get the boot drive.
    if (GetSystemInfo (BOOTUPDRIVE, nvResult, svResult) < 0) then
        szInfo = "Couldn't get BOOTUPDRIVE info.";
    else
        Sprintf(szInfo, "Boot drive: %s", svResult);
    endif;

    // Add the information to the list.
    ListAddString(listInfo, szInfo, AFTER);

    // Get info about the CD-ROM.
    if (GetSystemInfo (CDROM, nvResult, svResult) < 0) then
        szInfo = "Couldn't get CD-ROM info.";
    else
        if (nvResult = 0) then
            svResult = "No";
        else
            svResult = "Yes";
        endif;

        Sprintf(szInfo, "CDROM: %s", svResult);
    endif;

    // Add the information to the list.
    ListAddString(listInfo, szInfo, AFTER);

    // Get the video adapter.
    if (GetSystemInfo (VIDEO, nvResult, svResult) < 0) then
        szInfo = "Couldn't get VIDEO info.";
    endif;
endfunction

```

```

else
    switch (nvResult)
    case IS_UNKNOWN:
        szInfo = "VIDEO: UNKNOWN";
    case IS_SVGA:
        szInfo = "VIDEO: SVGA";
    case IS_XVGA:
        szInfo = "VIDEO: XVGA";
    case IS_UVGA:
        szInfo = "VIDEO: UVGA";
    endswitch;
endif;

// Add the information to the list.
ListAddString(listInfo, szInfo, AFTER);

// Get number of available colors.
if (GetSystemInfo (COLORS, nvResult, svResult) < 0) then
    szInfo = "Couldn't get COLORS info.";
else
    Sprintf(szInfo, "Number of colors: %d", nvResult);
endif;

// Add the information to the list.
ListAddString(listInfo, szInfo, AFTER);

// Get the current date.
if (GetSystemInfo (DATE, nvResult, svResult) < 0) then
    szInfo = "Couldn't get DATE info.";
else
    Sprintf(szInfo, "DATE: %s", svResult);
endif;

// Add the information to the list.
ListAddString(listInfo, szInfo, AFTER);

// Get the current time.
if (GetSystemInfo (TIME, nvResult, svResult) < 0) then
    szInfo = "Couldn't get TIME info.";
else
    Sprintf(szInfo, "TIME: %s", svResult);
endif;

// Add the information to the list.
ListAddString(listInfo, szInfo, AFTER);

// Get the operating system.
if (GetSystemInfo (OS, nvResult, svResult) < 0) then
    szInfo = "Couldn't get Operating System info.";
else
    switch (nvResult)
    case IS_WINDOWSNT:
        szInfo = "OS: Windows NT";
    case IS_WINDOWS9X:
        GetSystemInfo (WINMINOR, nvResult, svResult);

        if (nvResult < 10) then
            szInfo = "OS: Windows 95";
        else

```

```

        szInfo = "OS: Windows 98";
    endif;
endswitch;
endif;

// Add the information to the list.
ListAddString(listInfo, szInfo, AFTER);

// Display the information.
szTitle  = "System Information";
szMsg    = "The following is some information related to your system:\n";

SdShowInfoList (szTitle, szMsg, listInfo);

ListDestroy(listInfo);

end;

```

## GetTempFileNameIS


The **GetTempFileNameIS** function calls the Windows API **GetTempFileName** to create a temporary file and perform related actions. Note that unlike the Windows API **GetTempFileName**, **GetTempFileNameIS** creates the folder specified by `szPathName` if it does not already exist. Note that as with the **CreateDir** function, the newly created folder or folders are not logged for uninstallation.

### Syntax

```
GetTempFileNameIS( byval string szPathName, byval string szPrefixString, byval number nUnique, byref
    string svTempFileName, byval number nOptions );
```

## Parameters

**Table 182** • GetTempFileNameIS Parameters

Parameter	Description
<b>szPathName</b>	<p>Specifies the path for the lpPathName parameter of the Windows API <a href="#">GetTempFileName</a>.</p> <p>You can use text substitutions for this parameter.</p>
<b>szPrefixString</b>	<p>Specifies the string for the lpPrefixString parameter of the Windows API <a href="#">GetTempFileName</a>.</p> <p>You can use text substitutions for this parameter.</p>
<b>nUnique</b>	<p>Specifies the integer for the nUnique parameter of the Windows API <a href="#">GetTempFileName</a>.</p>
<b>svTempFileName</b>	<p>Specifies the value for the lpTempFileName parameter of the Windows API <a href="#">GetTempFileName</a>.</p> <p></p> <p><b>Note</b> • InstallScript strings are automatically <code>_MAX_PATH</code> or greater in length. Therefore, there is no need to manually size this string.</p>
<b>nOptions</b>	<p>Specifies an InstallScript-specific option. Pass any of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>● <b>GTFIS_OPTION_NONE</b>—No InstallScript-specific options. This is the default option.</li> <li>● <b>GTFIS_OPTION_DONT_RESOLVE_TEXTSUBS</b>—Do not call <b>TextSubSubstitute</b> to resolve text substitutions in <b>szPathName</b> and <b>szPrefixString</b>.</li> <li>● <b>GTFIS_OPTION_DONT_CREATE_DIR</b>—Do not create the directory specified by <b>szPathName</b> if it does not already exist. Note that if this option is specified and <b>szPathName</b> does not exist, the function fails.</li> <li>● <b>GTFIS_OPTION_DELETE_TEMP_FILE</b>—Delete the created temporary file before returning. (Note that this option is useful only if you specify <b>nUnique</b> such that the function creates a temporary file. For more information, see <a href="#">GetTempFileName Function</a>.) The function does not return failure if the temporary file cannot be deleted.</li> <li>●</li> </ul>

## Return Values

**Table 183** • GetTempFileNameS Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the function was successful.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was not successful.

# GetTrueTypeFontFileInfo

The **GetTrueTypeFontFileInfo** function returns in `svResult` information about the TrueType font file that is specified by `szTrueTypeFontFile`.

## Syntax

```
GetTrueTypeFontFileInfo ( szTrueTypeFontFile, nInfo, nLanguage, svResult );
```

## Parameters

**Table 184** • GetTrueTypeFontFileInfo Parameters

Parameter	Description
<b>szTrueTypeFontFile</b>	Specifies the fully qualified file name of the TrueType font file from which you want information. Note that non-TrueType files such as .fon and .fot files are not supported by this function, nor are TrueType collection files (.ttc files) supported.
<b>nInfo</b>	Specifies the information to be returned in svResult. Pass one of the following constants in this parameter: <ul style="list-style-type: none"> <li>• <b>TTFONTFILEINFO_FONTTITLE</b>—Specifies the font's title.</li> <li>• <b>name ID</b>—You can use a name ID to get <a href="#">font information</a>.</li> <li>•</li> </ul>
<b>nLanguage</b>	Specifies the language of the string to be returned in svResult. Pass a predefined language constant, numeric language ID, the system variable <a href="#">SELECTED_LANGUAGE</a> , or 0 (zero)—which has the same effect as passing <a href="#">SELECTED_LANGUAGE</a> —in this parameter. Note that not all information for a particular font may be available in all languages.
<b>svResult</b>	Returns the font information that you specified in nInfo. An ANSI (non-Unicode) string is returned even if the data in the file is stored in a Unicode format.

## Return Values

**Table 185** • GetTrueTypeFontFileInfo Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the function successfully retrieved the requested information.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was unable to retrieve the requested information.

# GetUpdateStatus

The **GetUpdateStatus** function is obsolete. If this function is called, it returns FALSE.

## Syntax

```
BOOL GetUpdateStatus();
```



# GetUpdateStatusReboot

The **GetUpdateStatusReboot** function is obsolete. If this function is called, it returns FALSE.

## Syntax

```
BOOL GetUpdateStatusReboot();
```

# GetValidDrivesList

The **GetValidDrivesList** function retrieves a list of all the drives attached to the target system that meet a certain criterion. This criterion includes the type of drive and the minimum amount of space on the drive. If a drive door is open, the drive name is still inserted into the list.

## Syntax

```
GetValidDrivesList (listID, nDriveType, nMinDriveSpace);
```

## Parameters

**Table 186** • GetValidDrivesList Parameters

Parameter	Description
<b>listID</b>	Returns a list of valid drive letters. The string list identified by listID must already have been initialized by a call to <a href="#">ListCreate</a> .
<b>nDriveType</b>	Specifies the type of drive to search for. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>-1</b>—Searches for all drive types.</li> <li>• <b>FIXED_DRIVE</b>—Searches only for fixed drives.</li> <li>• <b>REMOTE_DRIVE</b>—Searches only for remote drives. Remote drives are generally located on a network.</li> <li>• <b>REMOVEABLE_DRIVE</b>—Searches only for removable drives. Floppy drives are removable drives.</li> <li>• <b>CDROM_DRIVE</b>—Searches only for CD-ROM drives.</li> </ul>
<b>nMinDriveSpace</b>	Specifies the minimum amount of disk space in bytes that must be free on the drive to allow the drive to be included in the return list. If nMinDriveSpace is less than zero, GetValidDrivesList will not check for the minimum space on the drive. This is useful for floppy drives.

## Return Values

**Table 187** • GetValidDrivesList Return Values

Return Value	Description
<b>0</b>	GetValidDrivesList successfully retrieved the requested list.
<b>&lt; 0</b>	GetValidDrivesList was unable to retrieve the list.

## Additional Information

- You can specify the type of drive to search for and the minimum amount of disk space that must be available before the drive is listed.
- Network mapping drives can be returned as remote drives. GetValidDrivesList might not return all drives on the network. Those drives designated as mapping drives only are returned.

## GetValidDrivesList Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the GetValidDrivesList function.
*
* GetValidDrivesList is called twice: once to return a list of
* removable drives that have a minimum of 120,000 bytes free,
* then again to return a list of fixed drives that have a
* minimum of 1,000,000 bytes free.
*
\*-----*/

#define TITLE          "GetValidDrivesList Example"
#define MSG_REMOVABLE  "Removable drives with 120,000 bytes free:"
#define MSG_FIXED      "Fixed drives with 1,000,000 bytes free."
#define MSG_ERR        "GetValidDrivesList failed."

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_GetValidDrivesList(HWND);

function ExFn_GetValidDrivesList(hMSI)
    LIST    listID;
begin

    // Create a list to hold the removable drive names.
    listID = ListCreate (STRINGLIST);

    // Get removable drives with at least 120,000 bytes free.
    if (GetValidDrivesList (listID, REMOVEABLE_DRIVE, 120000) < 0) then
        // Report an error; then terminate.
        MessageBox (MSG_ERR, SEVERE);
        abort;
    else
        // Display the list of removable drives.
        SdShowInfoList (TITLE, MSG_REMOVABLE, listID);
    endif;

    // Destroy the list of removable drives.
    ListDestroy (listID);

    // Create a list to hold the fixed drive names.
    listID = ListCreate (STRINGLIST);

    // Get fixed drives with at least one million bytes free.
    if (GetValidDrivesList (listID, FIXED_DRIVE, 1000000) < 0) then
        // Report an error; then terminate.
        MessageBox (MSG_ERR, SEVERE);
        abort;
    else
        // Display the list of fixed drives.
        SdShowInfoList (TITLE, MSG_FIXED, listID);
    endif;

end;

```

# GetWCHARArrayFromISStringArray

The **GetWCHARArrayFromISStringArray** function returns a pointer to an array of pointers to Unicode character strings that are contained in the specified array.

## Syntax

```
GetWCHARArrayFromISStringArray ( vArray );
```

## Parameters

**Table 188** • GetWCHARArrayFromISStringArray Parameters

Parameter	Description
<b>vArray</b>	Specifies the string array to which you want a pointer.

## Return Values

**Table 189** • GetWCHARArrayFromISStringArray Return Values

Return Value	Description
<b>pointer</b>	A pointer to an array of pointers to Unicode character strings.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function failed.

## Additional Information

The **GetWCHARArrayFromISStringArray** function allocates additional memory for the array of pointers and the Unicode character strings. This pointer can be passed to functions that take LPWSTR\* arguments. After you are done with the newly created array, call [DeleteWCHARArray](#) to delete the array from memory.

If you call [CopyCHARArrayToISStringArray](#) to write the data from the array of pointers back to the original string array, be careful when modifying strings that are contained in the array. Since the lengths of the strings that are contained in string arrays are managed internally by the installation, if you change the length of a string the entire string will not be copied back to the original array when you call **CopyCHARArrayToISStringArray**.

# GetWindowHandle

The **GetWindowHandle** function gets the handle of the main window of the installation.

## Syntax

```
GetWindowHandle ( nHwndFlag );
```

## Parameters

**Table 190** • GetWindowHandle Parameters

Parameter	Description
<b>nHwndFlag</b>	Specifies the window handle of InstallShield's main window. Pass the predefined constant <code>HWND_INSTALL</code> in this parameter.

## Return Values

**Table 191** • GetWindowHandle Parameters

Return Value	Description
<b>X</b>	Where X is the handle of the window.
<b>&lt; 0</b>	GetWindowHandle was unable to retrieve the handle.

## GetWindowHandle Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the GetWindowHandle function.
 *
 * This script starts the setup in a normal window. After three
 * seconds, the window is minimized. Then after another pause,
 * the window is maximized and a message box is displayed.
 *
 \*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_GetWindowHandle(HWND);

function ExFn_GetWindowHandle(hMSI)
    NUMBER nHwnd;
    HWND hInstallHwnd;
begin

    // Specify a standard window for this setup.
    Enable (DEFWINDOWMODE);

    // Display the background window.
    Enable (BACKGROUND);

    // Get the setup's window handle.
    nHwnd = GetWindowHandle (HWND_INSTALL);

    // Wait three seconds.
    Delay (3);

```

```
// Send system command to minimize the window.
SendMessage (nHwnd, WM_SYSCOMMAND, SC_MINIMIZE, 0);

// Wait three seconds.
Delay (3);

// Send system command to maximize the window.
SendMessage (nHwnd, WM_SYSCOMMAND, SC_MAXIMIZE, 0);

// Display a message.
MessageBox ("Demo completed.", INFORMATION);

end;
```

# Built-In Functions (H-P)

For a list of functions by category, see [Built-In Functions by Category](#).

## Handler

The **Handler** function is obsolete. Use [HandlerEx](#) instead.

## HandlerEx

The **HandlerEx** function creates custom handlers for events such as the Help accelerator key (F1) and the Cancel button.

If the end user presses the F1 key, the currently defined HELP handler is executed. If the user presses the Cancel button, the currently defined EXIT handler is executed. If you have not defined custom HELP or EXIT handlers using the **HandlerEx** function, the default handlers are executed. The default EXIT handler displays the Exit dialog. The default HELP handler does nothing.

To execute custom handlers defined using **HandlerEx**, the InstallScript engine calls a unique label that is specified in the parameter Label when the nObject event occurs. When the InstallScript engine reaches a return statement in the handler code (under the label), control returns to the statement that would have executed next if the handler label were not called.


Using **HandlerEx**, you can specify custom handling of EXIT or HELP. You can display the **MessageBox**, **SprintfBox**, and **AskYesNo** dialogs inside Exit handlers; however, you cannot display Sd dialogs.

### Syntax

```
HandlerEx (nObject, Label);
```

## Parameters

**Table 1 • HandlerEx Parameters**

Parameter	Description
<b>nObject</b>	<p>Specifies the event to trap. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>EXIT</b>—Specifies that a custom handler is called when the Cancel button is pressed. If you do not define a handler, a default Exit dialog appears when the Cancel button is pressed.</li> <li>• <b>HELP</b>—Specifies that a custom handler is called when the F1 accelerator key is pressed. If you do not define a handler, nothing happens when the F1 accelerator key is pressed.</li> </ul>
<b>Label</b>	<p>Specifies the name of the label to which the program should jump if the specified button or accelerator key is pressed. Do not define this label as a numeric value or as a string variable.</p> <p>To cancel a currently defined handler and reinstall the default handler, pass -1 in this parameter. This method is useful in scripts that install a custom handler only for certain processes and then revert to the default handler.</p> <p> <b>Caution •</b> Do not define the label name as a numeric value or as a string variable.</p>

## Return Values

**Table 2 • HandlerEx Return Values**

Return Value	Description
<b>0</b>	<b>HandlerEx</b> successfully created the handle.
<b>&lt; 0</b>	<b>HandlerEx</b> was unable to create the handle.

## Additional Information

- The only accelerator available is the F1 function key (Help).
- Help (F1) allows you to launch the help engine or provide other suitable help. When the end user presses the F1 key, the InstallScript engine calls the currently defined Help handler. You can provide any type of help functionality in the Help handler. If you want to provide context-sensitive help, you must keep track of the context in your script.

For example, you can have a string variable that contains a current context string. You can then use that string variable in a switch-case statement inside the Help handler to execute the appropriate Help events based on the value of the context string. You can also test the value of the nSdDialog global variable in your Help handling routine. nSdDialog is set to the dialog ID (as described in **\_isres.h** in the *InstallShield Program Files*



*Folder\Script\Isrt\Include* folder) of the currently executing Sd dialog. (When no Sd dialog is executing, *nSdDialog* is not defined.) You can therefore give the user access to Sd dialog-specific help when an Sd dialog is executing.

- As with Help handlers, you can also define and execute custom and Sd dialog-specific EXIT handlers.

## HandlerEx Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the HandlerEx function.
 *
 * This script shows how to install error and help handlers in
 * an installation.
 *
 * First, the handlers are installed. Then Sd dialogs are
 * displayed to the end user. The handler code is invoked if
 * the end user presses F1 (help) or clicks the cancel
 * button while the dialogs are displayed.
 *
 * Note that the script uses the InstallScript Language
 * Reference and Help Library help files for demonstration purposes.
 * Insert copies of these files (Langref.chm and HelpLib.chm from
 * the InstallShield Program Files Folder\Program folder) in the
 * Language Independent area of the Support Files/Billboards view.
 \*-----*/

// Included header files -----
#include "ifx.h"

export prototype void ExFn_Handler();
prototype OnHelp_Handler();
prototype OnExit_Handler();

// Define help topic IDs
#define HELP_WELCOME      101
#define HELP_REGISTERUSER 102

INT      nHelpID;

function void ExFn_Handler()
    STRING svName, svCompany, szMsg;
begin
    // Install the help handler.
    HandlerEx (HELP, OnHelp_Handler);

    // Install the exit handler.
    HandlerEx (EXIT, OnExit_Handler);

    // Set message for display in dialogs.
```

```

        szMsg = "Press F1 to display InstallShield's " +
            "help for this dialog.\n" +
            "Press Cancel to invoke the custom exit handler.";

WelcomeDialog:
    // Set the help topic ID.
    nHelpID = HELP_WELCOME;

    // Display the Welcome dialog.
    SdWelcome ("SdWelcome Dialog", szMsg);

    // Set the help topic ID.
    nHelpID = HELP_REGISTERUSER;

    if SdRegisterUser ("Register", szMsg, svName, svCompany) = BACK then
        // Go back to the Welcome dialog.
        goto WelcomeDialog;
    endif;

end;

function OnHelp_Handler()
    STRING szHelpTopic;
begin
    // Set the topic to be displayed from the help.
    switch (nHelpID)
        case HELP_WELCOME      : szHelpTopic = "5223 ";
        case HELP_REGISTERUSER : szHelpTopic = "5211 ";
    endswitch;

    // Launch the InstallScript language reference and
    // display the selected help topic.

    LaunchApplication (WINDIR ^ "hh.exe", "-mapid " + szHelpTopic +
        SUPPORTDIR ^ "Help_Lib.chm",
        "", SW_SHOW, INFINITE, LAAW_OPTION_WAIT);

end;

function OnExit_Handler()
begin
    // Ask for confirmation to abort.
    if AskYesNo ("Do you really want to exit?", FALSE) = YES then
        // Abort the setup.
        abort;
    else
        // Return to the setup.
        return 0;
    endif;
end;

```

## HIBYTE



**Project** • This information applies to InstallScript projects.

The **HIBYTE** function extracts the high-order byte from the 16-bit integer value specified by shValue.

## Syntax

```
HIBYTE ( shValue );
```

## Parameters

**Table 3 • HIBYTE Parameters**

Parameter	Description
<b>shValue</b>	Specifies the 16-bit integer from which you want to extract the higher byte.

## Return Values

This function returns the high-order byte of the integer.

# HIWORD

The **HIWORD** function extracts and returns the high-order word (upper two bytes) from the 32-bit integer value specified by IValue.

InstallShield's HIWORD differs from the corresponding C macro in that it uses sign extension. As a result, the high-order bytes of the value returned by HIWORD are filled with ones if IValue is negative. If necessary, you can use the bitwise AND operator (&) to combine the result with 0xFFFF to produce a positive value, as shown below:

```
lValue = HIWORD(lValue);
lValue = lValue & 0xFFFF;
```

## Syntax

```
HIWORD ( lValue );
```

## Parameters

**Table 4 • HIWORD Parameters**

Parameter	Description
<b>lValue</b>	Specifies the 32-bit integer from which to extract the upper two bytes.

## Return Values

HIWORD returns the high-order word (upper two bytes) of IValue.

## HIWORD Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates HIWORD and LOWORD.
 *
 * This example script shows how to use HIWORD and LOWORD to
 * to get the low-order word and the high-order word from a value.
 \*-----*/

#define TITLE_TEXT "LOWORD/HIWORD Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_HIWORD(HWND);

function ExFn_HIWORD(hMSI)
    STRING  szMsg;
    NUMBER  nData, nLOWORD, nHIWORD;
begin

    nData = 305419896; // hex value: 12345678
    // Get the low-order word, 22136 (hex value: 5678).

    nLOWORD = LOWORD (nData);

    // Get the high-order word, 4660 (hex value: 1234).
    nHIWORD = HIWORD (nData);

    // Display the results.
    szMsg = "LOWORD: %ld\nHIWORD: %ld";
    sprintfBox (INFORMATION, TITLE_TEXT, szMsg, nLOWORD, nHIWORD);

end;
```

## InstallationInfo

The **InstallationInfo** function is obsolete. Use the [CreateInstallationInfo](#) function instead.

### Syntax

```
InstallationInfo (szCompany, szProduct, szVersion, szProductKey);
```

## Is

The **Is** function retrieves information commonly needed in a script.

## Syntax


```
Is ( nIsFlag, szIsData );
```

## Parameters

**Table 5 • Is Parameters**

Parameter	Description
<b>nlsFlag</b>	<p>Specifies the type of information to retrieve. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>BACKBUTTON</b>—Is the Back button that is displayed in some built-in dialogs enabled?</li> <li>• <b>CANCELBUTTON</b>—Is the Cancel button that is displayed in some built-in dialogs enabled?</li> <li>• <b>DOTNETFRAMEWORKINSTALLED</b>—Is a particular version of the .NET Framework or language pack installed? For more information on the predefined values that are supported, see the <a href="#">Additional Information</a> section.</li> <li>• <b>DOTNETSERVICEPACKINSTALLED</b>—Is a particular service pack—or a later version of the service pack—of the .NET Framework installed? For more information on the predefined values that are supported, see the <a href="#">Additional Information</a> section.</li> <li>• <b>DIR_WRITEABLE</b>—Can the installation write to the directory that is specified in szIsData?</li> <li>• <b>FILE_EXISTS</b>—Does the file that is specified in szIsData exist?</li> <li>• <b>FILE_LOCKED</b>—Is the file locked? Note that <b>Is</b> returns TRUE if the file is not accessible because of insufficient privileges.</li> <li>• <b>FILE_WRITEABLE</b>—Can the installation write to the file specified in szIsData?</li> <li>• <b>FONT_AVAILABLE</b>—Is the font with the title that is specified in szIsData installed? Note that the function searches for the font in all character sets. To search in a single character set, call the Windows API function <b>EnumFontFamiliesEx</b>; for details on this function, see Microsoft's Windows API documentation.</li> <li>• <b>FUNCTION_EXPORTED</b>—Does the DLL that is specified in szIsData export the function specified in szIsData? <b>Is</b> returns TRUE if the DLL exists and can be loaded, and if the function is exported. Otherwise, <b>Is</b> returns FALSE.</li> <li>• <b>LANGUAGE_SUPPORTED</b>—(InstallScript and InstallScript MSI projects) Does the installation support the language that is specified in szIsData?</li> <li>• <b>LOGGING</b>—(InstallScript projects only) Is logging of uninstallation information enabled?</li> <li>• <b>MATH_COPROCESSOR</b>—Does a math coprocessor exist in the target system?</li> <li>• <b>NEXTBUTTON</b>—Is the Next button that is displayed by some built-in dialogs enabled? On most dialogs, the Next button is enabled by default.</li> <li>• <b>PATH_EXISTS</b>—Does the path that is specified in szIsData exist?</li> <li>• <b>REGDBREMOTEREGCONNECTED</b>—Is a remote registry currently connected?</li> <li>• <b>REBOOTED</b>—Is the installation running after a reboot?</li> </ul>

Table 5 • Is Parameters (cont.)

Parameter	Description
<b>nIsFlag (cont.)</b>	<ul style="list-style-type: none"> <li>• <b>SETUP_PACKAGE</b>—(InstallScript projects only) Is the setup running from a self-extracting executable file?  During maintenance mode or during uninstallation, <code>Is(SETUP_PACKAGE, szIsData)</code> returns TRUE if the original installation was a self-extracting executable file (although maintenance or uninstallation runs not from the self-extracting executable file, but from the copy of <b>Setup.exe</b> in the DISK1TARGET folder).</li> <li>• <b>SKIN_LOADED</b>—Is a dialog skin loaded?</li> <li>• <b>URL</b>—Is the string that is specified in <code>szIsData</code> a URL? (That is, does the string begin with "http://" or "https://" or "file://" or "ftp://")</li> <li>• <b>USER_ADMINISTRATOR</b>—Does the current user have administrator privileges? <b>Is</b> returns TRUE when <code>USER_ADMINISTRATOR</code> is passed in <code>nFlag</code> in all cases, except for some cases on Windows Vista and later systems; on these later systems, <b>Is</b> returns TRUE if the <code>USER_ADMINISTRATOR</code> is passed in <code>nFlag</code> and the <code>SE_GROUP_USE_FOR_DENY_ONLY</code> security identifier (SID) attribute is not set for the group. That is, if the current user is in the Administrators group but that user is running the installation with a standard access token on Windows Vista or later, <b>Is</b> returns FALSE.</li> <li>• <b>USER_INADMINGROUP</b>—Is the current user in the Administrators group? <b>Is</b> returns TRUE for this constant, regardless of whether the <code>SE_GROUP_USE_FOR_DENY_ONLY</code> SID attribute is set for the group (that is, regardless of whether the user with administrative privileges is running the installation with a standard access token).</li> <li>• <b>USER_POWERUSER</b>—Does the current user belong to the Power Users group?</li> <li>• <b>VALID_PATH</b>—Is the path specified in <code>szIsData</code> a legal path? This will not confirm the existence of a path, it will only check its syntax. You can use this constant when you retrieve path information from the user. The function will then check to see if the path information was entered correctly.  A valid URL string meets the following criteria: (1) It begins with <b>http://</b>, <b>https://</b>, or <b>file://</b>. (2) It contains only numbers, letters, and the following characters: ! \$ % &amp; ' ( ) * + - . / \ _</li> </ul>
<b>nIsFlag (cont.)</b>	<ul style="list-style-type: none"> <li>• <b>WEB_BASED_SETUP</b>—(InstallScript projects only) Is the installation being run from the Internet?</li> </ul> <div>  <p><b>Note</b> • <i>Is(WEB_BASED_SETUP, szIsData)</i> checks whether the current instance of the installation is being run from the Web. For this reason, when the installation is run from Add or Remove Programs, <i>Is(WEB_BASED_SETUP, szIsData)</i> always returns FALSE, since in that case the installation is always being run from the local files even if the original installation was run from the Web.</p> </div> <ul style="list-style-type: none"> <li>• <b>WINDOWS_SHARED</b>—Is Microsoft Windows running a shared copy from a network?  A shared copy of Microsoft Windows is installed on a network and has common files that are shared by many users.</li> </ul>

**Table 5 • Is Parameters (cont.)**

Parameter	Description
<b>szIsData</b>	<p>Specifies information that is dependent on the constant that is passed in <b>nlsFlag</b>, as shown below. Note that if the path or file name is enclosed in quotation marks, <b>Is</b> fails. To ensure that the path or file name is not enclosed in quotation marks, call <b>LongPathToQuote(szIsData, FALSE)</b> before calling <b>Is</b>. The following list provides an explanation of what <b>szIsData</b> should contain when each <b>nlsFlag</b> option is specified:</p> <ul style="list-style-type: none"> <li>• <b>DIR_WRITEABLE</b>—<b>szIsData</b> specifies the fully qualified path to be checked.</li> <li>• <b>DOTNETFRAMEWORKINSTALLED</b>—<b>szIsData</b> specifies the version of the .NET Framework or language pack to check. For more information on the predefined values that are supported, see the <a href="#">Additional Information</a> section.</li> <li>• <b>DOTNETSERVICEPACKINSTALLED</b>—<b>szIsData</b> specifies the minimum service pack and version of the .NET Framework to check, in the following format: <p><i>Service Pack Number .NET Version Registry Constant or Path</i></p> <p><i>Service Pack Number</i> indicates the number of the minimum service pack. For the .NET Framework 1.0, the value can be 0 through 3. For the .NET Framework 1.1 and later, all numeric values are supported. Both the service pack number and the pipe character ( ) delimiter must be included; otherwise, the function returns failure.</p> <p><i>.NET Version Registry Constant or Path</i> indicates the registry constant or registry path that shows the version of the .NET Framework to check. For more information on the predefined values that are supported and to learn how the <b>Is</b> function performs the check, see the <a href="#">Additional Information</a> section.</p> </li> <li>• <b>FILE_EXISTS</b>—<b>szIsData</b> specifies the fully qualified file name. You can specify a valid URL in this parameter. To check the validity of a URL, call <b>Is(VAILD_PATH, szIsData)</b>.</li> <li>• <b>FILE_LOCKED</b>—<b>szIsData</b> specifies the fully qualified file name.</li> <li>• <b>FILE_WRITEABLE</b>—<b>szIsData</b> specifies the fully qualified file name.</li> <li>• <b>FONT_AVAILABLE</b>—<b>szIsData</b> specifies the font title.</li> <li>• <b>FUNCTION_EXPORTED</b>—<b>szIsData</b> specifies the fully qualified file name of the DLL followed by a pipe character ( ) and the name of the function; for example: <p><i>C:\MyDLLFolder\MyDLL.dll MyFunction</i></p> </li> <li>• <b>LANGUAGE_SUPPORTED</b>—<b>szIsData</b> specifies the path to the <b>Setup.ini</b> file and the language ID, in the following format: <p><i>Path to Setup.ini Language ID</i></p> <p>If the path to <b>Setup.ini</b> is not specified, the current installation is used. If the language ID is not specified, <b>SELECTED_LANGUAGE</b> is used.</p> <p>The language ID should be a four-digit hexadecimal language code, including the 0x prefix. For example, if for English, the value should be 0x0409. To build a string with <b>STANDARD_SELECTED_LANGUAGE</b> in this format, use a statement such as:</p> <pre>Sprintf (szLang, "0x%.04lx", STANDARD_SELECTED_LANGUAGE);</pre> </li> </ul>



**Table 5 • Is Parameters (cont.)**

Parameter	Description
<b>szIsData (cont.)</b>	<ul style="list-style-type: none"> <li>• <b>LOGGING</b>—szIsData is ignored.</li> <li>• <b>MATH_COPROCESSOR</b>—szIsData specifies szIsData is ignored.</li> <li>• <b>PATH_EXISTS</b>—szIsData specifies the fully qualified path. You can specify a valid URL in this parameter. To check the validity of a URL, call Is(VALID_PATH, szIsData).</li> <li>• <b>REGDBREMOTEREGCONNECTED</b>—szIsData is ignored.</li> <li>• <b>SETUP_PACKAGE</b>—szIsData is ignored.</li> <li>• <b>SKIN_LOADED</b>—szIsData is ignored.</li> <li>• <b>USER_ADMINISTRATOR</b>—szIsData is ignored.</li> <li>• <b>USER_POWERUSER</b>—szIsData is ignored.</li> <li>• <b>VALID_PATH</b>—szIsData specifies the fully qualified path.</li> <li>• <b>WEB_BASED_SETUP</b>—szIsData is ignored.</li> <li>• <b>WINDOWS_SHARED</b>—szIsData is ignored.</li> </ul>

## Return Values

**Table 6 • Is Return Values**

Return Value	Description
<b>TRUE (1)</b>	Indicates that the answer is true.
<b>FALSE (0)</b>	Indicates that the answer is false.
<b>&lt; 0</b>	The Is function was unable to answer the question.

## Additional Information

### .NET Framework Version and Service Details

The following predefined constants are supported for specifying a particular version of the .NET Framework using DOTNETFRAMEWORKINSTALLED or DOTNETSERVICEPACKINSTALLED:

- **REGDB\_KEYPATH\_DOTNET\_40\_CLIENT**
- **REGDB\_KEYPATH\_DOTNET\_40\_FULL**
- **REGDB\_KEYPATH\_DOTNET\_35**
- **REGDB\_KEYPATH\_DOTNET\_30\_SP**—Use this variable to detect whether the SP1 (or a later service pack) of the .NET Framework 3.0 is installed.
- **REGDB\_KEYPATH\_DOTNET\_30**—Use this variable to detect whether the RTM version of the .NET Framework 3.0 is installed.
- **REGDB\_KEYPATH\_DOTNET\_20**

- [REGDB\\_KEYPATH\\_DOTNET\\_11](#)
- [REGDB\\_KEYPATH\\_DOTNET\\_10](#)



**Tip** • Since each of these predefined constants correspond to the appropriate registry path under `HKEY_LOCAL_MACHINE`, you can also specify a registry path directly in `szIsData` for `DOTNETFRAMEWORKINSTALLED` or `DOTNETSERVICEPACKINSTALLED`.

The installation of the .NET Framework 3.0 writes the registry value `InstallSuccess`, with 1 as the value data, in the following registry location:

**`HKEY_LOCAL_MACHINE\Software\Microsoft\NET Framework Setup\NDP\v3.0\Setup\`**

Therefore, `REGDB_KEYPATH_DOTNET_30` is set to that location.

The installations of all other versions of the .NET Framework write the registry value `Install` with a value data of 1 in the following registry location:

**`HKEY_LOCAL_MACHINE\Software\Microsoft\NET Framework Setup\NDP\Version Number\`**

The `REGDB_KEYPATH_DOTNET_35`, `REGDB_KEYPATH_DOTNET_20`, `REGDB_KEYPATH_DOTNET_11`, and `REGDB_KEYPATH_DOTNET_10` constants are set to the appropriate values, based on that path.

Note that the installation of the .NET Framework 3.0 SP1 also writes the registry value `Install` with a value data of 1 in the following registry location:

**`HKEY_LOCAL_MACHINE\Software\Microsoft\NET Framework Setup\NDP\v3.0\`**

Therefore, `REGDB_KEYPATH_DOTNET_30_SP` is set to that location.

If you use the `Is` function with the `DOTNETFRAMEWORKINSTALLED` constant, the function automatically checks for both the `Install` and `InstallSuccess` values. If the `Install` or `InstallSuccess` value exists and its value data is set to 1, `Is` returns `TRUE`. Otherwise, `Is` returns `FALSE`.

The .NET Framework 1.0 service pack installations do not create registry values that indicate the service pack number. Therefore, if you use `DOTNETSERVICEPACKINSTALLED` to test for the .NET Framework 1.0, the `Is` function compares the version of the `mscorlib.dll` file in `FOLDER_DOTNET_10` against known version numbers to determine the service pack:

- SP3 is present if the version of `mscorlib.dll` is 1.0.3705.6018 or later.
- SP2 is present if the version of `mscorlib.dll` is 1.0.3705.288 or later, but earlier than 1.0.3705.6018.
- SP1 is present if the version of `mscorlib.dll` is 1.0.3705.209 or later, but earlier than 1.0.3705.288.
- The original RTM version is present if the version of `mscorlib.dll` is 1.0.3705.0 or later, but earlier than 1.0.3705.209.

If you use `DOTNETSERVICEPACKINSTALLED` to test for any of the other .NET Framework versions, the function compares the `REGDB_VALUENAME_SP` value under the .NET version registry constant or path that is specified in `szIsData` with the service pack number that is specified in `szIsData`. Note that the comparison is an equal-to-or-later-than version comparison; therefore, if you specify a service pack of 2 in `szIsData`, the function returns `true` whenever service pack 2 or later is installed.

## .NET Framework Language Pack Details

Version 1.1 and later of the .NET Framework includes support for language packs; version 1.0 does not. For version 1.1 and later of the .NET Framework, you can test whether a particular .NET language pack is installed through the DOTNETFRAMEWORKINSTALLED constant by specifying the appropriate .NET version constant and the locale identifier (LCID) of the language (converted to a string). Separate the .NET version constant and the LCID with the caret operator. For example, use the following syntax to test whether the German language pack for .NET Framework 1.1 is installed:

```
NumToStr( szLang, ISLANG_GERMAN_STANDARD );
```

```
REGDB_KEYPATH_DOTNET_11 ^ szLang;
```

As documented by Microsoft, the .NET Framework 1.1 supports the following LCIDs:

**Table 7 • Supported .NET LCIDs**

Language	LCID	Corresponding Is Constant
<b>Chinese (Simplified)</b>	2052 (0x0804)	ISLANG_CHINESE_SIMPLIFIED
<b>Chinese (Traditional)</b>	1028 (0x0404)	ISLANG_CHINESE_TRADITIONAL
<b>Czech</b>	1029 (0x0405)	ISLANG_CZECH_STANDARD
<b>Danish</b>	1030 (0x0406)	ISLANG_DANISH_STANDARD
<b>Dutch</b>	1043 (0x0413)	ISLANG_DUTCH_STANDARD
<b>Finnish</b>	1035 (0x040B)	ISLANG_FINNISH_STANDARD
<b>French</b>	1036 (0x040C)	ISLANG_FRENCH_STANDARD
<b>German</b>	1031 (0x0407)	ISLANG_GERMAN_STANDARD
<b>Greek</b>	1032 (0x0408)	ISLANG_GREEK_STANDARD
<b>Hungarian</b>	1038 (0x040E)	ISLANG_HUNGARIAN_STANDARD
<b>Italian</b>	1040 (0x0410)	ISLANG_ITALIAN_STANDARD
<b>Japanese</b>	1041 (0x0411)	ISLANG_JAPANESE_STANDARD
<b>Korean</b>	1042 (0x0412)	ISLANG_KOREAN_STANDARD
<b>Norwegian</b>	1044 (0x0414)	ISLANG_NORWEGIAN_BOKMAL
<b>Polish</b>	1045 (0x0415)	ISLANG_POLISH_STANDARD
<b>Portuguese (Brazilian)</b>	1046 (0x0416)	ISLANG_PORTUGUESE_BRAZILIAN
<b>Portuguese (Portugal)</b>	2070 (0x0816)	ISLANG_PORTUGUESE_STANDARD
<b>Russian</b>	1049 (0x0419)	ISLANG_RUSSIAN_STANDARD

**Table 7 •** Supported .NET LCIDs (cont.)

Language	LCID	Corresponding Is Constant
<b>Spanish</b>	3082 (0x0C0A)	ISLANG_SPANISH_MODERNSORT
<b>Swedish</b>	1053 (0x041D)	ISLANG_SWEDISH_STANDARD
<b>Turkish</b>	1055 (0x041F)	ISLANG_TURKISH_STANDARD

## Is Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the Is function.
*
* The Is function is first called to check if a file is write
* protected. It is called a second time to check if the
* directory is write protected. This is useful for checking
* the write protection of directories on a network. The third
* time the Is function is called, the function checks if Windows
* is installed on a network server or if it is installed on the
* local system.
*
* Note: Before running this script, set the defined constants
*       so that they point to an existing file on the target
*       system.
*
\*-----*/

#define EXAMPLE_DIR    "C:\\WINDOWS"
#define EXAMPLE_FILE   EXAMPLE_DIR^"Win.com"
#define TITLE          "Is Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_Is(HWND);

function ExFn_Is(hMSI)
    NUMBER nResult;
begin

    // Check if the EXAMPLE_FILE file is write-protected.
    nResult = Is (FILE_WRITEABLE, EXAMPLE_FILE);

    // Report the results.
    if (nResult = TRUE) then
        sprintfBox (INFORMATION, TITLE, "%s is writeable.", EXAMPLE_FILE);

```

```

elseif (nResult = FALSE) then
    sprintfBox (INFORMATION, TITLE, "%s is not writeable.", EXAMPLE_FILE);
else
    sprintfBox (INFORMATION, TITLE,
        "Unable to determine if %s is writeable.", EXAMPLE_FILE);
endif;

// Check if the directory is write-protected.
nResult = Is (DIR_WRITEABLE, EXAMPLE_DIR);

// Report the results.
if (nResult = TRUE) then
    sprintfBox (INFORMATION, TITLE, "%s is writeable.", EXAMPLE_DIR);
elseif (nResult = FALSE) then
    sprintfBox (INFORMATION, TITLE, "%s is not writeable.", EXAMPLE_DIR);
else
    sprintfBox (INFORMATION, TITLE,
        "Unable to determine if %s is writeable.", EXAMPLE_DIR);
endif;

// Check if Windows is being shared on a network.
nResult = Is (WINDOWS_SHARED, "");

// Report the results.
if (nResult = TRUE) then
    MessageBox ("Windows is shared.", INFORMATION);
elseif (nResult = FALSE) then
    MessageBox ("Windows is not shared.", INFORMATION);
else
    MessageBox ("Unable to determine if Windows is shared.", SEVERE);
endif;

end;

```

## ISCompareServicePack

The **ISCompareServicePack** function is supported only for compatibility with scripts created in InstallShield Professional. It is recommended that you determine the Windows service pack number by checking the value of `SYSINFO.WINNT.nServicePack` instead.

The **ISCompareServicePack** function compares the service pack number installed on a Windows system to the service pack number specified by `szServicePack`.

### Syntax

```
ISCompareServicePack ( szServicePack );
```

## Parameters

**Table 8 • ISCompareServicePack Parameters**

Parameter	Description
<b>szServicePack</b>	Specifies the Service Pack number to be compared with the Service Pack number on the target computer. This string must be in the format "Service Pack n", where n is the Service Pack number. The comparison is case sensitive.

## Return Values

**Table 9 • ISCompareServicePack Return Values**

Return Value	Description
<b>LESS_THAN (1)</b>	No Service Pack is installed or the Service Pack number on the target system is less than the value passed in szServicePack.
<b>EQUALS (2)</b>	The Service Pack numbers match.
<b>GREATER_THAN (0)</b>	The Service Pack number on the target system is greater than the number passed in szServicePack.
<b>-1</b>	ISCompareServicePack failed to compare the Service Pack numbers.

## ISCompareServicePack Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the ISCompareServicePack function.
 *
 * Note: ISCompareServicePack is defined in Sdint.rul.
 *
 \*-----*/
#define SERVICE_PACK "Service Pack 3"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"
```

```

export prototype ExFn_ISCompareServicePack(HWND);

function ExFn_ISCompareServicePack(hMSI)
    BOOL bWinNT;
    NUMBER nvResult;
    STRING svResult;
begin

    // Determine which operating system is running.
    GetSystemInfo (OS, nvResult, svResult);

    if (nvResult = IS_WINDOWSNT) then
        // Running Windows NT.
        bWinNT = TRUE;

        // Check if Service Pack 3 is installed.
        nvResult = ISCompareServicePack (SERVICE_PACK);

        if (nvResult < 0) then
            MessageBox ("Error: ISCompareServicePack failed.", SEVERE);
        elseif (nvResult = LESS_THAN) then
            MessageBox ("No service pack or the service pack is less than "
                + SERVICE_PACK, INFORMATION);
        elseif (nvResult = EQUALS) then
            MessageBox ("The service pack is " + SERVICE_PACK, INFORMATION);
        elseif (nvResult = GREATER_THAN) then
            MessageBox ("The service pack is greater than "
                + SERVICE_PACK, INFORMATION);
        endif;
    else
        MessageBox ("The target system is not running Windows NT.", SEVERE);
    endif;

end;

```

## ISDeterminePlatform

The **ISDeterminePlatform** function sets the system variable [SYSINFO](#), a structured variable whose members are used to specify information about the operating platform of the target computer. ISDeterminePlatform is called directly by the setup engine during setup initialization.

### Parameters

None

### Return Values

None

## IsEmpty

The **IsEmpty** function checks whether a variable of type VARIANT has been initialized.

To check whether a variable of type OBJECT has been assigned a reference to a valid object, call [IsObject](#).

## Syntax

```
IsEmpty ( vVariant );
```

## Parameters

**Table 10 • IsEmpty Parameters**

Parameter	Description
<b>vVariant</b>	Specifies the variable to be checked.

## Return Values

**Table 11 • IsEmpty Return Values**

Return Value	Description
<b>TRUE (1)</b>	vVariant has been initialized.
<b>FALSE (0)</b>	vVariant has not been initialized.

## IsEmpty Example

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the IsEmpty function.
 *
 \*-----*/

function OnBegin()
    VARIANT vVariant;
    BOOL bEmpty;
    STRING szString;
begin
    bEmpty = IsEmpty ( vVariant );
    /* The message "vVariant is empty." is displayed. */
    if bEmpty then
        MessageBox ( "vVariant is empty." , INFORMATION );
    else
        MessageBox ( "vVariant is not empty." , INFORMATION );
    endif;

    /* Initialize vVariant. */
    vVariant = szString;

    bEmpty = IsEmpty ( vVariant );
    /* The message "vVariant is not empty." is displayed. */
    if bEmpty then
        MessageBox ( "vVariant is empty." , INFORMATION );
    else
        MessageBox ( "vVariant is not empty." , INFORMATION );
    endif;
endfunction
```



```

endif;

/* Terminate example setup script. */
abort;
end;

```

## IsObject

The **IsObject** function checks whether a variable of type OBJECT has been assigned a reference to a valid object, using the [CreateObject](#) or [GetObject](#) functions.

### Syntax

```
IsObject ( oObject );
```

### Parameters

**Table 12 • IsObject Parameters**

Parameter	Description
<b>oObject</b>	Specifies the variable to be checked.

### Return Values

**Table 13 • IsObject Return Values**

Return Value	Description
<b>TRUE</b>	oObject has been assigned a reference to a valid object.
<b>FALSE</b>	oObject has not been assigned a reference to a valid object.

## LaunchApp

The **LaunchApp** function enables you to launch another application from within the script. **LaunchApp** calls `LaunchAppAndWait( szCommand, szCmdLine, LAAW_OPTION_NOWAIT )`. For information on **LaunchApp** parameters and return values, see [LaunchAppAndWait](#).

### LaunchApp Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*

```

```

* InstallShield Example Script
*
* Demonstrates the LaunchApp function.
*
* LaunchApp is called to execute an application.
*
* Note: Before running this script, set the preprocessor
*       constants so that they reference the fully qualified
*       names of the Windows Notepad executable and a valid
*       text file on the target system.
*
/*-----*/

#define APPLICATION WINDIR^"Notepad.exe"
#define CMD_LINE    WINDIR^"Readme.txt"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_LaunchApp(HWND);

function ExFn_LaunchApp(hMSI)
begin

    // Launch the Windows Notepad application to edit
    // the Windows Readme.txt file.
    if (LaunchApp (APPLICATION, CMD_LINE) < 0) then
        MessageBox ("Unable to launch "+APPLICATION+".", SEVERE);
    endif;

end;

```

## LaunchAppAndWait

The **LaunchAppAndWait** function enables you to launch another application from within the script. **LaunchAppAndWait** calls the following:

```
LaunchApplication( szProgram, szCmdLine, "", LAAW_STARTUPINFO.wShowWindow, LAAW_PARAMETERS.nTimeOut,
nOptions | LAAW_OPTION_CHANGEDIRECTORY | LAAW_OPTION_FIXUP_PROGRAM );
```

For more information on parameters and return values for **LaunchAppAndWait**, see [LaunchApplication](#).

### Syntax

```
LaunchAppAndWait ( szProgram, szCmdLine, nOptions );
```

## LaunchAppAndWait Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script

```

```

*
* Demonstrates the LaunchAppAndWait function.
*
* This script presents the user with three options:
*
* -- Launch Notepad; after it closes, continue setup
* -- Launch Notepad and continue setup immediately
* -- Exit installation
*
* If the user selects the first option, the installation
* launches Notepad and then waits for it to close before
* continuing. If the user selects the second option, the
* installation launches Notepad and then continues immediately
* to execute the script. If the user selects the third option,
* the installation exits.
*
\*-----*/

#define PROGRAM          WINDIR^"Notepad.EXE"
#define LAUNCH_WAIT_TEXT "Launch Notepad; after it closes, continue setup"
#define LAUNCH_GO_TEXT   "Launch Notepad and continue setup immediately"
#define EXIT_TEXT        "Exit installation"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_LaunchAppAndWait(HWND);

function ExFn_LaunchAppAndWait(hMSI)
    STRING szProgram, szCmdLine, szMsg;
    BOOL bLaunchAndGo, bLaunchAndWait, bExit;
    NUMBER nWait;
begin

    // Run the installation in a normal window;
    Enable (BACKGROUND);
    Enable (DEFWINDOWMODE);

    // Disable the Back button in installation dialogs.
    Disable (BACKBUTTON);

    // Get an option from the user.
    AskOptions (EXCLUSIVE, "Test",
                LAUNCH_WAIT_TEXT, bLaunchAndWait,
                LAUNCH_GO_TEXT, bLaunchAndGo,
                EXIT_TEXT, bExit);

    if !bExit then

        // Set variable to pass to LaunchAppAndWait
        // to indicate whether or not to wait.
        if bLaunchAndWait then
            nWait = WAIT;
        else
            nWait = NOWAIT;
        endif;

        // Launch Notepad; the value of nWait determines
        // when execution of the installation continues.

```

```
if (LaunchAppAndWait (PROGRAM, "", nWait) < 0) then
    MessageBox ("Unable to launch "+ PROGRAM +".",SEVERE);
endif;

MessageBox ("Setup will now exit.", INFORMATION);

endif;

end;
```

## LaunchAppAndWaitInitStartupInfo

The **LaunchAppAndWaitInitStartupInfo** function initializes the [LAAW\\_STARTUPINFO](#) and [LAAW\\_PARAMETERS](#) system variables to the appropriate default values. This function is called automatically during installation initialization.

### Syntax

```
LaunchAppAndWaitInitStartupInfo ( );
```

## System Variables

**Table 14** • LaunchAppAndWaitInitStartupInfo System Variables

System Variable	Default Value
<b>LAAW_STARTUPINFO.cb</b>	SizeOf(LAAW_STARTUPINFO)
<b>LAAW_STARTUPINFO.lpReserved</b>	NULL
<b>LAAW_STARTUPINFO.lpDesktop</b>	NULL
<b>LAAW_STARTUPINFO.lpTitle</b>	NULL
<b>LAAW_STARTUPINFO.wShowWindow</b>	SW_SHOWDEFAULT
<b>LAAW_STARTUPINFO.lpReserved2</b>	NULL
<b>LAAW_STARTUPINFO.cbReserved2</b>	0
<b>LAAW_STARTUPINFO.dwFlags</b>	STARTF_USESHOWWINDOW
<b>LAAW_PARAMETERS.szStatusText</b>	""
<b>LAAW_PARAMETERS.szCommandLineResult</b>	""
<b>LAAW_PARAMETERS.lpProcessAttributes</b>	NULL
<b>LAAW_PARAMETERS.lpThreadAttributes</b>	NULL
<b>LAAW_PARAMETERS.bInheritHandles</b>	FALSE
<b>LAAW_PARAMETERS.dwCreationFlags</b>	NORMAL_PRIORITY_CLASS
<b>LAAW_PARAMETERS.lpEnvironment</b>	NULL
<b>LAAW_PARAMETERS.lpCurrentDirectory</b>	NULL
<b>LAAW_PARAMETERS.nCallbackInterval</b>	1000
<b>LAAW_PARAMETERS.nLaunchResult</b>	0

## Parameters

None.

## Return Values

**Table 15** • LaunchAppAndWaitInitStartupInfo Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	This function always returns ISERR_SUCCESS.

# LaunchApplication

The **LaunchApplication** function launches and optionally waits for the specified application.

**LaunchApplication** uses either the Windows API function [CreateProcess](#) or the Windows API function [ShellExecuteEx](#) (if `nOptions` includes `LAAW_OPTION_USE_SHELLEXECUTE`) to launch the specified application. After the application is launched, if the `LAAW_OPTION_WAIT` or `LAAW_OPTION_WAIT_INCL_CHILD` options are specified, the installation calls [WaitForApplication](#) to wait for the application to terminate; once the process has completed or the specified timeout value has elapsed, the installation continues.

## Syntax

```
LaunchApplication( byval string szProgram, byval string szCmdLine, byval string szDirectory, byval
    number nShowWindow, byval number nTimeOut, byval number nOptions );
```

## Parameters

**Table 16 •** LaunchApplication Parameters

Parameter	Description
<b>szProgram</b>	<p>Specifies the complete path and file name of the application to be launched.</p> <p>If <code>nOptions</code> does not include <code>LAAW_OPTION_USE_SHELLEXECUTE</code>, you can instead specify the file name of the application to be launched in the <code>szCmdLine</code> parameter; if you do so, pass an empty string ("" ) in the <code>szProgram</code> parameter.</p> <p>The <code>szProgram</code> parameter supports URLs only if the <code>LAAW_OPTION_USE_SHELLEXECUTE</code> option is used.</p>
<b>szCmdLine</b>	<p>Specifies the command-line parameters to pass to the launched application. To launch the application with no command-line parameters, pass a null string ("" ).</p> <p>If <code>nOptions</code> does not include <code>LAAW_OPTION_USE_SHELLEXECUTE</code>, you can also specify the file name of the application to be launched in the <code>szCmdLine</code> parameter; if you do so, be sure to do the following:</p> <ul style="list-style-type: none"> <li>• Separate the path to the application and the command line with a single space.</li> <li>• If the fully qualified name of the application includes long folder names and/or a long file name, pass it to <a href="#">LongPathToQuote</a> before adding it to <code>szCmdLine</code>. For example:</li> </ul> <pre>szApplicationPath = WINDIR ^ "Notepad.exe"; szApplicationCmdLine = SRCDIR ^ "Readme.txt"; LongPathToQuote( szApplicationPath, TRUE ); szCmdLine = szApplicationPath + " " + szApplicationCmdLine;</pre>
<b>szDirectory</b>	<p>Specifies the working directory for the application. Note that if an empty string ("" ) is specified (and <code>LAAW_OPTION_USE_SHELLEXECUTE</code> is not specified), the function passes <code>NULL</code> to <b>CreateProcess</b> instead of an empty string.</p>

**Table 16 •** LaunchApplication Parameters (cont.)




Parameter	Description
<b>nShowWindow</b>	<p>Specifies the initial window state of the application as passed to <b>CreateProcess</b> in the wShowWindow parameter of the STARTUPINFO structure, or passed to <b>ShellExecuteEx</b> in the nShow parameter of the SHELLEXECUTEINFO structure (if LAAW_OPTION_USE_SHELLEXECUTE is specified). You can specify any documented value for the aforementioned functions; some values are predefined in <b>ISRTWindows.h</b>:</p> <ul style="list-style-type: none"> <li>• SW_FORCEMINIMIZE</li> <li>• SW_HIDE</li> <li>• SW_MAX</li> <li>• SW_MAXIMIZE</li> <li>• SW_MINIMIZE</li> <li>• SW_NORMAL</li> <li>• SW_RESTORE</li> <li>• SW_SHOW</li> <li>• SW_SHOWDEFAULT</li> <li>• SW_SHOWMAXIMIZED</li> <li>• SW_SHOWMINIMIZED</li> <li>• SW_SHOWMINNOACTIVE</li> <li>• SW_SHOWNA</li> <li>• SW_SHOWNOACTIVATE</li> <li>• SW_SHOWNORMAL</li> </ul>
<b>nTimeOut</b>	<p>Specifies the nTimeOut value (in milliseconds) passed to the <b>WaitForApplication</b> function. Note that if you do not specify LAAW_OPTION_WAIT, the specified value is ignored.</p>



**Table 16 •** LaunchApplication Parameters (cont.)

Parameter	Description
<b>nOptions</b>	<p>Specifies various options, including whether the installation should wait for the launched application to terminate before continuing.</p> <p>Pass one of the following predefined constants in this parameter. You can combine these constants by using the bitwise OR operator ( ), with the following exceptions: you cannot combine LAAW_OPTION_NOWAIT with LAAW_OPTION_WAIT, and you cannot combine LAAW_OPTION_HIDDEN, LAAW_OPTION_MINIMIZED, and LAAW_OPTION_MAXIMIZED.</p> <ul style="list-style-type: none"> <li>• <b>LAAW_OPTION_USE_SHELLEXECUTE</b>—Indicates that the function should call the Windows API function <b>ShellExecuteEx</b> instead of calling <b>CreateProcess</b> to launch the application.</li> <li>• <b>LAAW_OPTION_NOWAIT</b>—Passed through the function to <b>WaitForApplication</b>. Note that using this parameter is equivalent to calling the function <a href="#">LaunchApp</a>.</li> <li>• <b>LAAW_OPTION_WAIT</b>—Passed through the function to <b>WaitForApplication</b>.</li> <li>• <b>LAAW_OPTION_USE_CALLBACK</b>—Passed through the function to <b>WaitForApplication</b>.</li> <li>• <b>LAAW_OPTION_SET_BATCH_INSTALL</b>—Specifies that the function internally sets BATCH_INSTALL to a non-zero value if it detects that the launched application causes a change that requires a reboot, such as updating the RunOnce key. The function attempts to determine this by checking the state of the system before launching the specified application and then comparing the current system state after launching the application (and waiting for the application to complete if LAAW_OPTION_WAIT is specified.) This flag can be used when launching third-party installations that do not return status information indicating that a reboot is needed.</li> </ul> <p>LAAW_OPTION_SET_BATCH_INSTALL is available for InstallScript event-driven code in InstallScript and InstallScript MSI projects. It does not have any effect in InstallScript custom actions.</p> <ul style="list-style-type: none"> <li>• <b>LAAW_OPTION_SHOW_HOURLASS</b>—Specifies that the cursor changes to an hourglass while the launched application is running.</li> <li>• <b>LAAW_OPTION_WAIT_INCL_CHILD</b>—Passed through the function to <b>WaitForApplication</b>.</li> </ul>


**Table 16 •** LaunchApplication Parameters (cont.)

Parameter	Description
<b>nOptions (cont.)</b>	 <p><b>Note •</b> When <code>LAAW_OPTION_WAIT_INCL_CHILD</code> is used, the function detects and waits for only direct child processes of the launched process—not for any additional child processes that are launched by child processes of the initially launched process.</p> <p>For details about using <code>LAAW_OPTION_USE_SHELLEXECUTE</code> with <code>LAAW_OPTION_WAIT_INCL_CHILD</code>, see the <a href="#">Additional Information</a>.</p> <p>This parameter also supports the following options, but they are no longer recommended:</p> <ul style="list-style-type: none"> <li>• <b>LAAW_OPTION_HIDDEN</b>—Specifies that the launched application's main window is initially hidden.</li> <li>• <b>LAAW_OPTION_MINIMIZED</b>—Specifies that the launched application's main window is initially minimized.</li> <li>• <b>LAAW_OPTION_MAXIMIZED</b>—Specifies that the launched application's main window is initially maximized.</li> <li>• <b>LAAW_OPTION_NO_CHANGEDIRECTORY</b>—This is obsolete. This parameter is ignored.</li> <li>• <b>LAAW_OPTION_CHANGEDIRECTORY</b>—Specifies that <b>LaunchApplication</b> should temporarily update the current directory of the installation to that of the application about to be launched. By default, <b>LaunchApplication</b> does not modify the current directory of the installation. Note that <b>LaunchApplication</b> resets the current directory of the installation back to the original value before <b>LaunchApplication</b> returns.</li> </ul>  <p><b>Note •</b> Calling <b>LaunchAppAndWait</b> instead of <b>LaunchApplication</b> automatically includes this option.</p> <ul style="list-style-type: none"> <li>• <b>LAAW_OPTION_FIXUP_PROGRAM</b> —Specifies that <b>LaunchApplication</b> should call <a href="#">LongPathFromShortPath</a> on <code>szProgram</code> in order to ensure that the call to <b>CreateProcess</b> or <b>ShellExecuteEx</b> works correctly. In most cases, this should not be needed.</li> </ul>  <p><b>Note •</b> Calling <b>LaunchAppAndWait</b> instead of <b>LaunchApplication</b> automatically includes this option.</p>

## Return Values

**Table 17** • LaunchApplication Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	Indicates that the application was launched successfully.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the application was not launched successfully.



**Note** • If the application cannot be launched, the `LAAW_PARAMETERS` system variable's `nLaunchResult` member contains the result of calling the Windows API function **GetLastError** after the **CreateProcess** or **ShellExecuteEx** call. If the function is successful and the `LAAW_OPTION_WAIT` option was specified, the `LAAW_PARAMETERS` system variable's `nLaunchResult` member contains the return code of the launched application.

## Additional Information

- If you are not using `LAAW_OPTION_USE_SHELLEXECUTE`, update the `LAAW_STARTUPINFO` system variable before calling **LaunchApplication** to customize the behavior of the application.

If you are using `LAAW_OPTION_USE_SHELLEXECUTE`, customize the `LAAW_SHELLEXECUTEINFO` structure to customize the behavior of the application.

If you are using `LAAW_OPTION_USE_SHELLEXECUTE` on systems running Windows Vista or later and you want to launch the application using the full administrator account (similar to right-clicking the executable file to be run and clicking Run as Administrator), set `LAAW_SHELLEXECUTEVERB` to **runas** before using **LaunchApplication** in your script:

```
LAAW_SHELLEXECUTEVERB = "runas";
```

This ensures that the application is always run with full administrator privileges regardless of whether the application to be launched has an application manifest with relevant settings. Note that this may trigger a User Account Control (UAC) prompt for consent or credentials.

On systems running operating systems earlier than Windows Vista, if **runas** is used, a Run As dialog box is displayed. The behavior is similar to right-clicking the executable file to be run and clicking Run As. This dialog box enables the end user to select the user account that should be used to run the application.

- To obtain identification information about the launched process, use the `LAAW_PROCESS_INFORMATION` system variable or the `LAAW_SHELLEXECUTEINFO` system variable (if you are using `LAAW_OPTION_USE_SHELLEXECUTE`).
- Note that if you are using `LAAW_OPTION_USE_SHELLEXECUTE`, the `LAAW_PROCESS_INFORMATION` is not used, except for the `hProcess` member. The `hProcess` member is updated to the process handle of the launched process (from the `LAAW_SHELLEXECUTEINFO.hProcess` member).
- **ShellExecuteEx** does not return the process ID of the launched process. Thus, if `LAAW_OPTION_USE_SHELLEXECUTE` is used, the `LAAW_OPTION_WAIT_INCL_CHILD` option works only if the Windows API **GetProcessId** is available so that the function can determine the process ID. According to the

Windows API documentation, **GetProcessId** is available on Windows XP SP1 and later and Windows Server 2003 and later. If this API is not available, **LAOW\_OPTION\_WAIT\_INCL\_CHILD** behaves as **LAOW\_OPTION\_WAIT**.

- When the installation is run from any removable media, such as a CD or a DVD, the **Setup.exe** file on Disk1 may not be available during the entire installation. (If **Setup.exe** becomes unavailable while it is running, the operating system sometimes displays a prompt to request that the end user insert the correct disk, and this may cause the installation to fail.) Therefore, to avoid this problem, the **Setup.exe** file is copied to a Temp folder, and the installation is relaunched from there. The original **Setup.exe** then terminates. However, when this happens, **LaunchApplication** behaves as if the installation has completed, and it does not wait.

To avoid this issue, you may want to use the `/clone_wait` parameter when you are launching the child installation; when this occurs, the launched installation keeps the original launched process running, and the parent installation then waits. Note, however, that this may cause problems if the original CD containing **Setup.exe** is not available throughout the entire installation. This includes multiple-CD installations, where the first CD is not available during some parts of the installation.

The only other way to avoid this problem is to add code that determines the ID of the child processes of the launched process and wait for the child process to complete.

## LaunchApplicationInit

The **LaunchApplicationInit** function initializes the **LAOW\_STARTUPINFO** and **LAOW\_PARAMETERS** system variables to the appropriate default values. This function is called automatically during installation initialization. This function supersedes the **LaunchAppAndWaitInitStartupInfo** function.

### Syntax

```
LaunchApplicationInit( );
```

## System Variables

**Table 18** • LaunchApplicationInit System Variables

System Variable	Default Value
<b>LAAW_PARAMETERS.bCallbackEndedWait</b>	FALSE
<b>LAAW_PARAMETERS.bInheritHandles</b>	FALSE
<b>LAAW_PARAMETERS.dwCreationFlags</b>	NORMAL_PRIORITY_CLASS
<b>LAAW_PARAMETERS.lpCurrentDirectory</b>	NULL
<b>LAAW_PARAMETERS.lpEnvironment</b>	NULL
<b>LAAW_PARAMETERS.lpProcessAttributes</b>	NULL
<b>LAAW_PARAMETERS.lpThreadAttributes</b>	NULL
<b>LAAW_PARAMETERS.nCallbackInterval</b>	1000
<b>LAAW_PARAMETERS.nLaunchResult</b>	0
<b>LAAW_PARAMETERS.nTimeOut</b>	INFINITE
<b>LAAW_PARAMETERS.nTimeOutCheckInterval</b>	1000
<b>LAAW_PARAMETERS.nWaitForInputIdleMax</b>	2000
<b>LAAW_PARAMETERS.nWaitResult</b>	WAIT_OBJECT_0
<b>LAAW_PARAMETERS.szCommandLineResult</b>	""
<b>LAAW_PARAMETERS.szStatusText</b>	""
<b>LAAW_SHELLEXECUTEINFO.cbSize</b>	SizeOf(LAAW_SHELLEXECUTEINFO)
<b>LAAW_SHELLEXECUTEINFO.dwHotKey</b>	0
<b>LAAW_SHELLEXECUTEINFO.fMask</b>	SEE_MASK_NOCLOSEPROCESS   SEE_MASK_FLAG_NO_UI
<b>LAAW_SHELLEXECUTEINFO.hIconMonitor</b>	NULL
<b>LAAW_SHELLEXECUTEINFO.hInstApp</b>	NULL
<b>LAAW_SHELLEXECUTEINFO.hkeyClass</b>	NULL
<b>LAAW_SHELLEXECUTEINFO.hProcess</b>	NULL

**Table 18** • LaunchApplicationInit System Variables (cont.)

System Variable	Default Value
<b>LAAW_SHELLEXECUTEINFO.hwnd</b>	GetWindowHandle( HWND_INSTALL )
<b>LAAW_SHELLEXECUTEINFO.lpClass</b>	NULL
<b>LAAW_SHELLEXECUTEINFO.lpDirectory</b>	NULL
<b>LAAW_SHELLEXECUTEINFO.lpFile</b>	NULL
<b>LAAW_SHELLEXECUTEINFO.lpIDList</b>	NULL
<b>LAAW_SHELLEXECUTEINFO.lpParameters</b>	NULL
<b>LAAW_SHELLEXECUTEINFO.lpVerb</b>	&LAAW_SHELLEXECUTEVERB
<b>LAAW_SHELLEXECUTEINFO.nShow</b>	SW_SHOWDEFAULT
<b>LAAW_SHELLEXECUTEVERB</b>	"open"
<b>LAAW_STARTUPINFO.cb</b>	SizeOf(LAAW_STARTUPINFO)
<b>LAAW_STARTUPINFO.cbReserved2</b>	0
<b>LAAW_STARTUPINFO.dwFlags</b>	STARTF_USESHOWWINDOW
<b>LAAW_STARTUPINFO.lpDesktop</b>	NULL
<b>LAAW_STARTUPINFO.lpReserved</b>	NULL
<b>LAAW_STARTUPINFO.lpReserved2</b>	NULL
<b>LAAW_STARTUPINFO.lpTitle</b>	NULL
<b>LAAW_STARTUPINFO.wShowWindow</b>	SW_SHOWDEFAULT

## Parameters

None.

## Return Values

**Table 19** • LaunchApplicationInit Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	This function always returns ISERR_SUCCESS.

## ListAddItem

The **ListAddItem** function adds a numeric element to a number list before or after the current element.



**Note** • *ListAddItem works only with numbered lists.*



### Task

#### To traverse a list:

1. Call [ListGetFirstItem](#) to get the first element in the list.
2. Call [ListGetNextItem](#) repeatedly until you reach the end of the list.

To make a specific element in the list the current element, call [ListSetIndex](#).

## Syntax

```
ListAddItem ( listID, nItem, nPlacementFlag );
```

## Parameters

**Table 20 • ListAddItem Parameters**

Parameter	Description
<b>listID</b>	Specifies the name of a number list. The list identified by listID must already have been initialized by a call to <a href="#">ListCreate</a> .
<b>nItem</b>	Specifies the numeric element to add to the list.
<b>nPlacementFlag</b>	Specifies where to put nItem relative to the current element. The new element will go either before or after the current element. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>AFTER</b>—Adds the new element after the current element in the list.</li> <li>• <b>BEFORE</b>—Adds the new element before the current element in the list.</li> </ul>

## Return Values

**Table 21 • ListAddItem Return Values**

Return Value	Description
<b>&gt; = ISERR_SUCCESS (0)</b>	The function was successful.
<b>&lt; ISERR_SUCCESS (0)</b>	The function was not successful.
<b>ISERR_LIST_NOSUCHLIST (-501)</b>	Indicates that a list with the specified ID does not exist. Valid list IDs are return values from the <a href="#">ListCreate</a> function.

## ListAddItem Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ListAddItem function.
*
* First, an empty list is created by a call to ListCreate.
* Then ListAddItem is called three times to add the numbers
* 1, 3, and 2 to the list. Although the numbers are not added
* in ascending order, they are arranged in that order in the
* list by means of the placement flag that is passed in the
* third parameter of each call to ListAddItem. After the list
* has been built, it is displayed. Finally the current element
* is displayed.
```



```

*
\*-----*/

#define TITLE "ListAddItem Example"
#define MSSG  "The following is a list of the items added:\n\n"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ListAddItem(HWND);

function ExFn_ListAddItem(hMSI)
    NUMBER nvItem, nvResult;
    LIST listID;
    STRING szMsg, svItem;
begin

    // Create an empty number list.
    listID = ListCreate (NUMBERLIST);

    //If an error occurred, report it; then terminate.
    if (listID = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Add the number 1 to the list.
    if (ListAddItem (listID, 1, AFTER) < 0) then
        MessageBox ("First call to ListAddItem failed.", INFORMATION);
        abort;
    endif;

    // The integer 1, in position 1, is the current element.
    // Add the number 3 after the current element.
    if (ListAddItem (listID, 3, AFTER) < 0) then
        MessageBox ("Second call to ListAddItem failed.", INFORMATION);
        abort;
    endif;

    // The integer 3, in position 2, is the current element.
    // Add the number 2 to the list before the current element.
    if (ListAddItem (listID, 2, BEFORE) < 0) then
        MessageBox ("Third call to ListAddItem failed.", INFORMATION);
        abort;
    endif;

    // Retrieve and display the current element.
    ListCurrentItem (listID, nvItem);

    sprintfBox (INFORMATION, TITLE, "Current Item: %d", nvItem);

    // Start building the message to report the numbers.
    szMsg = MSSG;

    // Get the numbers and add them to the message.
    nvResult = ListGetFirstItem (listID, nvItem);
    while (nvResult != END_OF_LIST)
        NumToStr (svItem, nvItem);
        szMsg = szMsg + svItem + " ";

```

```

        nvResult = ListGetNextItem (listID, nvItem);
    endwhile;

    // Display the numbers.
    MessageBox (szMsg, INFORMATION);

    // Remove the list from memory.
    ListDestroy (listID);

end;

```

## ListAddList

The **ListAddList** function adds elements from one list (listAdd) to the destination list (listDest). This function is available for both string lists and number lists; however, the types of listDest and listAdd must match.

### Syntax

```
ListAddList (listDest, listAdd, nPlacementFlag);
```

## Parameters

**Table 22** • ListAddList Parameters

Parameter	Description
<b>listDest</b>	Specify the destination list. <b>ListAddList</b> adds new elements either before or after the current element of this list.
<b>listAdd</b>	Specify the list that contains the elements to be added to the destination list. <b>ListAddList</b> adds all elements (from the current element in this list to the last element in this list) to ListDest.
<b>nPlacementFlag</b>	Specify whether <b>ListAddList</b> should add the list elements before or after the current element of listDest. Available values are: <ul style="list-style-type: none"> <li>• BEFORE</li> <li>• AFTER</li> </ul>

## Return Values

**Table 23** • ListAddList Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	Indicates that the function was successful.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was not successful.

# ListAddString

The **ListAddString** function adds a string to a string list before or after the current element.



**Note** • *ListAddString works only with string lists.*



### Task

#### To traverse a list:

1. Call [ListGetFirstItem](#) to get the first element in the list.
2. Call [ListGetNextItem](#) repeatedly until you reach the end of the list.

To make a specific element in the list the current element, call [ListSetIndex](#).

## Syntax

```
ListAddString ( listID, szString, nPlacementFlag );
```

## Parameters

**Table 24 • ListAddString Parameters**

Parameter	Description
<b>listID</b>	Specifies the name of a string list. The list identified by listID must already have been initialized by a call to <a href="#">ListCreate</a> .
<b>szString</b>	Specifies the string to add to the list.
<b>nPlacementFlag</b>	Specifies where to put szString relative to the current element. The new string will go either before or after the current element. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>AFTER</b>—Adds the new string after the current element in the list.</li> <li>• <b>BEFORE</b>—Adds the new string before the current element in the list.</li> </ul>

## Return Values

**Table 25 • ListAddString Return Values**

Return Value	Description
<b>&gt; = ISERR_SUCCESS (0)</b>	The function was successful.
<b>&lt; ISERR_SUCCESS (0)</b>	The function was not successful.
<b>ISERR_LIST_NOSUCHLIST (-501)</b>	Indicates that a list with the specified ID does not exist. Valid list IDs are return values from the <a href="#">ListCreate</a> function.

## ListAddString Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ListAddString function.
*
* ListAddString is called to add a string to the string list.
* Then it is called again to add another string after the
* first. This string is also set as the current element.
* ListAddString is called for the third time to add a string
* before the current element.
*
*-----*/
```

```

#define TITLE_TEXT "ListAddString Example"
#define MSG_TEXT   "Hardware devices:"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ListAddString(HWND);

function ExFn_ListAddString(hMSI)
    STRING szString, svString;
    LIST listID;
begin

    // Create an empty string list.
    listID = ListCreate (STRINGLIST);

    // If an error occurred, report it; then terminate.
    if (listID = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Add a string to the list. This string becomes the current element.
    szString = "Keyboard";
    if (ListAddString (listID, szString, AFTER) < 0) then
        MessageBox ("ListAddString failed.", INFORMATION);
    endif;

    // Add a second string; insert it after the current element.
    // This string then becomes the current element.
    szString = "Mouse";
    if (ListAddString (listID, szString, AFTER) < 0) then
        MessageBox ("ListAddString failed.", INFORMATION);
    endif;

    // Add a third string; insert it before the current element.
    szString = "Monitor";

    if (ListAddString (listID, szString, BEFORE) < 0) then
        MessageBox ("ListAddString failed.", INFORMATION);
    endif;

    // Show the list of strings.
    SdShowInfoList (TITLE_TEXT, MSG_TEXT, listID);

    // Retrieve and display the current element.
    ListCurrentString (listID, svString);

    MessageBox (svString, INFORMATION);

    // Remove the list from memory.
    ListDestroy (listID);

end;

```

# ListAppendFromArray



**Project** • This information applies to InstallScript projects.

The **ListAppendFromArray** function appends the elements in the array that is specified by `varSource` to the list that is specified by `listResult`.

## Syntax

```
ListAppendFromArray ( listResult, varSource, bString );
```

## Parameters

**Table 26** • ListAppendFromArray Parameters

Parameter	Description
<b>listResult</b>	Specifies the name of a list. The list identified by <code>listResult</code> must already have been initialized by a call to <a href="#">ListCreate</a> .
<b>varSource</b>	Specifies the array to append to the list.
<b>bString</b>	Set this parameter to TRUE if <code>varSource</code> is a string array and <code>listResult</code> is a string list; set this parameter to FALSE if <code>varSource</code> is a number array and <code>listResult</code> is a number list.

## Return Values

**Table 27** • ListAppendFromArray Return Values

Return Value	Description
<b>&gt;= 0</b>	The new number of elements in the list.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was unable to append the array to the list.

## Comments

The array and list types must match (that is, both be string or both be numeric) or the function will fail. If necessary, convert the data appropriately by calling [ListConvertNumToStr](#) or [ListConvertStrToNum](#) before calling [ListAppendToArray](#).

# ListAppendToArray



**Project** • This information applies to InstallScript projects.

The **ListAppendToArray** function appends the elements in the list that is specified by listSource to the array that is specified by varResult, resizing the array appropriately to store the new elements.

## Syntax

```
ListAppendToArray ( varResult, listSource, bString );
```

## Parameters

**Table 28** • ListAppendToArray Parameters

Parameter	Description
<b>varResult</b>	Specifies an array.
<b>listSource</b>	Specifies the list to append to the array. The list identified by listSource must already have been initialized by a call to <a href="#">ListCreate</a> .
<b>bString</b>	Set this parameter to TRUE if varResult is a string array and listSource is a string list; set this parameter to FALSE if varResult is a number array and listSource is a number list.

## Return Values

**Table 29** • ListAppendToArray Return Values

Return Value	Description
<b>&gt;= 0</b>	The new number of elements in the array.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was unable to append the list to the array.

## Comments

The array and list types must match (that is, both be string or both be numeric) or the function will fail. If necessary, convert the data appropriately by calling ListConvertNumToStr or ListConvertStrToNum before calling ListAppendToArray.

The current size of the array determines where the end of the array is and thus where the new elements are placed. For example, if the size of the array is 10 but only one element contains a string, the new data is still added as the 11th and later elements. If you use an autosized array the new elements are added to the end of the array.

# ListConvertNumToStr



**Project** • This information applies to InstallScript projects.

The **ListConvertNumToStr** function converts each numeric element in listNumber to its string equivalent and appends it to listString.

## Syntax

```
ListConvertNumToStr (listString, listNumber);
```

## Parameters

**Table 30** • ListConvertNumToStr Parameters

Parameter	Description
<b>listString</b>	Specifies the string list to which to append the converted numbers. The list identified by listString must already have been initialized by a call to <a href="#">ListCreate</a> .
<b>listNumber</b>	Specifies a number list. The list identified by listNumber must already have been initialized by a call to <b>ListCreate</b> .

## Return Values

**Table 31** • ListConvertNumToStr Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the function successfully converted the number list to a string list.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was unable to convert the number list to a string list.

## Comments

The function uses [NumToStr](#) to convert the numeric elements to strings; if the conversion fails for an element in listNumber, a null string ("") is appended to listString for that element.

# ListConvertStrToNum



**Project** • This information applies to InstallScript projects.

The **ListConvertStrToNum** function converts each string element in listString to its numeric equivalent and appends it to listNumber.

## Syntax

```
ListConvertStrToNum (listNumber, listString );
```



## Parameters

**Table 32** • ListConvertStrToNum Parameters

Parameter	Description
<b>listNumber</b>	Specifies the number list to which to append the converted strings. The list identified by listNumber must already have been initialized by a call to <a href="#">ListCreate</a> .
<b>listString</b>	Specifies a string list. The list identified by listString must already have been initialized by a call to <a href="#">ListCreate</a> .

## Return Values

**Table 33** • ListConvertStrToNum Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the function successfully converted the string list to a number list.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was unable to convert the string list to a number list.

## Comments

The function uses [StrToNum](#) to convert the strings to numbers; if the conversion fails for an element in listString, 0 is appended to listNumber for that element.

# ListCount

The **ListCount** function returns the number of elements in a list.



**Note** • This function works with both strings and number lists.

## Syntax

```
ListCount ( listID );
```

## Parameters

**Table 34 •** ListCount Parameters

Parameter	Description
<b>listID</b>	Specifies the name of a string or number list.

## Return Values

**Table 35 •** ListCount Return Values

Return Value	Description
<b>&gt;= 0</b>	The number of items in the list.
<b>ISERR_LIST_NOSUCHLIST (-501)</b>	Indicates that a list with the specified ID does not exist. Valid list IDs are return values from the <a href="#">ListCreate</a> function.
<b>&lt; 0</b>	ListCount was unable to determine the number of elements in the list.

## ListCount Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ListCount function.
*
* The following adds the names of the program folders to a
* string list and then displays the number of strings in
* the list.
*
\*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ListCount(HWND);

function ExFn_ListCount(hMSI)
    STRING svString;
    LIST listID;
    NUMBER nCount;
begin

    // Create a string list.
    listID = ListCreate (STRINGLIST);
```

```

// If an error occurred, report it; then terminate.
if (listID = LIST_NULL) then
    MessageBox ("Unable to create list.", SEVERE);
    abort;
endif;

// Get the names of the program folders into a list.
GetGroupNameList (listID);

// Count the number of program folders in the list.
nCount = ListCount (listID);

// Report error or display the folder count.
if (nCount < 0) then
    MessageBox ("ListCount failed.", SEVERE);
else
    sprintfBox (INFORMATION, "ListCount",
               "There are %i program folders.", nCount);
endif;

// Remove the list from memory.
ListDestroy (listID);

end;

```

## ListCreate

The **ListCreate** function creates an empty string or number list. You can create any number of lists in a script. A list may contain any number of elements. The only constraint is the amount of available free memory.

When calling any of the list functions, you must pass a valid ID of the list returned by this function. Verify this function was successful in creating the list. Otherwise, all the list functions will fail on the invalid list.

When you no longer need the list, you can destroy the list with the [ListDestroy](#) function.

Each list has a pointer that identifies an element as the “current” element of the list. The various list functions reposition the current element of the list.




---

**Note** • A list cannot contain both types of elements—numbers and strings. InstallScript provides separate functions to work with string lists and with number lists. You cannot use the ID of a number list with the string list functions and vice versa. Use list functions that end in “Item” for number lists and use list functions that end in “String” for string lists.

### Syntax

```
ListCreate ( nListType );
```

## Parameters

**Table 36 • ListCreate Parameters**

Parameter	Description
<b>nListType</b>	Specifies the type of list to create. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>NUMBERLIST</b>—Specifies a number list.</li> <li>• <b>STRINGLIST</b>—Specifies a string list.</li> </ul>

## Return Values

**Table 37 • ListCreate Return Values**

Return Value	Description
<b>ListID</b>	The ID of the newly created, empty list. You must use this ID whenever you want to use this list in other InstallScript list functions. You must check this variable and be sure the function did not return LIST_NULL.
<b>LIST_NULL (-1)</b>	Indicates that InstallShield is unable to create a list. This is a seldom seen condition that is the result of a serious memory problem. You might experience difficulties in continuing the setup with such memory problems.

## Additional Information

Before you can pass a valid list ID to any function that requires a list, you must build the list using ListCreate.

## ListCreate Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ListCreate and ListDestroy functions.
*
* ListCreate is called to create a numbered list. ListDestroy
* is called to destroy it.
*
\*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ListCreate(HWND);
```

```

function ExFn_ListCreate(hMSI)
    LIST listID;
    NUMBER nvItem;
begin

    // Create an empty number list.
    listID = ListCreate (NUMBERLIST);

    // If an error occurred, report it; then terminate.
    if (listID = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Add the number 1078 to the list.
    ListAddItem (listID, 1078, AFTER);

    // Add the number 304 to the list
    ListAddItem (listID, 304, AFTER);

    // Retrieve the current item in the list (304).
    ListCurrentItem (listID, nvItem);

    // Display the current item.
    sprintfBox (INFORMATION, "ListCreate",
        "Current item in list: %d", nvItem);

    // Retrieve the first item in the list (1078).
    ListGetFirstItem (listID, nvItem);

    // Display the first item.
    sprintfBox (INFORMATION, "ListCreate",
        "First item in list: %d", nvItem);

    // Remove the list from memory.
    ListDestroy (listID);

end;

```

## ListCurrentItem

The **ListCurrentItem** function retrieves the current element from the number list specified in listID.

You can also use the [ListGetFirstItem](#) and [ListGetNextItem](#) functions to traverse the list and make any element the current element.




---

**Note** • *ListCurrentItem works only with numbered lists.*

### Syntax

```
ListCurrentItem ( listID, nvItem );
```

## Parameters

**Table 38 • ListCurrentItem Parameters**

Parameter	Description
<b>listID</b>	Specifies a number list.
<b>nvItem</b>	Returns the value of the current element in the list.

## Return Values

**Table 39 • ListCurrentItem Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully retrieved the current element in a number list.
<b>&lt; 0</b>	Indicates that the function was unable to retrieve the current element in a number list.
<b>END_OF_LIST (1)</b>	Indicates that the list is empty and therefore does not have a current element.
<b>ISERR_LIST_NOSUCHLIST</b>	Indicates that a list with the specified ID does not exist. Valid list IDs are return values from the <a href="#">ListCreate</a> function.

## ListCurrentItem Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ListCurrentItem function.
*
* This script adds three numbers to a list and then retrieves
* and displays the value of the current list item.
*
\*-----*/

#define TITLE_TEXT "ListCurrentItem Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ListCurrentItem(HWND);
```

```

function ExFn_ListCurrentItem(hMSI)
    STRING szMsg;
    LIST listID;
    NUMBER nItem;
begin

    // Create the number list.
    listID = ListCreate (NUMBERLIST);

    // If an error occurred, report it; then terminate.
    if (listID = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Add the numbers 100, 200, and 300 to the list.
    for nItem = 100 to 300 step 100
        ListAddItem (listID, nItem, AFTER);
    endfor;

    // Get the current element from the number list.
    if (ListCurrentItem (listID, nItem) < 0) then
        MessageBox ("ListCurrentItem failed.", SEVERE);
    else
        // Report the value of the current item.
        szMsg = "Value of current element in list: %d";
        sprintfBox (INFORMATION, TITLE_TEXT, szMsg, nItem);
    endif;

end;

```

## ListCurrentString

The **ListCurrentString** function retrieves the current element from the string list specified in listID.

You can also use the [ListGetFirstString](#) and [ListGetNextString](#) functions to traverse the list and make any element the current element.




---

**Note** • *ListCurrentString works only with string lists.*

### Syntax

```
ListCurrentString ( listID, svString );
```

## Parameters

**Table 40** • ListCurrentString Parameters

Parameter	Description
<b>listID</b>	Specifies a string list.
<b>svString</b>	Returns the value of the current element in the list.

## Return Values

**Table 41** • ListCurrentString Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully retrieved the current element in a string list.
<b>&lt; 0</b>	Indicates that the function was unable to retrieve the current element in a string list.
<b>END_OF_LIST (1)</b>	Indicates that the list is empty and therefore does not have a current element.
<b>ISERR_LIST_NOSUCHLIST (-501)</b>	Indicates that a list with the specified ID does not exist. Valid list IDs are return values from the <a href="#">ListCreate</a> function.

## ListCurrentString Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ListCurrentString function.
*
* ListAddString retrieves the current string element from a
* string list.
*
\*-----*/

#define TITLE_TEXT "ListCurrentString Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_ListCurrentString(HWND);

function ExFn_ListCurrentString(hMSI)
```



```

    STRING szString, svString, szMsg;
    LIST listID;
begin

    // Create the string list.
    listID = ListCreate (STRINGLIST);

    // If an error occurred, report it; then terminate.
    if (listID = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Add three strings to the list.
    ListAddString (listID, "First string", AFTER);
    ListAddString (listID, "Second string", AFTER);
    ListAddString (listID, "Third string", AFTER);

    // Get the current element in the string list.
    if (ListCurrentString (listID, svString) < 0) then
        MessageBox ("ListCurrentString failed.", SEVERE);
    else
        // Report the value of the current item.
        szMsg = "Current element in list: '%s'";
        sprintfBox (INFORMATION, TITLE_TEXT, szMsg, svString);
    endif;

end;

```

## ListDeleteAll

The **ListDeleteAll** function deletes all elements in the specified list. This function can be used for both string and number lists.

### Syntax

```
ListDeleteAll ( list);
```

## Parameters

**Table 42** • ListDeleteAll Parameters

Parameter	Description
<b>list</b>	The list specified to have all elements deleted.

## Return Values

**Table 43** • ListDeleteAll Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	Indicates that the function was successful.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was not successful.

# ListDeleteItem

The **ListDeleteItem** function removes the current element from the number list you specify in listID.

You can also use the [ListGetFirstItem](#) and [ListGetNextItem](#) functions to traverse the list and make any element the current element.



**Note** • *ListDeleteItem works only with numbered lists.*

## Syntax

```
ListDeleteItem ( listID );
```

## Parameters

**Table 44** • ListDeleteItem Parameters

Parameter	Description
<b>listID</b>	Specifies the number list from which to delete the current element.

## Return Values

**Table 45** • ListDeleteItem Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully deleted the current element from a number list.
<b>&lt; 0</b>	Indicates that the list is empty and therefore does not have a current element.
<b>END_OF_LIST (1)</b>	Indicates that the list is empty and therefore does not have a current element.
<b>ISERR_LIST_NOSUCHLIST</b>	Indicates that a list with the specified ID does not exist. Valid list IDs are return values from the <a href="#">ListCreate</a> function.

## ListDeleteItem Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ListDeleteItem function.
*
* This script first creates a list of numbers and prompts the
* user for a number to delete. Then ListFindItem is called to
* make the selected number the current item. Next,
* ListDeleteItem is called to delete the selected number from
* the list. Finally, the numbers remaining in the list are
* displayed.
*
\*-----*/

#define TITLE_TEXT "ListDeleteItem Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ListDeleteItem(HWND);

function ExFn_ListDeleteItem(hMSI)
```

```

STRING    svItem, szMsg;
LIST      listID;
NUMBER    nvItem, nvResult;
begin

    // Create the number list.
    listID = ListCreate (NUMBERLIST);

    // If an error occurred, report it; then terminate.
    if (listID = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Add the numbers 1, 2, 3, and 4 to the list.
    for nvItem = 1 to 4
        ListAddItem (listID, nvItem, AFTER);
    endfor;

    repeat
        // Disable the Back button.
        Disable (BACKBUTTON);

        // Prompt user to select a number to delete.
        if AskText ("The numbers 1, 2, 3, and 4 have been added to a list.\n"+
            "Select one of those numbers to delete.", "", svItem)then
        endif;

        // Convert the value from string data to a number data.
        nvResult = StrToNum (nvItem, svItem);

        if (nvItem < 1) || (nvItem > 4) then
            MessageBox ("You must enter a number between 1 and 4.", WARNING);
        endif;
    until (nvItem > 0) && (nvItem < 5);

    // Make the first list element the current element
    // so ListFindItem will search from the top of the list.
    ListSetIndex (listID, 0);

    // Search for the selected number.
    nvResult = ListFindItem(listID, nvItem);

    // Delete the selected number from the list.
    if (ListDeleteItem (listID) < 0) then
        MessageBox ("Unable to delete selected number.", SEVERE);
    else
        // Start building the message to report the remaining numbers.
        szMsg = "The list now contains the following numbers:\n\n";

        // Get the remaining numbers and add them to the message.
        nvResult = ListGetFirstItem (listID, nvItem);
        while (nvResult != END_OF_LIST)
            NumToStr (svItem, nvItem);
            szMsg = szMsg + svItem + " ";
            nvResult = ListGetNextItem (listID, nvItem);
        endwhile;

        // Display the remaining numbers.

```

```

        MessageBox (szMsg, INFORMATION);
    endif;

    // Remove the list from memory.
    ListDestroy (listID);

end;
```

## ListDeleteString

The **ListDeleteString** function removes the current element from the string list you specify in listID.

You can also use the [ListGetFirstString](#) and [ListGetNextString](#) functions to traverse the list and make any element the current element.




---

**Note** • *ListCurrentString works only with string lists.*

### Syntax

```
ListDeleteString ( listID );
```

## Parameters

**Table 46 • ListDeleteString Parameters**

Parameter	Description
<b>listID</b>	Specifies the string list from which to delete the current element.

## Return Values

**Table 47 • ListDeleteString Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully deleted the current element from a string list.
<b>&lt; 0</b>	Indicates that the function was unable to delete the current element from a string list.
<b>END_OF_LIST (1)</b>	Indicates that the list is empty and therefore does not have a current element.
<b>ISERR_LIST_NOSUCHLIST (-501)</b>	Indicates that a list with the specified ID does not exist. Valid list IDs are return values from the <a href="#">ListCreate</a> function.

## ListDeleteString Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the ListDeleteString function.
 *
 * This script first creates a list of strings and then displays
 * the current string in a dialog. Next, ListDeleteString
 * is called twice, deleting the last two strings in the list.
 * The current string is then displayed again.
 *
 \*-----*/

#define TITLE_TEXT "ListDeleteString & ListCurrentString"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ListDeleteString(HWND);
```

```

function ExFn_ListDeleteString(hMSI)
    STRING    szString, svString, szMsg;
    LIST      listID;
begin

    // Create the string list.
    listID = ListCreate (STRINGLIST);

    // If an error occurred, report it; then terminate.
    if (listID = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Add three strings to the list.
    ListAddString (listID, "First string", AFTER);
    ListAddString (listID, "Second string", AFTER);
    ListAddString (listID, "Third string", AFTER);

    // Display the current string in the list.
    if (ListCurrentString (listID, svString) < 0) then
        MessageBox ("First call to ListCurrentString failed.", SEVERE);
    else
        szMsg  = "Current string in list: %s";
        sprintfBox (INFORMATION, TITLE_TEXT, szMsg, svString);

    endif;

    // Remove the current string ("Value three").
    if (ListDeleteString (listID) < 0) then
        MessageBox ("First call to ListDeleteString failed.", SEVERE);
    endif;

    // Remove the current string ("Value two").
    if (ListDeleteString (listID) < 0) then
        MessageBox ("Second call to ListDeleteString failed.", SEVERE);
    endif;

    // Display the current string in the list.
    if (ListCurrentString(listID, svString) < 0) then
        MessageBox ("Second call to ListCurrentString failed.", SEVERE);
    else
        szMsg  = "Current string in list: %s";
        sprintfBox (INFORMATION, TITLE_TEXT, szMsg, svString);

    endif;

    // Remove the list from memory.
    ListDestroy (listID);

end;

```

# ListDestroy

The **ListDestroy** function destroys the contents of a list and the list itself. Use this function to remove the string or number list identified in listID.

You should destroy all the lists you create when you no longer need them or at the end of the setup script. When you destroy a list, you free all memory associated with the list.



**Note** • This function works with both string and number lists. After you destroy a list, do not use that in any list function.

## Syntax

```
ListDestroy ( listID );
```

## Parameters

**Table 48** • ListDestroy Parameters

Parameter	Description
<b>listID</b>	Specifies the string or number list to destroy.

## Return Values

**Table 49** • ListDestroy Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully destroyed the list, removing it from memory.
<b>&lt; 0</b>	Indicates that the function was unable to destroy the list.
<b>ISERR_LIST_NOSUCHLIST (-501)</b>	Indicates that a list with the specified ID does not exist. Valid list IDs are return values from the <a href="#">ListCreate</a> function.

## ListDestroy Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the ListCreate and ListDestroy functions.
 *
```



```

* ListCreate is called to create a number list. ListDestroy
* is called to destroy it.
*
\*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ListDestroy(HWND);

function ExFn_ListDestroy(hMSI)
    LIST listID;
    NUMBER nvItem;
begin
    // Create an empty number list.
    listID = ListCreate (NUMBERLIST);

    // If an error occurred, report it; then terminate.
    if (listID = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Add the number 1078 to the list.
    ListAddItem (listID, 1078, AFTER);

    // Add the number 304 to the list
    ListAddItem (listID, 304, AFTER);

    // Retrieve the current item in the list (304).
    ListCurrentItem (listID, nvItem);

    // Display the current item.
    sprintfBox (INFORMATION, "ListCreate",
        "Current item in list: %d", nvItem);

    // Retrieve the first item in the list (1078).
    ListGetFirstItem (listID, nvItem);

    // Display the first item.
    sprintfBox (INFORMATION, "ListCreate",
        "First item in list: %d", nvItem);

    // Remove the list from memory.
    ListDestroy (listID);

end;

```

## ListFindItem

The **ListFindItem** function searches for a specific element in a number list, starting at the current element and continuing through the list from that point. If you want to start the search from the beginning of the list, use the [ListGetFirstItem](#) function. When ListFindItem finds the element, it becomes the current element in the list.



**Note** • The `ListFindItem` function works only with number lists.

## Syntax

```
ListFindItem ( listID, nItem );
```

## Parameters

**Table 50** • ListFindItem Parameters

Parameter	Description
<b>listID</b>	Specifies the number list to search.
<b>nItem</b>	Specifies the item to find in the list.

## Return Values

**Table 51** • ListFindItem Return Values

Return Value	Description
<b>0</b>	The function successfully found the requested element.
<b>&lt; 0</b>	An error prevented the function from searching the specified list. This error occurs, for example, if the list specified by listID does not exist.
<b>END_OF_LIST (1)</b>	The function searched to the end of the list and did not find the requested element.
<b>ISERR_LIST_NOSUCHLIST (-501)</b>	A list with the specified ID does not exist. Valid list IDs are return values from the <b>ListCreate</b> function.

## ListFindItem Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the ListFindItem function.
 *
 * This script creates a number list and adds three numbers
 * to it. The user is then asked to guess one of the numbers.
 * ListFindItem is called to search the list for the number the
```

```

* user entered. A message box is displayed to tell the user
* whether the guess was right or wrong.
*
\*-----*/

#define TITLE_TEXT "ListFindItem Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ListFindItem(HWND);

function ExFn_ListFindItem(hMSI)
    STRING svItem;
    NUMBER nItem, nResult;
    LIST listID;
begin

    // Create a number list.
    listID = ListCreate (NUMBERLIST);

    // If an error occurred, report it; then terminate.
    if (listID = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Add three numbers to the list
    ListAddItem (listID, 1, AFTER);
    ListAddItem (listID, 5, AFTER);
    ListAddItem (listID, 9, AFTER);

    // Prompt user for a number.
    if AskText ("Three numbers between 1 and 10 have been added to a list.\n"+
        "Try to guess one of those numbers.", "", svItem) = NEXT then
        // Convert the value from string data to number data.
        if StrToNum (nItem, svItem) < 0 then
            MessageBox ("You did not enter a number: operation canceled.", SEVERE);
        else
            // Make the first list element the current element
            // so ListFindItem searches from the top of the list.
            ListSetIndex (listID, 0);

            // Search for the list of numbers.
            nResult = ListFindItem (listID, nItem);

            // Display the results of the search.
            if (nResult < 0) then
                MessageBox ("ListFindItem failed.", SEVERE);
            elseif (nResult = END_OF_LIST) then
                sprintfBox (WARNING, TITLE_TEXT, "Sorry, %d is not in the list.", nItem);
            elseif (nResult = 0) then
                sprintfBox (INFORMATION, TITLE_TEXT, "Yes, %d is in the list.", nItem);
            endif;
        endif;
    endif;

    // Remove list from memory.
    ListDestroy (listID);

```

```
end;
```

## ListFindKeyValueString

The **ListFindKeyValueString** function searches a string or number list for a specified value. It returns a value from an additional list that corresponds with the position of the found string in the first list. This enables you to search lists of key-value pairs for a particular key and get the corresponding value.

### Syntax

```
ListFindKeyValueString (byval LIST listKeys, byval LIST listValues, byval string szKey, byref string  
    svValue);
```

## Parameters

**Table 52 • ListFindKeyValueString Parameters**

Parameter	Description
<b>listKeys</b>	<p>Indicate the list of keys to be searched. You can specify a string or number list. The list must be initialized by calling <b>ListCreate</b> before calling <b>ListFindKeyValueString</b>.</p> <p>Note that the starting point for the search is always the beginning of the list, regardless of the current position of the list. When the function returns, the current position in the list is the found element.</p>
<b>listValues</b>	<p>Indicate the list of values that corresponds with the list of keys. You can specify a string or number list. The list must be initialized by calling <b>ListCreate</b> before calling <b>ListFindKeyValueString</b>.</p> <p>Note that listKeys and listValues do not necessarily need to be the same type of list.</p>
<b>szKey</b>	<p>Specify the string for which you want to search.</p> <p>If listKeys is a number list, szKey is converted to a number by using the <b>StrToNum</b> function, which is then used to search the list.</p> <p><b>ListFindKeyValueString</b> calls <b>ListFindString</b> (if the list is a string list) or <b>ListFindItem</b> (if the list is a number list) to search the list. Searches for string lists are case-sensitive, since <b>ListFindString</b> performs case-sensitive searches.</p>
<b>svValue</b>	<p>The value from the second list is returned in this parameter. If listValues is a number list, the number is converted to a string using <b>NumToStr</b> and then passed back as a string.</p>

## Return Values

**Table 53 •**

Return Value	Description
<b>0</b>	The function successfully found the requested element.
<b>&lt; 0</b>	An error prevented the function from searching the specified list. This error occurs, for example, if the list specified by listID does not exist.
<b>END_OF_LIST (1)</b>	The function searched to the end of the list and did not find the requested element.
<b>ISERR_LIST_NOSUCHLIST (-501)</b>	A list with the specified ID does not exist. Valid list IDs are return values from the <b>ListCreate</b> function.

# ListFindString

The **ListFindString** function searches for a specified element in a string list, starting at the current element and continuing from that point. If you want to start the search from the beginning of the string list, call the [ListGetFirstString](#) function. When ListFindString finds the string, it becomes the current element in the list.



**Note** • The ListFindString function performs a case-sensitive comparison of the strings, and works only with string lists.

## Syntax

```
ListFindString ( listID, szString );
```

## Parameters

**Table 54** • ListFindString Parameters

Parameter	Description
<b>listID</b>	Specifies the string list to search.
<b>szString</b>	Specifies the string to find in the list. InstallShield performs a case-sensitive comparison when searching for this string.

## Return Values

**Table 55** • ListFindString Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully found the requested element.
<b>&lt; 0</b>	Indicates that an error prevented the function from searching the specified list. This error will occur, for example, if the list specified by listID does not exist.
<b>END_OF_LIST (1)</b>	Indicates that InstallShield searched to the end of the list and did not find the requested element.
<b>ISERR_LIST_NOSUCHLIST (-501)</b>	Indicates that a list with the specified ID does not exist. Valid list IDs are return values from the <a href="#">ListCreate</a> function.

## ListFindString Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the ListFindString function.
 *
 * This script prompts the user to input the name of a program
 * folder on the target system. ListFindString is then called
 * to search for this folder name in a list that contains all
 * folder names on the target system. A message is displayed
 * to report the results of the search.
 *
\*-----*/

#define TITLE_TEXT "ListFindString Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_ListFindString(HWND);

function ExFn_ListFindString(hMSI)
    STRING szString;
    LIST listID, listSubfoldersID;
    NUMBER nResult;
begin

    // Create string lists
    listID = ListCreate (STRINGLIST);
    listSubfoldersID = ListCreate (STRINGLIST);

    // If an error occurred, report it; then terminate.
    if (listID = LIST_NULL) || (listSubfoldersID = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Fill the list with the names of program folders.
    GetFolderNameList (FOLDER_PROGRAMS, listID, listSubfoldersID);

    // Prompt user to enter a folder name.
    AskText ("Please enter the name of an existing program folder:",
        "", szString);

    // Make the first list element the current element
    // so ListFindItem will search from the top of the list.
    ListSetIndex (listSubfoldersID, 0);

    // Search the list for the folder name entered by the user.
    // Reminder: The string comparison is case sensitive.
    nResult = ListFindString (listSubfoldersID, szString);

    // Report the search result.
    if (nResult < 0) then
        MessageBox ("ListFindString failed.", SEVERE);
    elseif (nResult = END_OF_LIST) then
        sprintfBox (WARNING, TITLE_TEXT, "%s could not be found.", szString);
    elseif (nResult = 0) then
        sprintfBox (INFORMATION, TITLE_TEXT, "ListFindString found: %s.",

```

```
        szString);  
    endif;  
  
    // Remove list from memory.  
    ListDestroy (listID);  
    ListDestroy (listSubfoldersID);  
  
end;
```

## ListGetFirstItem

The **ListGetFirstItem** function retrieves the first element from a number list. The first item becomes the current element in the list.



---

**Note** • The *ListGetFirstItem* function works only with number lists.

### Syntax

```
ListGetFirstItem ( listID, nvItem );
```



## Parameters

**Table 56 • ListGetFirstItem Parameters**

Parameter	Description
<b>listID</b>	Specifies the number list from which to retrieve the first element.
<b>nvItem</b>	Returns the first element of the number list.

## Return Values

**Table 57 • ListGetFirstItem Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully retrieved the first element in a number list.
<b>-1</b>	Indicates that an error prevented the function from retrieving the first element in a number list.
<b>END_OF_LIST (1)</b>	Indicates that the list is empty.
<b>ISERR_LIST_NOSUCHLIST (-501)</b>	Indicates that a list with the specified ID does not exist. Valid list IDs are return values from the <a href="#">ListCreate</a> function.

## ListGetFirstItem Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ListGetFirstItem and ListGetNextItem
* functions.
*
* This script starts by creating a number list and adding
* three numbers to it. ListGetFirstItem is called then to get
* the first number from the list. This call also makes the
* first number the current element. A loop process follows to
* display the number retrieved from the list and then call
* ListGetNextItem to get the next number. The loop executes
* until ListGetFirstItem or ListGetNextItem reaches the end
* of the list.
*
```

```

\*-----*/

#define TITLE_TEXT "ListGetFirstItem & ListGetNextItem"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ListGetFirstItem(HWND);

function ExFn_ListGetFirstItem(hMSI)
    STRING szMsg;
    LIST listID;
    NUMBER nvItem, nResult;
begin

    // Create a number list.
    listID = ListCreate (NUMBERLIST);

    // If an error occurred, report it; then terminate.
    if (listID = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Add three numbers to the list.
    ListAddItem (listID, 1024, AFTER);
    ListAddItem (listID, 360, AFTER);
    ListAddItem (listID, 777, AFTER);

    // Get the first number from the list.
    nResult = ListGetFirstItem (listID, nvItem);

    // Loop while not at end of list.
    while (nResult != END_OF_LIST)
        // Display the number retrieved from the list.
        SprintfBox (INFORMATION, TITLE_TEXT, "%i", nvItem);

        // Get the next number from the list.
        nResult = ListGetNextItem (listID, nvItem);
    endwhile;

    // Remove the list from memory.
    ListDestroy (listID);

end;

```

## ListGetFirstString

The **ListGetFirstString** function retrieves the first element from a string list. The first string becomes the current element in the list.

The ListGetFirstString function works only with string lists.

### Syntax

```
ListGetFirstString ( listID, svString );
```

## Parameters

**Table 58 • ListGetFirstString Parameters**

Parameter	Description
<b>listID</b>	Specifies the string list from which to retrieve the first element.
<b>svString</b>	Returns the first element of the string list.

## Return Values

**Table 59 • ListGetFirstString Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully retrieved the first element in a string list.
<b>-1</b>	Indicates that an error prevented the function from retrieving the first element in a string list.
<b>END_OF_LIST (1)</b>	Indicates that the list is empty.
<b>ISERR_LIST_NOSUCHLIST (-501)</b>	Indicates that a list with the specified ID does not exist. Valid list IDs are return values from the <a href="#">ListCreate</a> function.

## ListGetFirstString Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ListGetFirstString and ListGetNextString
* functions.
*
* This script starts by creating a string list and adding
* program folder names to it. ListGetFirstString is called
* then to get the first string from the list. This call also
* makes the first string the current element. A loop process
* follows to display the string retrieved from the list and
* then call ListGetNextString to get the next string. The
* loop executes until ListGetFirstString or ListGetNextString
* reaches the end of the list.
*
\*-----*/
```

```

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ListGetFirstString(HWND);

function ExFn_ListGetFirstString(hMSI)
    STRING svString;
    LIST listID;
    NUMBER nResult;
begin
    // Create a string list.
    listID = ListCreate (STRINGLIST);

    // If an error occurred, report it; then terminate.
    if (listID = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Get the names of the program folders into a list.
    if GetGroupNameList(listID) < 0 then ;
        // Report ListCreate failure.
        MessageBox ("Unable to retrieve program folder names.", SEVERE);
    else
        // Get the first string in the list.
        nResult = ListGetFirstString (listID, svString);

        // Loop while list items continue to be retrieved.
        while (nResult != END_OF_LIST)
            // Display the current element.
            MessageBox (svString, INFORMATION);

            // Get the next string in the list.
            nResult = ListGetNextString (listID, svString);
        endwhile;
    endif;

    // Remove the list from memory.
    ListDestroy (listID);

end;

```

## ListGetIndex



**Project** • This information applies to InstallScript projects.

The **ListGetIndex** function retrieves the index of the specified list's current element. Index numbers starts at zero (0).

### Syntax

```
ListGetIndex ( listID, nIndex );
```

## Parameters

**Table 60** • ListGetIndex Parameters

Parameter	Description
<b>listID</b>	Specifies the ID of the string or number list whose current element's index is to be retrieved.
<b>nIndex</b>	Returns the index of the list's current element.

## Return Values

**Table 61** • ListGetIndex Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the function successfully retrieved the index of the list's current element.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was unable to retrieve the index of the list's current element.
<b>ISERR_LIST_NOSUCHLIST</b>	Indicates that a list with the specified ID does not exist. Valid list IDs are return values from the <a href="#">ListCreate</a> function.

## Additional Information

- Use [ListCurrentItem](#) and [ListCurrentString](#) to retrieve the value of the current element.
- This function works on both string and number lists.

# ListGetNextItem

The **ListGetNextItem** function retrieves the item after the current element in a number list. The retrieved item becomes the current element in the list.



**Note** • The *ListGetNextItem* function works only with number lists.

## Syntax

```
ListGetNextItem ( listID, nvItem );
```

## Parameters

**Table 62 • ListGetNextItem Parameters**

Parameter	Description
<b>listID</b>	Specifies the number list from which to retrieve the next element.
<b>nvItem</b>	Returns the item that follows the current element in the number list. That item becomes the current element in the list.

## Return Values

**Table 63 • ListGetNextItem Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully retrieved the element after the current element in a number list.
<b>-1</b>	Indicates that an error prevented the function from retrieving the specified element in a number list.
<b>END_OF_LIST (1)</b>	Indicates that the current item is the last element in the list.
<b>ISERR_LIST_NOSUCHLIST (-501)</b>	Indicates that a list with the specified ID does not exist. Valid list IDs are return values from the <a href="#">ListCreate</a> function.

## ListGetNextItem Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ListGetFirstItem and ListGetNextItem
* functions.
*
* This script starts by creating a number list and adding
* three numbers to it. ListGetFirstItem is called then to get
* the first number from the list. This call also makes the
* first number the current element. A loop process follows to
```

```

* display the number retrieved from the list and then call
* ListGetNextItem to get the next number. The loop executes
* until ListGetFirstItem or ListGetNextItem reaches the end
* of the list.
*
\*-----*/

#define TITLE_TEXT "ListGetFirstItem & ListGetNextItem"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ListGetNextItem(HWND);

function ExFn_ListGetNextItem(hMSI)
    STRING szMsg;
    LIST listID;
    NUMBER nvItem, nResult;
begin

    // Create a number list.
    listID = ListCreate (NUMBERLIST);

    // If an error occurred, report it; then terminate.
    if (listID = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Add three numbers to the list.
    ListAddItem (listID, 1024, AFTER);
    ListAddItem (listID, 360, AFTER);
    ListAddItem (listID, 777, AFTER);

    // Get the first number from the list.
    nResult = ListGetFirstItem (listID, nvItem);

    // Loop while not at end of list.
    while (nResult != END_OF_LIST)
        // Display the number retrieved from the list.
        sprintfBox (INFORMATION, TITLE_TEXT, "%i", nvItem);

        // Get the next number from the list.
        nResult = ListGetNextItem (listID, nvItem);
    endwhile;

    // Remove the list from memory.
    ListDestroy (listID);

end;

```

## ListGetNextString

The **ListGetNextString** function retrieves the element after the current element in a string list. The retrieved element becomes the current element in the list.



**Note** • The `ListGetNextString` function works only with string lists.

## Syntax

```
ListGetNextString ( listID, svString );
```

## Parameters

**Table 64** • ListGetNextString Parameters

Parameter	Description
<b>listID</b>	Specifies the string list from which to retrieve the next element.
<b>svString</b>	Returns the string that follows the current element of the string list. This string becomes the current element in the list.

## Return Values

**Table 65** • ListGetNextString Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully retrieved the element after the current element in a string list.
<b>-1</b>	Indicates that an error prevented the function from retrieving the specified element in a string list.
<b>END_OF_LIST (1)</b>	Indicates that the current item is the last element in the list.
<b>ISERR_LIST_NOSUCHLIST (-501)</b>	Indicates that a list with the specified ID does not exist. Valid list IDs are return values from the <a href="#">ListCreate</a> function.

## ListGetNextString Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
```



```

* InstallShield Example Script
*
* Demonstrates the ListGetFirstString and ListGetNextString
* functions.
*
* This script starts by creating a string list and adding
* program folder names to it. ListGetFirstString is called
* then to get the first string from the list. This call also
* makes the first string the current element. A loop process
* follows to display the string retrieved from the list and
* then call ListGetNextString to get the next string. The
* loop executes until ListGetFirstString or ListGetNextString
* reaches the end of the list.
*
\*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ListGetNextString(HWND);

function ExFn_ListGetNextString(hMSI)
    STRING svString;
    LIST listID;
    NUMBER nResult;
begin
    // Create a string list.
    listID = ListCreate (STRINGLIST);

    // If an error occurred, report it; then terminate.
    if (listID = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Get the names of the program folders into a list.
    if GetGroupNameList(listID) < 0 then ;
        // Report ListCreate failure.
        MessageBox ("Unable to retrieve program folder names.", SEVERE);
    else
        // Get the first string in the list.
        nResult = ListGetFirstString (listID, svString);

        // Loop while list items continue to be retrieved.
        while (nResult != END_OF_LIST)
            // Display the current element.
            MessageBox (svString, INFORMATION);

            // Get the next string in the list.
            nResult = ListGetNextString (listID, svString);
        endwhile;
    endif;

    // Remove the list from memory.
    ListDestroy (listID);

end;

```

# ListGetType

The **ListGetType** function determines whether a list is a STRINGLIST or NUMBERLIST.

## Syntax

```
ListGetType (listID);
```

## Parameters

**Table 66 • ListGetType Parameters**

Parameter	Description
<b>listID</b>	Specifies the name of a string list. The list identified by listID must already have been initialized by a call to <a href="#">ListCreate</a> .

## Return Values

**Table 67 • ListGetType Return Values**

Return Value	Description
<b>STRINGLIST</b>	Indicates that the list is a string list.
<b>NUMBERLIST</b>	Indicates that the list is a number list.
<b>&lt;0</b>	Indicates that the function was unable to determine the type of list.

## ListGetType Example



**Note •** The list identified by listID must already have been initialized by a call to [ListCreate](#)<LangrefListCreate.htm>.

```
//Sample code
LIST listID;
int nType;

program
    listID = ListCreate(STRINGLIST);
    if (LIST_NULL != listID) then
        nType = ListGetType(listID);
        if (STRINGLIST = nType) then
            MessageBox("It is a STRINGLIST", INFORMATION);
        else
            MessageBox("It is a NUMBERLIST", INFORMATION);
        endif;
    endif;
    ListDestroy(listID);
```

```

listID = ListCreate(NUMBERLIST);
if (LIST_NULL != listID) then
    nType = ListGetType(listID);
    if (STRINGLIST = nType) then
        MessageBox("It is a STRINGLIST", INFORMATION);
    else
        MessageBox("It is a NUMBERLIST", INFORMATION);
    endif;
endif;
ListDestroy(listID);
endprogram

```

## ListReadFromFile

The **ListReadFromFile** function reads a text file into a list. After you load a text file into a list, you can use it for various functions in the setup, such as displaying a README file at the end of the setup or writing a string list to the disk with [ListWriteToFile](#).

This function detects the newline characters at the end of each string and uses the characters as delimiters for each element in the list.




---

**Note** • The *ListReadFromFile* function operates on string lists and text files only.

### Syntax

```
ListReadFromFile ( listID, szFile );
```

## Parameters

**Table 68 • ListReadFromFile Parameters**

Parameter	Description
<b>listID</b>	Returns a list of the lines read from the file specified by szFile. The list identified by listID must already have been initialized by a call to <a href="#">ListCreate</a> .
<b>szFile</b>	Specifies the fully qualified name of the file that will be read into the list.

## Return Values

**Table 69 • ListReadFromFile Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully read the lines of text in a file into a list.
<b>ISERR_LIST_NOSUCHLIST (-501)</b>	Indicates that a list with the specified ID does not exist. Valid list IDs are return values from the <a href="#">ListCreate</a> function.
<b>&lt; 0</b>	Indicates that the function was unable to read the lines from a text file into a list.

## ListReadFromFile Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ListReadFromFile and ListWriteToFile
* functions.
*
* ListReadFromFile is called to read and display the
* Autoexec.bat file. After the list has been displayed,
* a remark is appended to the end of the list and the list
* is written to a new file. The original file remains
* unchanged.
*
* Note: In order for this script to run correctly, there must
* be a batch file named Autoexec.bat in the root
```

```

*      directory of drive C.
*
\*-----*/

#define OLD_FILE "C:\\Autoexec.bat"
#define NEW_FILE "C:\\Autoexec.lst"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ListReadFromFile(HWND);

function ExFn_ListReadFromFile(hMSI)
    LIST listID;
begin

    // Create a string list.
    listID = ListCreate (STRINGLIST);

    // If an error occurred, report it; then terminate.
    if (listID = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Read the file into a string list.
    if (ListReadFromFile (listID, OLD_FILE) < 0) then
        // Report ListReadFromFile failure.
        MessageBox ("Unable to read" + OLD_FILE + ".", SEVERE);
    else
        // Display the list.
        SdShowInfoList ("Demo", "The example list file:", listID);

        // Add a new string to the list.
        ListAddString (listID, "REM Added by IS setup.", AFTER);

        // Write the list to a file.
        if (ListWriteToFile (listID, NEW_FILE) < 0) then
            MessageBox ("Unable to write list to " + NEW_FILE + ".", SEVERE);
        else
            MessageBox ("Successfully wrote list " + NEW_FILE + ".", INFORMATION);
        endif;
    endif;

    // Remove the list from memory.
    ListDestroy (listID);

end;

```

## ListSetCurrentItem

The **ListSetCurrentItem** function assigns the value of `nItem` to the current element in a number list.



**Note** • The *ListSetCurrentItem* function works only with number lists.

## Syntax

```
ListSetCurrentItem ( listID, nItem );
```

## Parameters

**Table 70** • ListSetCurrentItem Parameters

Parameter	Description
<b>listID</b>	Specifies the name of a number list whose current element is to be updated. The list identified by listID must already have been initialized by a call to <a href="#">ListCreate</a> .
<b>nItem</b>	Specifies the numeric value that will replace the current element.

## Return Values

**Table 71** • ListSetCurrentItem Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully updated the current element in a number list.
<b>&lt; 0</b>	Indicates that the function was unable to update the current element in a number list.
<b>END_OF_LIST (1)</b>	Indicates that the list is empty.
<b>ISERR_LIST_NOSUCHLIST (-501)</b>	Indicates that a list with the specified ID does not exist. Valid list IDs are return values from the <a href="#">ListCreate</a> function.

## ListSetCurrentItem Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the ListSetCurrentItem function.
 *
 * This script first creates a number list and then adds two
 * numbers to it. Next, the list is displayed in a dialog.
 * ListSetCurrentItem is then called to update the value of
```

```

* the current element in the list. Finally, this new list is
* displayed.
*
\*-----*/

#define MSG_TEXT  "Elements in listID:\n\n"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ListSetCurrentItem(HWND);

function ExFn_ListSetCurrentItem(hMSI)
    LIST listID;
    NUMBER nItem, nvResult, nvItem;
    STRING szMsg, svItem;
begin

    // Create an empty number list.
    listID = ListCreate (NUMBERLIST);

    // If an error occurred, report it; then terminate.
    if (listID = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Add two numbers to the list.
    ListAddItem (listID, 1024, AFTER);
    ListAddItem (listID, 360, AFTER);

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Start building the message to report the numbers.
    szMsg = MSG_TEXT;

    // Get the numbers and add them to the message.
    nvResult = ListGetFirstItem (listID, nvItem);
    while (nvResult != END_OF_LIST)
        NumToStr (svItem, nvItem);
        szMsg = szMsg + svItem + " ";
        nvResult = ListGetNextItem (listID, nvItem);
    endwhile;

    // Display the numbers.
    MessageBox (szMsg, INFORMATION);

    // Replace the value of the second item with a new value.
    if (ListSetCurrentItem (listID, 777) < 0) then
        MessageBox ("ListSetCurrentItem failed.", SEVERE);
    else
        // Start building the message to report the numbers.
        szMsg = MSG_TEXT;

        // Get the numbers and add them to the message.
        nvResult = ListGetFirstItem (listID, nvItem);

        while (nvResult != END_OF_LIST)

```

```

        NumToStr (svItem, nvItem);
        szMsg = szMsg + svItem + " ";
        nvResult = ListGetNextItem (listID, nvItem);
    endwhile;

    // Display the numbers.
    MessageBox (szMsg, INFORMATION);
endif;

// Remove list from memory.
ListDestroy (listID);

end;

```

## ListSetCurrentString

The **ListSetCurrentString** function assigns the value of szString to the current element in the string list.




---

**Note** • The *ListSetCurrentString* function works only with string lists.

### Syntax

```
ListSetCurrentString ( listID, szString );
```



## Parameters

**Table 72** • ListSetCurrentString Parameters

Parameter	Description
<b>listID</b>	Specifies the name of a string list whose current element is to be updated. The list identified by listID must already have been initialized by a call to <a href="#">ListCreate</a> .
<b>szString</b>	Specifies the string value that will replace the current element.

## Return Values

**Table 73** • ListSetCurrentString Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully updated the current element in a number list.
<b>ISERR_LIST_NOSUCHLIST (-501)</b>	Indicates that a list with the specified ID does not exist. Valid list IDs are return values from the <a href="#">ListCreate</a> function.
<b>&lt; 0</b>	Indicates that the function was unable to update the current element in a number list. A common reason for this error is that the index is out of range of available list elements.

## ListSetCurrentString Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ListSetCurrentString function.
*
* In this script, two strings are added to an empty list.
* The list is then displayed in a dialog. Next,
* ListSetCurrentString is called to replace the current
* element. The list is then displayed again in a
* dialog.
*
\*-----*/
```

```

#define TITLE_TEXT "ListSetCurrentString Example"
#define MSG_TEXT   "Elements in listID:"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ListSetCurrentString(HWND);

function ExFn_ListSetCurrentString(hMSI)
    LIST listID;
begin
    // Create an empty string list.
    listID = ListCreate (STRINGLIST);

    // If an error occurred, report it; then terminate.
    if (listID = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Add two strings to the list.
    ListAddString (listID, "Keyboard", AFTER);
    ListAddString (listID, "Monitor", AFTER);

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Display the list.
    SdShowInfoList (TITLE_TEXT, MSG_TEXT, listID);

    // Replace the value of the second item with a new value.
    if (ListSetCurrentString(listID, "Mouse") < 0) then
        MessageBox ("ListSetCurrentString failed.", SEVERE);
    endif;

    // Display the new altered list.
    SdShowInfoList (TITLE_TEXT, MSG_TEXT, listID);

    // Remove the list from memory.
    ListDestroy (listID);

end;

```

## ListSetIndex

The **ListSetIndex** function makes a specific element in a string or number list the current element, using an index. You can also use constants to traverse a list one element at a time or to jump to the beginning or end of a list. By using indices to access items in a list, you can treat numeric and string lists as arrays.

After you set the indexed element as the current element, you can use either the [ListCurrentItem](#) or [ListCurrentString](#) function in the script to retrieve the value of the indexed (current) item.




---

**Note** • The *ListSetIndex* function works with both string and number lists.

## Syntax

```
ListSetIndex (listID, nIndex);
```

## Parameters

**Table 74 • ListSetIndex Parameters**

Parameter	Description
<b>listID</b>	Specifies the name of the string or number list whose index is to be set.
<b>nIndex</b>	<p>Specifies the number of the element you want to set as the current element. List element numbering begins at zero (0). For example, if you enter 5 in the parameter nIndex, the item in the sixth physical location in the list becomes the current element.</p> <p>Pass a numeric value or one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>LISTFIRST</b>—Moves to the first element in the list.</li> <li>• <b>LISTLAST</b>—Moves to the last element in the list.</li> <li>• <b>LISTNEXT</b>—Moves to the next element in the list.</li> <li>• <b>LISTPREV</b>—Moves to the previous element in the list.</li> </ul>

## Return Values

**Table 75 • ListSetIndex Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully updated the current element in the list.
<b>&lt; 0</b>	Indicates that the function was unable to update the current element the list.
<b>END_OF_LIST (1)</b>	Indicates that the index is out of range of available list elements.
<b>ISERR_LIST_NOSUCHLIST (-501)</b>	Indicates that a list with the specified ID does not exist. Valid list IDs are return values from the <a href="#">ListCreate</a> function.

## ListSetIndex Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the ListSetIndex function.
 *
 * This example script displays the last three items from a list
 * of program folders.
 *
 * Note: If the target system has a shell other than Explorer,
 *       the GetGroupNameList function may return an error.
 *
\*-----*/

#define TITLE_TEXT "ListSetIndex Example"
#define MSG_TEXT  "Please note the last three items in this list."

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ListSetIndex(HWND);

function ExFn_ListSetIndex(hMSI)
    STRING    svString;
    LIST      listID;
    NUMBER    nResult, nIndex;
begin

    // Create string list.
    listID = ListCreate (STRINGLIST);

    // If an error occurred, report it; then terminate.
    if (listID = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Get the names of the program folders into a list.
    if GetGroupNameList (listID) < 0 then ;
        // Report ListCreate failure.
        MessageBox ("Unable to retrieve program folder names.", SEVERE);
    else
        // Display the list.
        SdShowInfoList( TITLE_TEXT, MSG_TEXT, listID);

        // Make the third item from the end of the list the current
        // the current element. If there are fewer than three items,
        // display them all. List indexing begins at 0.
        nIndex = ListCount (listID);

        if nIndex > 3 then
            nIndex = nIndex - 3;
        endif;
    endif;
endfunction

```

```

nResult = ListSetIndex (listID, nIndex);

// Loop while there are items to display.
while (nResult != END_OF_LIST)
    // Get the current list item.
    ListCurrentString (listID, svString);

    // Display the current list item.
    MessageBox (svString, INFORMATION);

    // Make the next item the current item.
    nResult = ListSetIndex (listID, LISTNEXT);
endwhile;
endif;

// Remove the list from memory.
ListDestroy (listID);

end;

```

## ListValid

The **ListValid** function indicates whether the list specified by listID is valid, that is, whether it has been initialized by calling [ListCreate](#) and not destroyed by calling [ListDestroy](#). This function works with both string and number lists.

### Syntax

```
ListValid ( listID );
```

## Parameters

**Table 76 • ListValid Parameters**

Parameter	Description
<b>listID</b>	Specifies the string or number list to be checked.

## Return Values

**Table 77 • ListValid Return Values**

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the list is valid.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the list is not valid.
<b>ISERR_LIST_NOSUCHLIST</b>	Indicates that the list has not been initialized.

## ListValid Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the ListValid function.
 *
 \*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

function OnBegin()
    LIST listID;
    number nListValid;
begin
    // Create an empty number list.
    listID = ListCreate (NUMBERLIST);

    // If an error occurred, report it; then terminate.
    if (listID = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Check whether the list ID is valid.
    nListValid = ListValid (listID);
    if (nListValid >= ISERR_SUCCESS) then
        MessageBox ("List ID is valid.", INFORMATION);
    else
        MessageBox ("List ID is not valid.", INFORMATION);
    endif;
end;

```

```

// Remove the list from memory.
ListDestroy (listID);

// Check whether the list ID is valid.
nListValid = ListValid (listID);
if (nListValid >= ISERR_SUCCESS) then
    MessageBox ("List ID is valid.", INFORMATION);
else
    MessageBox ("List ID is not valid.", INFORMATION);
endif;

// If you cut and paste this sample script into a project
// and run it, the following line aborts execution of the script.
abort;
end;

```

## ListValidType

The **ListValidType** function indicates whether the list specified by listID is of the specified type and is valid, that is, whether it has been initialized by calling [ListCreate](#) and not destroyed by calling [ListDestroy](#).

### Syntax

```
ListValidType ( listID, nType );
```



## Parameters

**Table 78 • ListValidType Parameters**

Parameter	Description
<b>listID</b>	Specifies the string or number list to be checked.
<b>nType</b>	Specifies the list type to which the specified list is compared. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li><b>NUMBERLIST</b>—Specifies a number list.</li> <li><b>STRINGLIST</b>—Specifies a string list.</li> </ul>

## Return Values

**Table 79 • ListValidType Return Values**

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the list is valid and is of the specified type.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the list is not valid or is not of the specified type.
<b>ISERR_LIST_NOSUCHLIST</b>	Indicates that the list has not been initialized.
<b>ISERR_LIST_TYPERISMATCH</b>	Indicates that the list is not of the specified type.

## ListValidType Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the ListValidType function.
 *
 \*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

function OnBegin()
    LIST listID;
    number nListValidType;
begin
    // Create an empty string list.
    listID = ListCreate (STRINGLIST);

```

```

// If an error occurred, report it; then terminate.
if (listID = LIST_NULL) then
    MessageBox ("Unable to create list.", SEVERE);
    abort;
endif;

// Check whether the list ID is valid and what type it is.
nListValidType = ListValidType (listID, NUMBERLIST);
if (nListValidType >= ISERR_SUCCESS) then
    MessageBox ("List ID is valid and is a number list.",
        INFORMATION);
else
    nListValidType = ListValidType (listID, STRINGLIST);
    if (nListValidType >= ISERR_SUCCESS) then
        MessageBox ("List ID is valid and is a string list.",
            INFORMATION);
    else
        MessageBox ("List ID is not valid.", INFORMATION);
    endif;
endif;

// Remove the list from memory.
ListDestroy (listID);

// Check whether the list ID is valid and what type it is.
nListValidType = ListValidType (listID, NUMBERLIST);
if (nListValidType >= ISERR_SUCCESS) then
    MessageBox ("List ID is valid and is a number list.",
        INFORMATION);
else
    nListValidType = ListValidType (listID, STRINGLIST);
    if (nListValidType >= ISERR_SUCCESS) then
        MessageBox ("List ID is valid and is a string list.",
            INFORMATION);
    else
        MessageBox ("List ID is not valid.", INFORMATION);
    endif;
endif;

// If you cut and paste this sample script into a project
// and run it, the following line aborts execution of the script.
abort;
end;

```

## ListWriteToFile

The **ListWriteToFile** function writes a string list to a text file. Each string appears on a separate line in the text file.



**Note** • Note if the file already exists and the pre-existing file is Unicode, it writes the file as Unicode. Otherwise, it writes the file as ANSI.

### Syntax

```
ListWriteToFile ( listID, szFileName );
```

## Parameters

**Table 80** • ListWriteToFile Parameters

Parameter	Description
<b>listID</b>	Specifies the name of a string list to write into a text file.
<b>szFileName</b>	Specifies the fully qualified name of the file to which the string list is to be written. If the file does not exist, it is created. If the file already exists, it is overwritten.

## Return Values

**Table 81** • ListWriteToFile Return Values

Return Value	Description
<b>0</b>	The function was successful.
<b>ISERR_LIST_NOSUCHLIST (-501)</b>	Indicates that a list with the specified ID does not exist. Valid list IDs are return values from the ListCreate function.
<b>&lt; 0</b>	The function failed.

## ListWriteToFile Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ListReadFromFile and ListWriteToFile
* functions.
*
* ListReadFromFile is called to read and display the
* Autoexec.bat file. After the list has been displayed,
* a remark is appended to the end of the list and the list
* is written to a new file. The original file remains
* unchanged.
*
* Note: In order for this script to run correctly, there must
*       be a batch file named Autoexec.bat in the root
*       directory of drive C.
*
\*-----*/
```

```

#define OLD_FILE "C:\\Autoexec.bat"
#define NEW_FILE "C:\\Autoexec.lst"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ListWriteToFile(HWND);

function ExFn_ListWriteToFile(hMSI)
    LIST listID;
begin
    // Create a string list.
    listID = ListCreate (STRINGLIST);

    // If an error occurred, report it; then terminate.
    if (listID = LIST_NULL) then
        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Read the file into a string list.
    if (ListReadFromFile (listID, OLD_FILE) < 0) then
        // Report ListReadFromFile failure.
        MessageBox ("Unable to read" + OLD_FILE + ".", SEVERE);
    else
        // Display the list.
        SdShowInfoList ("Demo", "The example list file:", listID);

        // Add a new string to the list.
        ListAddString (listID, "REM Added by IS setup.", AFTER);

        // Write the list to a file.
        if (ListWriteToFile (listID, NEW_FILE) < 0) then
            MessageBox ("Unable to write list to " + NEW_FILE + ".", SEVERE);
        else
            MessageBox ("Successfully wrote list " + NEW_FILE + ".", INFORMATION);
        endif;
    endif;

    // Remove the list from memory.
    ListDestroy (listID);

end;

```

## ListWriteToFileEx

The **ListWriteToFileEx** function writes or appends a string list to a text file. Each string appears on a separate line in the text file.

### Syntax

```
ListWriteToFileEx ( listID, szFileName, nOptions );
```

## Parameters

**Table 82 • ListWriteToFileEx Parameters**

Parameter	Description
<b>listID</b>	Specifies the name of a string list to write into a text file.
<b>szFileName</b>	Specifies the fully qualified name of the file to which the string list is to be written. If the file does not exist, it is created. If the file already exists and the <code>LWTF_OPTION_APPEND_TO_FILE</code> is not specified for <code>nOptions</code> , it is overwritten.
<b>nOptions</b>	<p>Specifies the encoding of the file and whether the string list can be appended to it if it already exists. Choose one or more of the following predefined constants:</p> <ul style="list-style-type: none"> <li>• <b>0</b>—If the file already exists and the pre-existing file is Unicode, writes the file as Unicode. Otherwise, writes the file as ANSI.</li> <li>• <b>LWTF_OPTION_WRITE_AS_UNICODE</b>—Writes the file as Unicode.</li> <li>• <b>LWTF_OPTION_WRITE_AS_ANSI</b>—Writes the file as ANSI.</li> <li>• <b>LWTF_OPTION_APPEND_TO_FILE</b>—Appends the contents of the list to the existing file. If the file does not exist, the function creates the file (which is the same behavior as if this option was not specified).</li> </ul> <p>You can combine two of these constants by using the bitwise OR operator (<code> </code>). For example, combine the <code>LWTF_OPTION_WRITE_AS_ANSI</code> and <code>LWTF_OPTION_APPEND_TO_FILE</code> constants if you want the contents of the string list to be append to the file as ANSI.</p>

## Return Values

**Table 83 • ListWriteToFileEx Return Values**

Return Value	Description
<b>ISERR_SUCCESS</b>	Indicates that the function was successful.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was not successful.

# LoadStringFromStringTable

The **LoadStringFromStringTable** function loads the value of the string entry specified by `szID` into `svString`. The `svString` buffer is automatically resized if necessary to accommodate the string value.

## Syntax

```
LoadStringFromStringTable ( szID, svString );
```

## Parameters

**Table 84** • LoadStringFromStringTable Parameters

Parameter	Description
<b>szID</b>	Specifies the string identifier of the string entry. Do not prefix the identifier with the at sign (@).
<b>svString</b>	Returns the string value associated with the identifier that is specified by szID.

## Return Values

**Table 85** • LoadStringFromStringTable Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the string value was successfully loaded into svString.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the string identifier was not found in the String Editor view.

## Additional Information

The **LoadStringFromStringTable** function is case-insensitive when it comes to string identifiers. Therefore, when you use a string identifier in your script, you do not necessarily need to match the case of the string identifier that is specified in the String Editor view. However, mixing case may prevent InstallShield from matching the string entries in the script to the corresponding string entries in the String Editor view at build time. Therefore, it is recommended that you use uppercase for all instances of string identifiers.

The **LoadStringFromStringTable** function is equivalent to the @ operator, with some exceptions:

- When you build a project that includes an InstallScript file (.rul) and the InstallScript code contains one or more references to string entries that use the @ operator, InstallShield validates the string entries at build time. If a string identifier in the project's InstallScript file is not defined as one of the project's string entries in the String Editor view, InstallShield displays build warning -7174. If you use **LoadStringFromStringTable**, InstallShield does not validate the string entries at build time; scripts that use **LoadStringFromStringTable** are assumed to have their own string-not-found error handling.
- If the specified identifier does not exist in the String Editor view, the @ operator causes a message box to be displayed at run time; **LoadStringFromStringTable** simply returns failure (< ISERR\_SUCCESS).

# LOBYTE



**Project** • This information applies to InstallScript projects.

The **LOBYTE** function extracts the low-order byte from the 16-bit integer value specified by shValue.

## Syntax

```
LOBYTE ( shValue );
```

## Parameters

**Table 86** • LOBYTE Parameters

Parameter	Description
<b>shValue</b>	Specifies the 16-bit integer from which you want to extract the lower byte.

## Return Values

This function returns the low-order byte of the integer.

# LogReadCustomNumber



**Project** • This information applies to InstallScript projects.

The **LogReadCustomNumber** function reads the number that is stored in the log file's custom logging section under the key name specified by szKey, and it returns the number data in nvValue.

## Syntax

```
LogReadCustomNumber( szKey, nvValue );
```

## Parameters

**Table 87** • LogReadCustomNumber Parameters

Parameter	Description
<b>szKey</b>	Specifies the key name that identifies the number being read from the log file.
<b>nvValue</b>	Returns the number data that is read from the log file.

## Return Values

**Table 88** • LogReadCustomNumber Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the function successfully read the number from the log file.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was unable to read the number from the log file.

## Additional Information

**LogReadCustomNumber** is not affected by whether logging is enabled or disabled. **LogReadCustomNumber** can read values that were entered with **LogWriteCustomNumber** and numbers that were entered as strings (for example, "123") with **LogWriteCustomString**. **LogReadCustomNumber** fails if used to read non-numeric data (for example, "123abc").

Custom log file entries do not affect maintenance or uninstallation of the product unless you add code to the script to read custom values and perform actions based on those values. **LogReadCustomString** cannot read data from the maintenance/uninstallation section of the log file (that is, the section where the installation automatically writes data, such as the files that are installed and the registry entries that are created, and from which it automatically reads data during maintenance or uninstallation).

To execute script code only when the product is being completely uninstalled, use an if-then statement:

```
if REMOVEALLMODE!=0 then
    /* this code is executed only during uninstallation */
endif;
```

To perform specific uninstallation actions when a particular feature is uninstalled, override the feature's UnInstalling event with the appropriate code.

## LogReadCustomNumber Example



**Project** • This information applies to InstallScript projects.

```
//-----
//
// InstallShield Example Script
//
```



```
// Demonstrates the LogWriteCustomString, LogWriteCustomNumber,
// LogReadCustomString, and LogReadCustomNumber functions.
//
//-----

// during installation, write some custom data to the log file
function OnMoved( )
    LIST listDrives;
    NUMBER nDriveCount, nvIgnore;
    STRING svDate;
begin

if (!MAINTENANCE) then
    // get current date
    GetSystemInfo(DATE, nvIgnore, svDate);

    // get current count of available drive letters
    listDrives = ListCreate(STRINGLIST);
    GetValidDrivesList(listDrives, -1, -1);
    nDriveCount = ListCount(listDrives);
    ListDestroy(listDrives);

    // write custom data to .ilg log file
    LogWriteCustomString("InstallDate", svDate);
    LogWriteCustomNumber("DriveCount", nDriveCount);
endif;

end;

// during a complete uninstallation, read custom data back from the log file
function OnMoving( )
    NUMBER nvDriveCount;
    STRING svInstallDate;
begin

if (REMOVEALLMODE) then
    LogReadCustomNumber("DriveCount", nvDriveCount);
    LogReadCustomString("InstallDate", svInstallDate);
    sprintfBox(INFORMATION, "Custom Log Data",
        "During installation, the drive count " +
        "was %d, and the date was %s.",
        nvDriveCount, svInstallDate);
endif;

end;
```

## LogReadCustomString



**Project** • This information applies to InstallScript projects.

**LogReadCustomString** reads the string that is stored the log file's custom logging section under the key name specified by szKey, and it returns the string data in svValue.

## Syntax

```
LogReadCustomString( szKey, svValue );
```

## Parameters

**Table 89** • LogReadCustomString Parameters

Parameter	Description
<b>szKey</b>	Specifies the key name that identifies the string being read from the log file.
<b>svValue</b>	Returns the string data that is read from the log file.

## Return Values

**Table 90** • LogReadCustomString Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the function successfully read the string from the log file.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was unable to read the string from the log file.

## Additional Information

**LogReadCustomString** is not affected by whether logging is enabled or disabled. **LogReadCustomString** can read values that were entered with either **LogWriteCustomString** or **LogWriteCustomNumber**.

Custom log file entries do not affect maintenance or uninstallation of the product unless you add code to the script to read custom values and perform actions based on those values. **LogReadCustomString** cannot read data from the maintenance/uninstallation section of the log file (that is, the section where the installation automatically writes data, such as the files that are installed and the registry entries that are created, and from which it automatically reads data during maintenance or uninstallation).



**Note** • If you selected the **Show Password dialog box during setup initialization** check box in the Password & Copyright panel of the Release Wizard or selected Yes for the Show Password Dialog setting in the Releases view, the default code for the OnCheckMediaPassword event handler function calls **LogWriteCustomString** with szKey set to MEDIA\_PASSWORD\_KEY in order to store a password-protected installation's password (as entered by the end user) so that maintenance and upgrade operations do not require that the same password be reentered. That default code also calls **LogWriteCustomString** with szKey set to MEDIA\_PASSWORD\_KEY in order to check whether the password is already stored before querying the end user for the password.

To execute script code only when the product is being completely uninstalled, use an if-then statement:

```
if REMOVEALLMODE!=0 then
    /* this code is executed only during uninstallation */
endif;
```

To perform specific uninstallation actions when a particular feature is uninstalled, override the feature's UnInstalling event with the appropriate code.

## LogReadCustomString Example



**Project** • This information applies to *InstallScript* projects.

```
//-----
//
//  InstallShield Example Script
//
//  Demonstrates the LogWriteCustomString, LogWriteCustomNumber,
//  LogReadCustomString, and LogReadCustomNumber functions.
//
//-----

// during installation, write some custom data to the log file
function OnMoved( )
    LIST listDrives;
    NUMBER nDriveCount, nvIgnore;
    STRING svDate;
begin

if (!MAINTENANCE) then
    // get current date
    GetSystemInfo(DATE, nvIgnore, svDate);

    // get current count of available drive letters
    listDrives = ListCreate(STRINGLIST);
    GetValidDrivesList(listDrives, -1, -1);
    nDriveCount = ListCount(listDrives);
    ListDestroy(listDrives);

    // write custom data to .ilg log file
    LogWriteCustomString("InstallDate", svDate);
    LogWriteCustomNumber("DriveCount", nDriveCount);
endif;

end;

// during a complete uninstallation, read custom data back from the log file
function OnMoving( )
    NUMBER nvDriveCount;
    STRING svInstallDate;
begin

if (REMOVEALLMODE) then
    LogReadCustomNumber("DriveCount", nvDriveCount);
    LogReadCustomString("InstallDate", svInstallDate);
    sprintfBox(INFORMATION, "Custom Log Data",
        "During installation, the drive count " +
        "was %d, and the date was %s.",
        nvDriveCount, svInstallDate);
endif;

end;
```

# LogWriteCustomNumber



**Project** • This information applies to InstallScript projects.

**LogWriteCustomNumber** writes the number specified by `nValue` to the log file in the custom logging section under the key name specified by `szKey`.

## Syntax

```
LogWriteCustomNumber( szKey, nValue );
```

## Parameters

**Table 91** • LogWriteCustomNumber Parameters

Parameter	Description
<b>szKey</b>	Specifies the key name that identifies the number being written to the log file. All key names written to a particular log file must be unique, regardless of whether they are written with LogWriteCustomNumber or LogWriteCustomString. If you specify a key name that already exists in the custom logging section, its value is overwritten.
<b>nValue</b>	Specifies the number to be written to the log file.

## Return Values

**Table 92** • LogWriteCustomNumber Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the function successfully wrote the number to the log file.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was unable to write the number to the log file.

## Additional Information

**LogWriteCustomNumber** fails if you call it while logging is disabled.

Custom log file entries do not affect maintenance or uninstallation of the product unless you add code to the script to read custom values and perform actions based on those values. **LogWriteCustomNumber** cannot write data to the maintenance/uninstallation section of the log file (that is, the section where the installation automatically writes data, such as the files that are installed and the registry entries that are created, and from which it automatically reads data during maintenance or uninstallation).

To execute script code only when the product is being completely uninstalled, use an if-then statement:

```
if REMOVEALLMODE!=0 then
    /* this code is executed only during uninstallation */
endif;
```

To perform specific uninstallation actions when a particular feature is uninstalled, override the feature's UnInstalling event with the appropriate code.

## LogWriteCustomNumber Example



**Project** • This information applies to InstallScript projects.

```
//-----
//
// InstallShield Example Script
//
// Demonstrates the LogWriteCustomString, LogWriteCustomNumber,
// LogReadCustomString, and LogReadCustomNumber functions.
//
//-----

// during installation, write some custom data to the log file
function OnMoved( )
    LIST listDrives;
    NUMBER nDriveCount, nvIgnore;
    STRING svDate;
begin

if (!MAINTENANCE) then
    // get current date
    GetSystemInfo(DATE, nvIgnore, svDate);

    // get current count of available drive letters
    listDrives = ListCreate(STRINGLIST);
    GetValidDrivesList(listDrives, -1, -1);
    nDriveCount = ListCount(listDrives);
    ListDestroy(listDrives);

    // write custom data to .ilg log file
    LogWriteCustomString("InstallDate", svDate);
    LogWriteCustomNumber("DriveCount", nDriveCount);
endif;

end;

// during a complete uninstallation, read custom data back from the log file
function OnMoving( )
    NUMBER nvDriveCount;
    STRING svInstallDate;
begin

if (REMOVEALLMODE) then
    LogReadCustomNumber("DriveCount", nvDriveCount);
    LogReadCustomString("InstallDate", svInstallDate);
    sprintfBox(INFORMATION, "Custom Log Data",
        "During installation, the drive count " +
        "was %d, and the date was %s.",
        nvDriveCount, svInstallDate);
endif;

end;
```

```
end;
```

# LogWriteCustomString



**Project** • This information applies to InstallScript projects.

**LogWriteCustomString** writes the string specified by `szValue` to the log file in the custom logging section under the key name specified by `szKey`.

## Syntax

```
LogWriteCustomString( szKey, szValue );
```

## Parameters

**Table 93** • LogWriteCustomString Parameters

Parameter	Description
<b>szKey</b>	Specifies the key name that identifies the string being written to the log file. All key names written to a particular log file must be unique, regardless of whether they are written with <code>LogWriteCustomString</code> or <code>LogWriteCustomNumber</code> . If you specify a key name that already exists in the custom logging section, its value is overwritten.
<b>szValue</b>	Specifies the value to be written to the log file.

## Return Values

**Table 94** • LogWriteCustomString Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the function successfully wrote the string to the log file.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was unable to write the string to the log file.

## Additional Information

**LogWriteCustomString** fails if you call it while logging is disabled.

Custom log file entries do not affect maintenance or uninstallation of the product unless you add code to the script to read custom values and perform actions based on those values. **LogWriteCustomString** cannot write data to the maintenance/uninstallation section of the log file (that is, the section where the installation automatically writes data, such as the files that are installed and the registry entries that are created, and from which it automatically reads data during maintenance or uninstallation).



**Note** • If you selected the **Show Password dialog box during setup initialization** check box in the Password & Copyright panel of the Release Wizard or selected Yes for the Show Password Dialog setting in the Releases view, the default code for the OnCheckMediaPassword event handler function calls **LogWriteCustomString** with szKey set to MEDIA\_PASSWORD\_KEY in order to store a password-protected installation's password (as entered by the end user) so that maintenance and upgrade operations do not require that the same password be reentered. That default code also calls **LogWriteCustomString** with szKey set to MEDIA\_PASSWORD\_KEY in order to check whether the password is already stored before querying the end user for the password.

To execute script code only when the product is being completely uninstalled, use an if-then statement:

```
if REMOVEALLMODE!=0 then
    /* this code is executed only during uninstallation */
endif;
```

To perform specific uninstallation actions when a particular feature is uninstalled, override the feature's UnInstalling event with the appropriate code.

## LogWriteCustomString Example



**Project** • This information applies to InstallScript projects.

```
//-----
//
// InstallShield Example Script
//
// Demonstrates the LogWriteCustomString, LogWriteCustomNumber,
// LogReadCustomString, and LogReadCustomNumber functions.
//
//-----

// during installation, write some custom data to the log file
function OnMoved( )
    LIST listDrives;
    NUMBER nDriveCount, nvIgnore;
    STRING svDate;
begin

if (!MAINTENANCE) then
    // get current date
    GetSystemInfo(DATE, nvIgnore, svDate);

    // get current count of available drive letters
    listDrives = ListCreate(STRINGLIST);
    GetValidDrivesList(listDrives, -1, -1);
    nDriveCount = ListCount(listDrives);
    ListDestroy(listDrives);

    // write custom data to .ilg log file
    LogWriteCustomString("InstallDate", svDate);
    LogWriteCustomNumber("DriveCount", nDriveCount);
endif;
```

```

end;

// during a complete uninstallation, read custom data back from the log file
function OnMoving( )
    NUMBER nvDriveCount;
    STRING svInstallDate;
begin
    if (REMOVEALLMODE) then
        LogReadCustomNumber("DriveCount", nvDriveCount);
        LogReadCustomString("InstallDate", svInstallDate);
        sprintfBox(INFORMATION, "Custom Log Data",
            "During installation, the drive count " +
            "was %d, and the date was %s.",
            nvDriveCount, svInstallDate);
    endif;
end;

```

## LongPathFromShortPath

Use the **LongPathFromShortPath** function to convert a short file name to its equivalent long file name.

For an explanation of long file names, see Long File Names.

### Syntax

```
LongPathFromShortPath ( svPath );
```



## Parameters

**Table 95** • LongPathFromShortPath Parameters

Parameter	Description
<b>svPath</b>	Specifies a short file name and returns its associated long file name.

## Return Values

**Table 96** • LongPathFromShortPath Return Values

Return Value	Description
<b>0</b>	Indicates that the function was successful.
<b>&lt; 0</b>	Indicates that the function was not successful.

## LongPathFromShortPath Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the functions LongPathToShortPath and
 * LongPathFromShortPath.
 *
 * First, LongPathToShortPath is called to convert a long path
 * to a short path. Then LongPathFromShortPath is called to
 * convert the short path to a long path. The result of each
 * is displayed in a message box.
 *
 \*-----*/

#define LONG_PATH "C:\\Program files"
#define TITLE "LongPathToShortPath & LongPathFromShortPath"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_LongPathFromShortPath(HWND);

function ExFn_LongPathFromShortPath(hMSI)
    STRING svPath, szTitle, szMsg;
begin

    // Prompt user to enter a long path.
    szMsg = "Please select an existing long path:";

```

```

AskPath (szMsg, LONG_PATH, svPath);

// Display the long path.
szMsg = "The long path is shown below: \n\n%s";
sprintfBox (INFORMATION, TITLE, szMsg, svPath);

//Convert the long path to a short path.
if (LongPathToShortPath (svPath) < 0) then;
    MessageBox ("LongPathToShortPath failed.", SEVERE);
    abort;
else
    // Display the short path.
    szMsg = "The short path is shown below: \n\n%s";
    sprintfBox (INFORMATION, TITLE, szMsg, svPath);
endif;

// Restore the long path from the short path.
if (LongPathFromShortPath (svPath) < 0) then
    MessageBox ("LongPathFromShortPath failed.", SEVERE);
else
    // Display the restored long path.
    szMsg = "The restored long path is shown below: \n\n%s";
    sprintfBox (INFORMATION, TITLE, szMsg, svPath);
endif;

end;

```

## LongPathToQuote

The **LongPathToQuote** function places double quotation marks around a long file name or removes the double quotation marks from a long file name.

For an explanation of long file names, see Long File Names.

Add double quotation marks to long file names that contain spaces before passing the long file names to the command line. You must remove the double quotation marks from long file names before converting them to short file names using the [LongPathToShortPath](#) function. If you do not, the quoted long file name remains intact.



**Note** • This function adds quotation marks only if there is a space character in the file name. For example, quotation marks are not added around C:\ThisismyApp because it is a long file name without a space.

### Syntax

```
LongPathToQuote ( svPath, nParameter );
```

## Parameters

**Table 97 • LongPathToQuote Parameters**

Parameter	Description
<b>svPath</b>	Specifies a long file name and returns that name with or without quotation marks, depending on the value passed in nParameter.
<b>nParameter</b>	<p>Specifies whether quotation marks are to be added to the long path or removed from the long path. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>TRUE</b>—Quotation marks are added to the long path.</li> <li>• <b>FALSE</b>—Quotation marks are removed from the long path.</li> </ul>

## Return Values

**Table 98 • LongPathToQuote Return Values**

Return Value	Description
<b>0</b>	Indicates that the function was successful.
<b>&lt; 0</b>	Indicates that the function was not successful.

## LongPathToQuote Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the LongPathToQuote function.
*
* This script calls LongPathToQuote to place double quotation
* marks around a long file name. The result is displayed in
* in a dialog. Then, LongPathToQuote is called again to
* remove the quotation marks and the result displayed in a
* dialog.
*
\*-----*/

// Define a constant for the base path (a long file name).
```

```

#define BASE_PATH "C:\\Program Files"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_LongPathToQuote(HWND);

function ExFn_LongPathToQuote(hMSI)
    STRING svPath, szMainDirectory, szMsg;
begin
    // Set up parameter for call to LongPathToQuote.
    svPath = BASE_PATH;

    // Place double quotation marks around the long file name in svPath.
    if (LongPathToQuote (svPath, TRUE) < 0) then
        MessageBox ("First call to LongPathToQuote failed.", SEVERE);
        abort;
    endif;

    // Display the quoted long file name in svPath.
    szMsg = "The quoted long file name:\n\n" + svPath;
    MessageBox (szMsg, INFORMATION);

    // Remove the quotation marks from the long file name in svPath.
    if (LongPathToQuote (svPath, FALSE) < 0) then
        MessageBox ("Second call to LongPathToQuote failed.", SEVERE);
        abort;
    endif;

    // Display the long file name with quotation marks removed.
    szMsg = "The unquoted long file name is shown below: \n\n" + svPath;

    MessageBox (szMsg, INFORMATION);

end;

```

## LongPathToShortPath

The **LongPathToShortPath** function converts a long file name to its equivalent short file name. The parameter `svPath` can be an absolute path or a relative path, and it may include a file name; but the folder or file it specifies must exist on the target system.


For an explanation of long file names, see Long File Names.

### Syntax

```
LongPathToShortPath ( svPath );
```

## Parameters

**Table 99** • LongPathToShortPath Parameters

Parameter	Description
<b>svPath</b>	Specifies a long file name and returns its associated short file name.
 <p><b>Note</b> • <i>LongPathToShortPath removes trailing backslashes from long file names.</i></p>	

## Return Values

**Table 100** • LongPathToShortPath Return Values

Return Value	Description
<b>0</b>	Indicates that the function was successful.
<b>&lt; 0</b>	Indicates that the function was not successful.

## Additional Information

Since **LongPathToShortPath** can succeed only if the specified folder or file can be found on the target system, you may need to set the current folder before specifying a relative path. For example, if **svPath** contains the relative path **InstallShield**, which exists in the folder **C:\Program Files**, the setup is not able to find it unless the current folder is **C:\Program Files**. Use the [ChangeDirectory](#) function to change the current folder when necessary before calling **LongPathToShortPath** so that the target folder or path can be found.

## LongPathToShortPath Example



**Note** • *To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.*

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the functions LongPathToShortPath and
* LongPathFromShortPath.
*
* First, LongPathToShortPath is called to convert a long path
* to a short path. Then LongPathFromShortPath is called to
* convert the short path to a long path. The result of each
* is displayed in a message box.
*
\*-----*/

```

```

#define LONG_PATH "C:\\Program Files"
#define TITLE "LongPathToShortPath & LongPathFromShortPath"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_LongPathToShortPath(HWND);

function ExFn_LongPathToShortPath(hMSI)
    STRING svPath, szTitle, szMsg;
begin
    // Prompt user to enter a long path.
    szMsg = "Please select an existing long path:";
    AskPath (szMsg, LONG_PATH, svPath);

    // Display the long path.
    szMsg = "The long path is shown below: \n\n%s";
    sprintfBox (INFORMATION, TITLE, szMsg, svPath);

    //Convert the long path to a short path.
    if (LongPathToShortPath (svPath) < 0) then;
        MessageBox ("LongPathToShortPath failed.", SEVERE);
        abort;
    else
        // Display the short path.
        szMsg = "The short path is shown below: \n\n%s";
        sprintfBox (INFORMATION, TITLE, szMsg, svPath);
    endif;

    // Restore the long path from the short path.
    if (LongPathFromShortPath (svPath) < 0) then
        MessageBox ("LongPathFromShortPath failed.", SEVERE);
    else
        // Display the restored long path.
        szMsg = "The restored long path is shown below: \n\n%s";
        sprintfBox (INFORMATION, TITLE, szMsg, svPath);
    endif;
end;

```

## LOWORD

The **LOWORD** function extracts the low-order word (two bytes) from the 32-bit integer value specified by lValue.

### Syntax

```
LOWORD ( lValue );
```

## Parameters

**Table 101** • LOWORD Parameters

Parameter	Description
<b>IValue</b>	Specifies the 32-bit integer from which you want to extract the lower two bytes.

## Return Values

This function returns the low-order word (lower two bytes) of the integer.

## LOWORD Example

To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates HIWORD and LOWORD.
 *
 * This example script shows how to use HIWORD and LOWORD to
 * to get the low-order word and the high-order word from a value.
 \*-----*/

#define TITLE_TEXT "LOWORD/HIWORD Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_LOWORD(HWND);

function ExFn_LOWORD(hMSI)
    STRING  szMsg;
    NUMBER  nData, nLOWORD, nHIWORD;
begin

    nData = 305419896; // hex value: 12345678
    // Get the low-order word, 22136 (hex value: 5678).

    nLOWORD = LOWORD (nData);

    // Get the high-order word, 4660 (hex value: 1234).
    nHIWORD = HIWORD (nData);

    // Display the results.
    szMsg = "LOWORD: %ld\nHIWORD: %ld";
    sprintfBox (INFORMATION, TITLE_TEXT, szMsg, nLOWORD, nHIWORD);

end;
```

# MaintenanceStart

The **MaintenanceStart** function creates a registry key and associated values that are used during initialization for maintenance or uninstallation and that provide information about the application to Add or Remove Programs. This function is called by the default code for the OnMoveData event handler.



**Note** • The InstallScript engine currently does not support writing or reading Add or Remove Programs information for a product in the 64-bit part of the registry. Therefore, using the `REGDB_OPTION_WOW64_64KEY` option with the `REGDB_OPTIONS` system variable is not supported for this registry function. Enabling the `REGDB_OPTION_WOW64_64KEY` option has no effect on where registry entries are created by this function.

**MaintenanceStart** creates the following values under the application uninstallation registry key:

**Table 102** • Application Uninstallation Registry Key Values

Value Name	Value Data
<b>Comments</b>	The value of the system variable <code>IFX_PRODUCT_COMMENTS</code> , if non-null; otherwise, the function does not create an entry.
<b>Contact</b>	The value of the system variable <code>IFX_PRODUCT_SUPPORT_CONTACT</code> , if non-null; otherwise, the function does not create an entry.
<b>DisplayIcon</b>	The value of the system variable <code>IFX_PRODUCT_ICON</code> , if non-null; otherwise, the function does not create an entry.
<b>DisplayName</b>	The value of the system variable <code>IFX_PRODUCT_NAME</code> , if non-null; otherwise, the function does not create an entry.
<b>DisplayVersion</b>	The value of the system variable <code>IFX_PRODUCT_VERSION</code> .
<b>HelpLink</b>	The value of the system variable <code>IFX_PRODUCT_SUPPORT_URL</code> , if non-null; otherwise, the function does not create an entry.
<b>HelpTelephone</b>	The value of the system variable <code>IFX_PRODUCT_SUPPORT_PHONE</code> , if non-null; otherwise, the function does not create an entry.
<b>InstallDate</b>	The date on which the installation is run, in the format <code>yyyymmdd</code> .
<b>InstallLocation</b>	The value of the system variable <code>TARGETDIR</code> .
<b>InstallSource</b>	The value of the system variable <code>SRCDIR</code> .
<b>Language</b>	The value of the system variable <code>SELECTED_LANGUAGE</code> .
<b>LogFile</b>	<code>&lt;DISK1TARGET&gt;\Setup.ilg</code>
<b>LogMode</b>	The value of the system variable <code>MAINT_OPTION</code> .



**Table 102** • Application Uninstallation Registry Key Values (cont.)

Value Name	Value Data
<b>ModifyPath</b>	<p>The value of the system variable <a href="#">UNINSTALL_STRING</a> if <a href="#">ADDREMOVE_HIDECHANGEOPTION</a> is FALSE and either of the following two conditions is true:</p> <ul style="list-style-type: none"> <li>• <a href="#">ADDREMOVE_HIDEREMOVEOPTION</a> is non-zero</li> <li>• <a href="#">ADDREMOVE_HIDEREMOVEOPTION</a> and <a href="#">ADDREMOVE_COMBINEDBUTTON</a> are both FALSE</li> </ul> <p>Otherwise, the function does not create an entry.</p>
<b>NoModify</b>	The value of the system variable <a href="#">ADDREMOVE_HIDEREMOVEOPTION</a> .
<b>NoRemove</b>	The value of the system variable <a href="#">ADDREMOVE_HIDEREMOVEOPTION</a> .
<b>NoRepair</b>	1 if either <a href="#">ADDREMOVE_HIDECHANGEOPTION</a> or <a href="#">ADDREMOVE_HIDEREMOVEOPTION</a> is non-zero; otherwise, the function does not create an entry.
<b>ProductGuid</b>	The value of the system variable <a href="#">PRODUCT_GUID</a> .
<b>ProductId</b>	The value of the system variable <a href="#">IFX_PRODUCT_REGISTEREDSERIALNUM</a> , if non-null; otherwise, the function does not create an entry.
<b>Publisher</b>	The value of the system variable <a href="#">IFX_COMPANY_NAME</a> , if non-null; otherwise, the function does not create an entry.
<b>Readme</b>	The value of the system variable <a href="#">IFX_PRODUCT_README</a> , if non-null; otherwise, the function does not create an entry.
<b>RegCompany</b>	The value of the system variable <a href="#">IFX_PRODUCT_REGISTEREDCOMPANY</a> , if non-null; otherwise, the function does not create an entry.
<b>RegOwner</b>	The value of the system variable <a href="#">IFX_PRODUCT_REGISTEREDOWNER</a> , if non-null; otherwise, the function does not create an entry.
<b>SystemComponent</b>	1 if the system variable <a href="#">ADDREMOVE_SYSTEMCOMPONENT</a> is non-zero; otherwise, the function does not create an entry.
<b>UninstallString</b>	<p>The value of the system variable <a href="#">UNINSTALL_STRING</a>. Additionally, the value of the system variable <a href="#">ADDREMOVE_STRING_REMOVEONLY</a> is appended to this registry data if <a href="#">ADDREMOVE_HIDEREMOVEOPTION</a> is FALSE and either of the following two conditions is true:</p> <ul style="list-style-type: none"> <li>• <a href="#">ADDREMOVE_HIDECHANGEOPTION</a> is non-zero</li> <li>• <a href="#">ADDREMOVE_HIDECHANGEOPTION</a> and <a href="#">ADDREMOVE_COMBINEDBUTTON</a> are both FALSE</li> </ul>

**Table 102** • Application Uninstallation Registry Key Values (cont.)

Value Name	Value Data
<b>URLInfoAbout</b>	The value of the system variable <a href="#">IFX_PRODUCT_URL</a> , if non-null; otherwise, the function does not create an entry.
<b>URLUpdateInfo</b>	The value of the system variable <a href="#">IFX_PRODUCT_UPDATE_URL</a> , if non-null; otherwise, the function does not create an entry.
<b>Version</b>	The packed DWORD equivalent of the data in the DisplayVersion value.
<b>VersionMajor</b>	The first byte of the data in the Version value.
<b>VersionMinor</b>	The second byte of the data in the Version value.

## Syntax

```
MaintenanceStart ( );
```

## Parameters

None

## Return Values

**Table 103** • MaintenanceStart Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully created the registry key and its associated values.
<b>&lt; 0</b>	Indicates that the function was unable to create the registry key and its associated values.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

## Additional Information

InstallScript installations always create VersionMajor and VersionMinor registry values in the Uninstall key. This applies to new installations that are created in InstallShield 2016, as well as installations that are upgraded from InstallShield 2009 or earlier. Previously, in InstallShield 2009 and earlier, the names of the values that InstallScript installations created were MajorVersion and MinorVersion; these are no longer created.

When the **MaintenanceStart** function is called, it creates the VersionMajor and VersionMinor value names in the registry. By default, it also deletes the MajorVersion and MinorVersion value names if they exist. If you do not want the MajorVersion and MinorVersion value names to be deleted from target systems, you can use the [REGDB\\_OPTIONS](#) option called REGDB\_OPTION\_NO\_DELETE\_OLD\_MAJMIN\_VERSION. If you want to continue using only the MajorVersion and MinorVersion value names, you must delete VersionMajor and VersionMinor after **MaintenanceStart** returns.

# MediaGetData



**Project** • This information applies to *InstallScript* projects.

The **MediaGetData** function retrieves information about a file media library.

**MediaGetData** calls `MediaGetDataEx(szMediaSource, nInfo, nvResult, svResult, FALSE)`. For information on parameters and return values for **MediaGetData**, see [MediaGetDataEx](#).

## Syntax

```
MediaGetData ( szMediaSource, nInfo, nvResult, svResult );
```

# MediaGetDataEx



**Project** • This information applies to *InstallScript* projects.

The **MediaGetDataEx** function retrieves information about a file media library, including media information that is stored in the project's string entries.

## Syntax

```
MediaGetDataEx ( szMediaSource, nInfo, nvResult, svResult, bCheckStringTable );
```

## Parameters

**Table 104** • MediaGetDataEx Parameters

Parameter	Description
<b>szMediaSource</b>	Specifies the name of the file media library whose information is to be retrieved; typically, the value of this argument is the system variable MEDIA.
<b>nInfo</b>	<p>Specifies the type of information to retrieve. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>MEDIA_FIELD_ADDREMOVE_NOMODIFY</b>—Retrieves the setting that you specified in the Disable Change Button setting in the General Information view. The value is returned in nvResult; 0 corresponds to No and 1 corresponds to Yes.</li> <li>• <b>MEDIA_FIELD_ADDREMOVE_NOREMOVE</b>—Retrieves the setting that you specified in the Disable Remove Button setting in the General Information view. The value is returned in nvResult; 0 corresponds to No and 1 corresponds to Yes.</li> <li>• <b>MEDIA_FIELD_COMPANY_NAME</b>—Retrieves the company name that you specified in the General Information view. The company name is returned in svResult.</li> <li>• <b>MEDIA_FIELD_MEDIA_FLAGS</b>—Retrieves the format of the specified file media library. The number returned in nvResult includes any applicable values from among the following bit flags: <ul style="list-style-type: none"> <li>MEDIA_FLAG_FORMAT_DIFFERENTIAL: Indicates a differential file media library.</li> <li>MEDIA_FLAG_UPDATEMODE_SUPPORTED: Indicates an update enabled file media library. This flag is always set.</li> </ul> </li> <li>• <b>MEDIA_FIELD_PREVIOUS_VERSIONS</b>—Retrieves the version information that you specified in the Supported Version(s) property of the Releases view or in the Update panel in the Release Wizard. The version information is returned in svResult.</li> <li>• <b>MEDIA_FIELD_PRODUCT_COMMENTS</b>—Retrieves the comments that you specified in the ARP Comments setting in the General Information view. The value is returned in svResult.</li> <li>• <b>MEDIA_FIELD_PRODUCT_EXE</b>—Retrieves the executable name that you specified in the General Information view. The executable name is returned in svResult.</li> </ul>

**Table 104 •** MediaGetDataEx Parameters (cont.)

Parameter	Description
<b>nInfo (cont.)</b>	<ul style="list-style-type: none"> <li>• <b>MEDIA_FIELD_PRODUCT_ICON</b>—Retrieves the display icon that you specified in the Display Icon setting in the General Information view. The value is returned in svResult.</li> <li>• <b>MEDIA_FIELD_PRODUCT_NAME</b>—Retrieves the product name that you specified in the General Information view. The product name is returned in svResult.</li> <li>• <b>MEDIA_FIELD_PRODUCT_README</b>—Retrieves the readme file that you specified in the Read Me setting in the General Information view. The value is returned in svResult.</li> <li>• <b>MEDIA_FIELD_PRODUCT_SUPPORT_CONTACT</b>—Retrieves the support contact that you specified in the Support Contact setting in the General Information view. The value is returned in svResult.</li> <li>• <b>MEDIA_FIELD_PRODUCT_SUPPORT_PHONE</b>—Retrieves the support phone number that you specified in the Support Phone Number setting in the General Information view. The value is returned in svResult.</li> <li>• <b>MEDIA_FIELD_PRODUCT_SUPPORT_URL</b>—Retrieves the support URL that you specified in the Support URL setting in the General Information view. The value is returned in svResult.</li> <li>• <b>MEDIA_FIELD_PRODUCT_UPDATE_URL</b>—Retrieves the product update URL that you specified in the Product Update URL setting in the General Information view. The value is returned in svResult.</li> <li>• <b>MEDIA_FIELD_PRODUCT_URL</b>—Retrieves the publisher/product URL that you specified in the Publisher/Product URL setting in the General Information view. The value is returned in svResult.</li> <li>• <b>MEDIA_FIELD_PRODUCT_VERSION</b>—Retrieves the version information that you specified in the General Information view. The version information is returned in svResult.</li> <li>• <b>MEDIA_FIELD_TARGETDIR</b>—Retrieves the value of TARGETDIR that you specified in the General Information view.</li> </ul>
<b>bCheckStringTable</b>	Specifies whether svResult returns the string that you entered in InstallShield (FALSE) or, if the string that you entered in InstallShield is a string identifier, the string value that is associated with that identifier (TRUE). If bCheckStringTable is TRUE but the string that you entered in InstallShield is not a string identifier, svResult returns the string that you entered in InstallShield.

## Return Values

**Table 105** • MediaGetDataEx Return Values

Return Value	Description
0	MediaGetData successfully retrieved the requested information.
< 0	MediaGetData failed to retrieve the requested information.

## MessageBeep

The **MessageBeep** function plays the default system sound.



**Tip** • You can also give audio cues by calling *PlayMMedia* to play an audio file.

### Syntax

```
MessageBeep ( nReserved );
```

### Parameters

**Table 106** • MessageBeep Parameters

Parameter	Description
<b>nReserved</b>	Pass zero in this parameter. No other value is allowed.

### Return Values

This function has no return values.

## MessageBeep Example

To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the MessageBeep function.
 *
 \*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_MessageBeep(HWND);
```

```
function ExFn_MessageBeep(hMSI)
begin

    // Play the default system sound.
    MessageBeep (0);

    // Display a message box.
    MessageBox ("Please listen for a sound.", INFORMATION);

    // Play the default system sound.
    MessageBeep (0);

end;
```

## MessageBox



**Project** • This information applies to the following project types:

- Basic MSI
- InstallScript
- InstallScript MSI

The **MessageBox** function presents a dialog that contains a message, an icon that indicates the nature of the message (information, warning, or severe), and an OK button. The default title depends on the value of *nType*, which also indicates the icon type. For example, if you pass **INFORMATION** in *nType*, the title **Information** appears in the title bar. The titles are blank by default, unless you set them by calling [SetDialogTitle](#) before calling **MessageBox**. If the titles are blank, the title bar text for message box is the product name (from **IFX\_SETUP\_TITLE**).

This function uses the Microsoft Windows API **MessageBox**. The operating environment, not the installation, determines the size and location of the message box. The operating environment also generates the text for the OK button in the local language (the language that the operating system is running under). You cannot change the text on this button. For more information regarding the use of Windows **MessageBox** types, consult the description of the **MessageBox** Windows API function in the appropriate Windows SDK.

Note the following when using Windows message box constants:

- Some Windows **MessageBox** type constants are predefined in the **ISRTWindows.h** file that is provided in the *InstallShield Program Files Folder\Script\Isrt\Include* folder. This file is automatically included in your installation when you include **lfx.h** in your script. You do not need to redefine any constants that are defined in **ISRTWindows.h**; doing so will result in a compiler warning. To determine which constants are predefined, refer to the **ISRTWindows.h** file.
- To use constants that are not defined in **ISRTWindows.h**, you must define them (using **#define**) in the declaration block of your installation script. You cannot simply include the **Windows.h** file that is usually part of a C++ program. The values that you need to assign to the undefined constants can generally be found in an include file that is provided with the appropriate Windows SDK or development tool. (For Microsoft Visual C++, most constants can be found in the **Winuser.h** file, which is located in the *InstallShield Program Files Folder\Script\Resource* folder.)
- Windows and InstallShield message box constants cannot be used together in an installation. If an InstallShield message box constant is combined with a Windows message box constant using the OR operator (**|**), the Windows message box constant is ignored.

- Some Windows message box styles are not supported on some Windows platforms. To determine whether a particular style is supported on the operating systems that are targeted by the installation, consult the appropriate Windows SDK.
- When a Windows message box style is used by the **MessageBox** function, the caption (title) of the message box is **Install**. If you need to display a different caption, use the [MessageBoxEx](#) function.

## Syntax

```
MessageBox ( szMsg, nType );
```

## Parameters

**Table 107 • MessageBox Parameters**

Parameter	Description
<b>szMsg</b>	Specifies the message to display. InstallShield does not automatically break the text of the message into separate lines to fit in the message box. If the message is too long for one line, insert a line break by embedding newline escape characters ( \n ) at appropriate places in the string.
<b>nType</b>	<p>Specifies the type of message box to create and the type of icon to display in the message box. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• INFORMATION</li> <li>• WARNING</li> <li>• SEVERE</li> </ul> <p>Any Windows API MessageBox type can also be specified in this parameter. Multiple styles can be combined logically using the OR operator to produce the required MessageBox type.</p>

## Return Values

The return value is not significant unless you are using standard Microsoft Windows message box styles. If you are using these styles, the return value is the same as the return value from the MessageBox API functions.

## Additional Information

The dialog that is displayed by the MessageBox function cannot be displayed with a skin; it appears the same regardless of whether you have specified a skin.

## MessageBox Example



**Project •** This information applies to the following project types:

- Basic MSI
- InstallScript



- *InstallScript MSI*



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the MessageBox function.
 *
 * This script displays three message boxes, each with a
 * different message and icon.
 *
 \*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_MessageBox(HWND);

function ExFn_MessageBox(hMSI)
    STRING szMsg;
begin

    // Display a message box that shows the information icon.
    szMsg = "This will install Example Program.";
    MessageBox (szMsg, INFORMATION);

    // Display a message box that shows the warning icon.
    szMsg = "Installing this version will replace previous one.";
    MessageBox (szMsg, WARNING);

    // Display a message box that shows the severe icon.
    szMsg = "Cannot install this application on floppy drives.";
    MessageBox (szMsg, SEVERE);

end;
```

## MessageBoxEx



**Project** • This information applies to the following project types:

- *Basic MSI*
- *InstallScript*
- *InstallScript MSI*

The **MessageBoxEx** function presents a dialog that contains a message, an icon that indicates the nature of the message (information, warning, or severe), and an OK button. The title is set by szCaption.

This function uses the Microsoft Windows API MessageBox. The operating environment, not InstallShield, determines the size and location of the message box. The operating environment also generates the text for the OK button in the local language (the language that the operating system is running under). You cannot change the text in this button. For more information regarding the use of Windows MessageBox types, consult the description of the MessageBox Windows API function in the appropriate Windows SDK.

Note the following when using Windows message box constants:

- Some Windows **MessageBox** type constants are predefined in the **ISRTWindows.h** file that is provided in the *InstallShield Program Files Folder\Script\Isrt\Include* folder. This file is automatically included in your installation when you include **lfx.h** in your script. You do not need to redefine any constants that are defined in **ISRTWindows.h**; doing so will result in a compiler warning. To determine which constants are predefined, refer to the **ISRTWindows.h** file.
- To use constants that are not defined in **ISRTWindows.h**, you must define them (using #define) in the declaration block of your installation script. You cannot simply include the **Windows.h** file that is usually part of a C++ program. The values that you need to assign to the undefined constants can generally be found in an include file that is provided with the appropriate Windows SDK or development tool. (For Microsoft Visual C++, most constants can be found in the **Winuser.h** file, which is located in the *InstallShield Program Files Folder\Script\Resource* folder.)
- Windows and InstallShield message box constants cannot be used together in an installation. If an InstallShield message box constant is combined with a Windows message box constant using the OR operator, the Windows message box constant will be ignored.
- Some Windows message box styles are not supported on some Windows platforms. To determine whether a particular style is supported on the operating system(s) targeted by the installation, consult the appropriate Windows SDK.

## Syntax

```
MessageBoxEx( szMsg, szCaption, nType );
```

## Parameters

**Table 108** • MessageBoxEx Parameters

Parameter	Description
<b>szMsg</b>	Specifies the message to display. InstallShield does not automatically break the text of the message into separate lines to fit in the message box. If the message is too long for one line, insert a line break by embedding newline escape characters ( \n ) at appropriate places in the string.
<b>szCaption</b>	Specifies the title to display. If you pass a null string ("" ) in this parameter, the value of the system variable <a href="#">IFX_SETUP_TITLE</a> is used as the dialog title.
<b>nType</b>	<p>Specifies the type of message box to create and the type of icon to display in the message box. Pass one of the following predefined constants in this parameter (Explorer shell icons are shown):</p> <ul style="list-style-type: none"> <li>• INFORMATION</li> <li>• WARNING</li> <li>• SEVERE</li> </ul> <p>Any Windows API MessageBox type can also be specified in this parameter. Multiple styles can be combined logically with the OR operator ( ) to produce the required MessageBox type.</p>

## Return Values

The return value is not significant unless you are using standard Microsoft Windows message box styles. If you are using these styles, the return value is the same as the return value from the MessageBox API functions.

## Additional Information

The message box that is displayed by the MessageBoxEx function cannot be displayed with a skin; it appears the same regardless of whether you have specified a skin.

# NumToStr

The **NumToStr** function converts a number to a string.

## Syntax

```
NumToStr ( svString, nValue );
```

## Parameters

**Table 109 • NumToStr Parameters**

Parameter	Description
<b>svString</b>	Returns the string equivalent of nValue.
<b>nValue</b>	Specifies the number to convert to a string.

## Return Values

**Table 110 • NumToStr**

Return Value	Description
<b>0</b>	Indicates that the function successfully converted the number to a string.
<b>&lt; 0</b>	Indicates that the function failed to convert the number to a string.

## NumToStr Example

To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the NumToStr function
 *
 * This script calls NumToStr convert the numeric value of free
 * disk space available on the system to a string so that it
 * can be displayed by the MessageBox function.
 *
 \*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_NumToStr(HWND);

function ExFn_NumToStr(hMSI)
    STRING svString;
    NUMBER nSpace, nResult;
begin

    // Get the amount of free space on drive C.
    nSpace = GetDiskSpace ("C:");

    // Convert the number to a string.
    nResult = NumToStr (svString, nSpace);

```

```

if (nResult < 0) then
    MsgBox ("NumToStr failed.", SEVERE);
else
    // Display the amount of free space on drive C.
    MsgBox (svString + " bytes free on drive C.", INFORMATION);
endif;

end;

```

## OpenFile

The **OpenFile** function opens an existing text file or binary file. Before you open the file you must set the file mode by calling [OpenFileMode](#).



**Tip** • Note the following:

- After you open a text file, call *GetLine* and *WriteLine* to read from and write to the file. When you finish reading from or writing to a file with *GetLine* or *WriteLine*, you must close the file using the *CloseFile* function.
- After you open a binary file, call *ReadBytes* and *WriteBytes* to read from and write to the file.
- You can use *SeekBytes* to position the file pointer before writing to a binary file.
- You can also search, read from, and write to text files using the *FileGrep*, *FileInsertLine*, and *FileDeleteLine* functions. However, these functions do not require you to open or close the files (this is handled internally).
- Use *CreateFile* to create a file. *CreateFile* leaves the new file open in append mode (text files) or read/write mode (binary files).

### Syntax

```
OpenFile ( nvFileHandle, szPath, szFileName );
```

## Parameters

**Table 111** • OpenFile Parameters

Parameter	Description
<b>nvFileHandle</b>	Returns the file handle of the file that was opened. Use this handle to identify the file when you call other file-related InstallScript functions.
<b>szPath</b>	Specifies the path—which may include a drive designation—of the file you want to open. You can specify a valid Uniform Resource Locator (URL) in this parameter, after setting the file mode to FILE_MODE_NORMAL or FILE_MODE_BINARYREADONLY (by calling OpenFileMode). If you pass a CGI or ASP request (for example, "http://www.mydomain.net/login.asp?name=Me&pwd=wow"), the response is sent to memory and can be read by ReadBytes. To check the validity of a URL, call Is(VALID_PATH, szURL). If you pass a null string ("") in this parameter, the function fails.
<b>szFileName</b>	Specifies the unqualified name—that is, without a drive designation or path—of the file you want to open. If you pass a null string ("") in this parameter, the function fails.

## Return Values

**Table 112** • OpenFile Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully opened the file.
<b>&lt; 0</b>	Indicates that the function was unable to open the file.

## OpenFile Example

To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the OpenFile and CloseFile functions.
*
* OpenFile is called to open a file, which is then read into
* a list. The file is then closed and the list is displayed.
*
* Note: Before running this script, set the preprocessor
*       constants so that they reference an existing file
*       in an existing directory.
*
*-----*/
```

```

#define EXAMPLE_FILE "Readme.txt"
#define EXAMPLE_DIR "C:\\Windows"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_OpenFile(HWND);

function ExFn_OpenFile(hMSI)
    STRING  svLine;
    NUMBER  nvFileHandle;
    LIST     listID;
begin

    // Set the file mode to normal.
    OpenFileMode (FILE_MODE_NORMAL);

    // Open the text file.
    if (OpenFile (nvFileHandle, EXAMPLE_DIR, EXAMPLE_FILE) < 0) then
        MessageBox ("OpenFile failed.", SEVERE);
        abort;
    endif;

    // Create an empty string list.
    listID = ListCreate (STRINGLIST);

    // Read lines from the text file into the string list.
    while GetLine (nvFileHandle, svLine) = 0
        ListAddString (listID, svLine, AFTER);
    endwhile;

    // Close the file.
    if (CloseFile (nvFileHandle) < 0) then
        MessageBox ("CloseFile failed.", SEVERE);
    endif;

    // Display the text that was read from the file.
    SdShowInfoList ("", "", listID);

end;

```

## OpenFileMode

The **OpenFileMode** function sets the mode of the file you want to open or create. The argument you pass as the parameter `nMode` sets the file mode to one of the following:

- ANSI or Unicode text file in append mode.
- ANSI or Unicode text file in read-only mode.
- Binary file in read/write mode.
- Binary file in read-only mode.

After you set the file mode, call [OpenFile](#) to open an existing file or [CreateFile](#) to create and open a new file.

## Syntax

```
OpenFileMode ( nMode );
```



## Parameters

**Table 113 • OpenFileMode Parameters**

Parameter	Description
<b>nMode</b>	<p>Specifies which mode to use to open a file. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li> <b>FILE_MODE_APPEND</b>—This constant allows a text file to be opened or created in append mode. When a file is opened in append mode using <b>OpenFile</b>, the file pointer is at the end of the file. You can use the <b>WriteLine</b> function to append lines to the end of the file. Files created using <b>CreateFile</b> are new (empty), so lines appended to the files are written at the beginning of the files. Note that if you open a file in append mode, the <b>GetLine</b> function will fail if you call it because the file pointer is at the end of the file. </li> <li> <b>FILE_MODE_APPEND_UNICODE</b>—Indicates that <b>CreateFile</b> should create new files as Unicode files, instead of the default of ANSI. This option does not affect how existing files are opened. Existing files are always opened and saved in the existing format. To convert a file from one type to another, use the <b>ListReadFromFile</b> and <b>ListWriteToFileEx</b> with the appropriate nOptions. </li> <li> <b>FILE_MODE_NORMAL</b>—This constant allows a text file to be opened in read-only mode. When a file is opened in read-only mode using <b>OpenFile</b>, the file pointer is at the beginning of the file. You can use the <b>GetLine</b> function to read from the file. Files created using <b>CreateFile</b> when <b>FILE_MODE_NORMAL</b> is in effect will actually be created in <b>FILE_MODE_APPEND</b> mode. </li> <li> <b>FILE_MODE_BINARY</b>—This constant allows a binary file to be opened and created in read/write mode. When you open or create a file in binary mode with <b>OpenFile</b> or <b>CreateFile</b>, you can call <b>ReadBytes</b> to read from the file and <b>WriteBytes</b> to write to the file. Writing to a binary file begins at the current file pointer position, which for a newly opened or created file is position 0—the beginning of the file. If you want to append to an existing binary file opened using <b>OpenFile</b>, you must use <b>SeekBytes</b> to position the file pointer before writing. To open a file on a CD-ROM or on a read-only drive, call <b>OpenFileMode</b> to set the file mode to read-only (<b>FILE_MODE_BINARYREADONLY</b>). </li> <li> <b>FILE_MODE_BINARYREADONLY</b>—This constant is just like the constant <b>FILE_MODE_BINARY</b>, except that it opens the binary file in read-only mode. When opening a binary file on CD-ROM or read-only drives, use this constant to open a binary file. <b>FILE_MODE_BINARY</b> will fail opening binary files on CD-ROM or read-only drives. </li> </ul>

## Return Values

**Table 114 • OpenFileMode Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully set the file mode.

**Table 114** • OpenFileMode Return Values (cont.)

Return Value	Description
< 0	Indicates that the function was unable to set the file mode.

## OpenFileMode Example

To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the OpenFileMode function.
 *
 * This script opens a text file in read-only (FILE_MODE_NORMAL)
 * mode. It then retrieves and displays the first line from
 * the file.
 *
 * Next, it opens a file in binary mode, positions the file
 * pointer at the 15th byte and reads 28 bytes from the file.
 *
 * Note: In order for this script to run correctly, you must set
 *       the preprocessor constants so that they reference
 *       existing files in an existing directory.
 *
\*-----*/

#define EXAMPLE_DIR      "C:\\\"
#define EXAMPLE_TEXT_FILE "ISExempl.txt"
#define EXAMPLE_BIN_FILE  "ISExempl.bin"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_OpenFileMode(HWND);

function ExFn_OpenFileMode(hMSI)
    STRING svLine, svString;
    NUMBER nvFileHandle;
begin
    // Set the file mode to normal.
    OpenFileMode (FILE_MODE_NORMAL);

    // Open the file.
    if (OpenFile (nvFileHandle, EXAMPLE_DIR, EXAMPLE_TEXT_FILE) < 0) then
        MessageBox ("OpenFile failed.", SEVERE);
        abort;
    endif;

    // Get the first line of the text file.
    GetLine (nvFileHandle, svLine);

    // Display the line.

```

```

MessageBox (svLine, INFORMATION);

// Close the file.
CloseFile (nvFileHandle);

// Set the file mode to binary read/write.
OpenFileMode (FILE_MODE_BINARY);

// Open the file.
if (OpenFile (nvFileHandle, EXAMPLE_DIR, EXAMPLE_BIN_FILE) < 0) then
    MessageBox ("OpenFile failed.", SEVERE);
else
    // Move the file pointer to byte 15.
    SeekBytes (nvFileHandle, 15, FILE_BIN_START);

    // Read 28 bytes from the binary file into svString.
    if (ReadBytes (nvFileHandle, svString, 0, 28) < 0) then
        MessageBox ("ReadBytes failed.", SEVERE);
    else
        // Display the string.
        MessageBox (svString, INFORMATION);
    endif;

    // Close the binary file.
    CloseFile (nvFileHandle);
endif;

end;

```

## ParsePath

The **ParsePath** function retrieves the specified part of an existing path. The function works with any valid path, including short paths, long paths, and UNC paths that may or may not include a specific file name. These are some sample paths that can be parsed with this function.

- \Path1\Path2\Filename.exe
- FileName
- Filename.exe
- \Path1\Path2\Filename
- D:
- D:\
- \\Server Name\Share Name\Share Directory
- Any other legal DOS path

### Syntax

```
ParsePath ( svReturnString, szPath, nOperation );
```

## Parameters

**Table 115 • ParsePath Parameters**

Parameter	Description
<b>svReturnString</b>	Returns the part of the path in szPath that is specified by nOperation.
<b>szPath</b>	Specifies the path to parse. When specifying a path that does not include a file name, you must append a backslash to the end of the path before passing it to ParsePath; otherwise the last part of the path will be interpreted as a file name.
<b>nOperation</b>	<p>Specifies which element of the path to return. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>DIRECTORY</b>—Indicates that the path minus the disk drive letter and file name should be returned in svReturnString. When this option is used with a UNC path, ParsePath returns the path without the server and shared device name, and without the file name if one was specified. For example, the UNC path \\TheServer\TheSharedDevice\TheApp\TheFile.exe is returned in svReturnString as \TheApp\.</li> <li>• <b>DISK</b>—Indicates that the disk drive designation (drive letter followed by a colon) should be returned in svReturnString. When this option is used with a UNC path, ParsePath returns the server and shared device name. For example, the UNC path \\TheServer\TheSharedDevice\TheApp\TheFile.exe is returned in svReturnString as \\TheServer\TheSharedDevice.</li> <li>• <b>EXTENSION_ONLY</b>—Indicates that the file extension should be returned in svReturnString. It does not include the period.</li> <li>• <b>FILENAME</b>—Indicates that the complete file name (that is, with its file extension) should be returned in svReturnString.</li> <li>• <b>FILENAME_ONLY</b>—Indicates that the file name only (that is, without its file extension) should be returned in svReturnString.</li> <li>• <b>PATH</b>—Indicates that the path minus the file name should be returned in svReturnString. This option differs from DIRECTORY in that the drive designation (if specified in szPath) is included in the returned path. When szPath specifies a UNC path the server and shared device name are included in the returned path. For example, the UNC path \\TheServer\TheSharedDevice\TheApp\TheFile.exe is returned in svReturnString as \\TheServer\TheSharedDevice\TheApp\.</li> </ul>

## Return Values

**Table 116 • ParsePath Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully parsed the path string.
<b>&lt; 0</b>	Indicates that the function was unable to parse the path string.

## ParsePath Example

To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the ParsePath function.
 *
 * The ParsePath is called six times to retrieve various
 * information from a fully-qualified file name.
 *
 \*-----*/

#define EXAMPLE_PATH "C:\\Windows\\Readme.txt"
#define TITLE_TEXT   "ParsePath example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ParsePath(HWND);

function ExFn_ParsePath(hMSI)
    STRING szMsg, svReturnString;
begin

    // Get the disk letter from the path.
    if (ParsePath (svReturnString, EXAMPLE_PATH, DISK) < 0) then
        MessageBox ("ParsePath failed", SEVERE);
    else
        szMsg = "nOperation = DISK\\n\\nParsed Path: %s";
        sprintfBox (INFORMATION, TITLE_TEXT , szMsg, svReturnString);
    endif;

    // Get the full path.
    szMsg = "nOperation = PATH\\n\\nParsed Path: %s";

    if (ParsePath (svReturnString, EXAMPLE_PATH, PATH) < 0) then
        MessageBox ("ParsePath failed", SEVERE);
    else
        sprintfBox (INFORMATION, TITLE_TEXT, szMsg, svReturnString);
    endif;

    // Get the directory name.
    if (ParsePath (svReturnString, EXAMPLE_PATH, DIRECTORY) < 0) then
        MessageBox ("ParsePath failed", SEVERE);
    else
        szMsg = "nOperation = DIRECTORY\\n\\nParsed Path: %s";
        sprintfBox (INFORMATION, TITLE_TEXT, szMsg, svReturnString);
    endif;

    // Get the file name and extension.
    if (ParsePath (svReturnString, EXAMPLE_PATH, FILENAME) < 0) then
        MessageBox ("ParsePath failed", SEVERE);
    else
        szMsg = "nOperation = FILENAME\\n\\nParsed Path: %s";
        sprintfBox (INFORMATION, TITLE_TEXT, szMsg, svReturnString);
    endif;
endfunction
```

```

endif;

// Get file name without extension.
if (ParsePath (svReturnString, EXAMPLE_PATH, FILENAME_ONLY) < 0) then
    MessageBox ("ParsePath failed", SEVERE);
else
    szMsg      = "nOperation = FILE_NAME_ONLY\n\nParsed Path: %s";
    sprintfBox (INFORMATION, TITLE_TEXT, szMsg, svReturnString);
endif;

// Get the file extension.
if (ParsePath (svReturnString, EXAMPLE_PATH, EXTENSION_ONLY) < 0) then
    MessageBox ("ParsePath failed", SEVERE);
else
    szMsg      = "nOperation = EXTENSION_ONLY\n\nParsed Path: %s";
    sprintfBox (INFORMATION, TITLE_TEXT, szMsg, svReturnString);
endif;

end;

```

## ParseUrl

The **ParseUrl** function retrieves the parts of the specified URL. The function works with any valid URL.

### Syntax

```
ParseUrl ( szUrl, pISUrlComponents );
```

## Parameters

**Table 117** • ParseUrl Parameters

Parameter	Description
<b>szUrl</b>	Specifies the URL to parse.
<b>pISUrlComponents</b>	Returns a pointer to an ISURL_COMPONENTS data structure that contains the parts of the specified URL.

## Return Values

**Table 118** • ParseUrl Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully parsed the path string.
<b>&lt; 0</b>	Indicates that the function was unable to parse the path string.

## ParseUrl Example

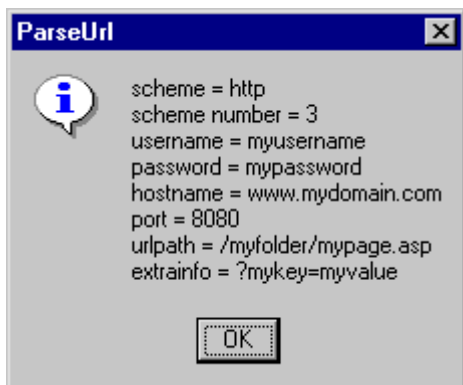
```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the ParseUrl function.
 *
 \*-----*/

function OnBegin()
    ISURL_COMPONENTS ISUrlComponents;
    STRING szUrl;
begin
    szUrl =
        "http://myusername:mypassword@www.mydomain.com:8080/" +
        "myfolder/mypage.asp?mykey=myvalue";
    ParseUrl ( szUrl, &ISUrlComponents );
    sprintfBox ( INFORMATION , "ParseUrl" ,
        "scheme = %s\nscheme number = %ld\nusername = %s\npassword = %s" +
        "\nhostname = %s\nport = %ld\nurlpath = %s\nextraInfo = %s",
        ISUrlComponents.szScheme, ISUrlComponents.nInternetScheme,
        /* nInternetScheme equals 3 for HTTP and 4 for HTTPS. */
        ISUrlComponents.szUserName, ISUrlComponents.szPassword,
        ISUrlComponents.szHostName, ISUrlComponents.nInternetPort,
        ISUrlComponents.szUrlPath, ISUrlComponents.szExtraInfo );
    abort;
end;

```

That script produces the following output:



**Figure 1:** ParseUrl Output

## PathAdd

The **PathAdd** function adds a path to the search path in the path buffer. With this function you can specify the position of the directory in relation to an existing directory in the path buffer. In addition, you can add the directory as the first or the last directory of the path buffer.

This function has no relationship to the path statement in the Autoexec.bat file or the path environment variable. It acts only on the path buffer, which helps you build, modify, and manipulate search paths. You can then add the modified path string to the Autoexec.bat file using the various batch file functions.


### Syntax

```
PathAdd ( szDir, szRefDir, bRefDir, bPosition );
```



## Parameters

**Table 119** • PathAdd Parameters

Parameter	Description
<b>szDir</b>	Specifies a path to add to the path buffer.   <b>Note</b> • If you specify a directory in szDir that currently exists in the path buffer, InstallShield does not duplicate it, and the position of the pre-existing directory is not modified. InstallShield ignores a backslash at the end of the directory name.
<b>szRefDir</b>	Specifies the path in the current path buffer relative to which the new path will be added.
<b>bRefDir</b>	Specifies whether or not szRefDir is a fully qualified path. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li><b>FULL</b>—szRefDir is a fully qualified path—it includes a drive designation and the complete path.</li> <li><b>PARTIAL</b>—szRefDir is the directory name only—without drive or path information.</li> </ul>
<b>bPosition</b>	Specifies the position relative to szRefDir at which szDir is to be inserted. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li><b>AFTER</b>—Specifies that szDir is to be inserted after szRefDir. If szRefDir specifies a null string (""), szDir is inserted at the end of the path in the path buffer.</li> <li><b>BEFORE</b>—Specifies that szDir is to be inserted before szRefDir. If szRefDir specifies a null string (""), szDir is inserted at the front of the path in the path buffer.</li> </ul>

## Return Values

**Table 120** • PathAdd Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully added a directory to the path buffer.
<b>&lt; 0</b>	Indicates that the function was unable to add a directory to the path buffer.

## PathAdd Example

To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the PathAdd function.
*
```

```

* This example script shows how to add paths to a search path
* in the path buffer. It begins by initializing the path buffer
* with a search path. Then, it adds a new path to the front of
* the search path. Next, it gets the modified search path from
* the search buffer and displays it.
*
* It then initializes the path buffer with the modified search
* path and adds a new path before the last path in the path
* buffer. Finally, it gets the modified search path from the
* search buffer and displays it.
*
* Note: PathGet is called after the first PathAdd statement
*       only so the value of the path buffer can be displayed
*       for demonstration purposes. In a production script,
*       you would finish building the search path in the buffer
*       before calling PathGet.
*
\*-----*/

#define TITLE "Path buffer example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_PathAdd(HWND);

function ExFn_PathAdd(hMSI)
    STRING svSearchPath;
begin

    // Set up the search path to pass as a parameter to PathSet.
    svSearchPath = "C:\\DOS;C:\\Windows;C:\\Temp";

    // Initialize the path buffer.
    PathSet (svSearchPath);

    // Display the initial search path.
    sprintfBox (INFORMATION,TITLE,
        "The starting search path is %s.",svSearchPath);

    // Add C:\\MSOffice as the first path in the search path.
    if (PathAdd("C:\\MSOffice", "", FULL, BEFORE) < 0) then
        MessageBox ("Unable to add C:\\MsOffice to path buffer.", SEVERE);
        abort;
    endif;

    // Get the search path from the path buffer; this call also releases
    // the memory allocated for the path buffer.
    PathGet (svSearchPath);

    // Display the search path.
    // svSearchPath will contain C:\\MSOffice;C:\\DOS;C:\\Windows;C:\\Temp.
    sprintfBox (INFORMATION,TITLE,
        "C:\\MSOffice added before first path.\\n\\nThe search path is %s.",
        svSearchPath);

    // Initialize path buffer to hold the search path.
    PathSet (svSearchPath);

```

```

// Add C:\APP2 before C:\Temp in the path buffer.
if (PathAdd ("C:\\APP2", "Temp", PARTIAL, BEFORE) < 0) then
    MessageBox ("Unable to add C:\\APP2 to path buffer.", SEVERE);
    abort;
endif;

// Get the search path from the path buffer; this call also releases
// the memory allocated for the path buffer.
PathGet (svSearchPath);

// Display the modified search path.
// svSearchPath will contain C:\MSOffice;C:\DOS;C:\Windows;C:\App2;C:\Temp.
sprintfBox (INFORMATION, TITLE,
    "C:\\APP2 added before C:\\Temp.\n\nThe search path is %s.",
    svSearchPath);

end;

```

## PathDelete

The **PathDelete** function deletes a specific directory in the path buffer. You can specify the name of the directory or enter a fully qualified path.

This function has no relationship to the path statement in the Autoexec.bat file or the path environment variable. It acts only on the path buffer, which helps you build, modify, and manipulate search paths.




---

**Tip** • Call [PathGet](#) to get the contents of the path buffer. Call [PathSet](#) to set contents of the path buffer.

### Syntax

```
PathDelete ( szDir, bDir );
```

## Parameters

**Table 121** • PathDelete Parameters

Parameter	Description
<b>szDir</b>	Specifies the path to remove from the path buffer.
<b>bDir</b>	Specifies whether or not szRefDir is a fully qualified path. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li><b>FULL</b>—szRefDir is a fully qualified path—it includes a drive designation and the complete path.</li> <li><b>PARTIAL</b>—szRefDir is the directory name only—without drive or path information.</li> </ul>

## Return Values

**Table 122** • PathDelete Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully deleted a directory from the path buffer.
<b>&lt; 0</b>	Indicates that the function was unable to delete a directory from the path buffer.

## PathDelete Example

To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the PathDelete function.
*
* This example script shows how to delete paths from the path
* buffer. It begins by initializing the path buffer with a
* search path. Then, it deletes any path that includes a
* reference to "Temp". Next, it gets the modified search path
* from the search buffer and displays it.
*
* It then initializes the path buffer with the modified search
* path and deletes a specific path. Finally, it gets the
* search path from the search buffer and displays it.
*
* Note: PathGet is called after the first PathDelete statement
* only so the value of the path buffer can be displayed
* for demonstration purposes. In a production script,
* you would finish building the search path in the buffer
* before calling PathGet.
```

```

*
\*-----*/

#define TITLE "Path buffer example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_PathDelete(HWND);

function ExFn_PathDelete(hMSI)
    STRING svSearchPath;
begin

    // Set up the search path to pass as a parameter to PathSet.
    svSearchPath = "C:\\DOS;C:\\WINDOWS;C:\\TEMP;" +
        "C:\\EXAMPLE\\SOURCE;D:\\WORK\\TEMP";

    // Initialize the path buffer.
    PathSet (svSearchPath);

    // Display the initial search path.
    sprintfBox (INFORMATION,TITLE,
        "The starting search path is %s.",svSearchPath);

    // Delete C:\\Temp from the path buffer.
    if (PathDelete ("TEMP", PARTIAL) < 0) then
        MessageBox ("First call to PathDelete failed.", SEVERE);
    endif;

    // Get the search path from the path buffer; this call also releases
    // the memory allocated for the path buffer.
    PathGet (svSearchPath);

    // Display the search path.
    // svSearchPath will contain C:\\DOS;C:\\WINDOWS;C:\\EXAMPLE\\SOURCE.
    sprintfBox (INFORMATION, TITLE,
        "All paths referencing 'Temp' were deleted.\\n\\nThe search path is %s.",
        svSearchPath);

    // Set up the path buffer again.
    PathSet (svSearchPath);

    // Delete C:\\EXAMPLE\\SOURCE from the path buffer.
    if (PathDelete ("C:\\EXAMPLE\\SOURCE", FULL) < 0) then
        MessageBox ("Second call to PathDelete failed.",SEVERE);
    endif;

    // Get the search path from the path buffer; this call also releases
    // the memory allocated for the path buffer.
    PathGet (svSearchPath);

    // Display the search path.
    // svSearchPath will contain C:\\DOS;C:\\WINDOWS.
    sprintfBox (INFORMATION, TITLE,
        "C:\\EXAMPLE\\SOURCE was deleted.\\n\\nPath is %s.",
        svSearchPath);

end;

```

# PathFind

The **PathFind** function searches the path buffer for a specific directory. You can specify the directory with either a fully qualified path or the directory name only.

This function has no relationship to the path statement in the Autoexec.bat file or the path environment variable. It acts only on the path buffer, which helps you build, modify, and manipulate search paths. You can then add this temporary path string to the Autoexec.bat file using the various batch file functions.

## Syntax

```
PathFind ( szDir, svResult, bDir, bSearch );
```

## Parameters

**Table 123 • PathFind Parameters**

Parameter	Description
<b>szDir</b>	Specifies the path to find in the path buffer.
<b>svResult</b>	Returns the full directory and path found in the path buffer returned by the function.
<b>bDir</b>	Specifies whether or not szDir contains a fully qualified or an unqualified directory name. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li><b>FULL</b>—szRefDir is a fully qualified path—it includes a drive designation and the complete path to a directory.</li> <li><b>PARTIAL</b>—szRefDir is the directory name only, without drive or path information.</li> </ul>
<b>bSearch</b>	Specifies where to begin the search. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li><b>CONTINUE</b>—Continues searching the path buffer at the location where the previous search was terminated.</li> <li><b>RESTART</b>—Starts the search from the beginning of the path buffer.</li> </ul>

## Return Values

**Table 124 • PathFind Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully searched the path buffer for a directory.
<b>&lt; 0</b>	Indicates that the function was unable to successfully search the path buffer for a directory.

## PathFind Example

To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the PathSet, PathFind, and PathGet functions.
*
* First, PathSet is called to place a search path into the path
* buffer. Then PathFind is then called to search the path
* buffer for instances of a specific path. Finally, PathGet
* is called to return the contents of the path buffer.
*
*-----*/
```

```

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_PathFind(HWND);

function ExFn_PathFind(hMSI)
    STRING szString, szMsg, svResult, svString, szDir;
    NUMBER nResult;
    BOOL    bDir, bSearch;
begin
    // Set up the search path to pass as a parameter to PathSet.
    szString = "C:\\DOS;C:\\USERS\\BIN;C:\\MSC\\BIN;";

    // Place the search path into the path buffer
    if (PathSet (szString) < 0) then
        // Report an error; then terminate.
        MessageBox ("PathSet failed.", SEVERE);
        abort;
    else
        szMsg = "PathSet set the path buffer to: %s";
        sprintfBox (INFORMATION, "PathSet Example", szMsg, szString);
    endif;

    // Set PathFind variables.
    szDir = "BIN";

    // Search the path buffer for paths that include a folder named "BIN".
    nResult = PathFind(szDir, svResult, PARTIAL, RESTART);

    // Error check PathFind.
    if (nResult < 0) then
        MessageBox("PathFind failed.", SEVERE);
        abort;
    endif;

    // Loop through the string to find all occurrences of the szDir string.

    while (nResult = 0)
        sprintfBox(INFORMATION, "PathFind example",
            "Search for %s.\n\nFound: %s", szDir, svResult);
        nResult = PathFind(szDir, svResult, PARTIAL, CONTINUE);
    endwhile;

    // Get the contents of the path buffer.
    if (PathGet (svString) < 0) then
        MessageBox ("PathGet failed.", SEVERE);
    else
        // Display the path string.
        sprintfBox (INFORMATION, "Path Get Example", "Path is: %s", svString);
    endif;
end;

```



# PathGet

The **PathGet** function retrieves the search path currently stored in the path buffer, which is a temporary storage area created by a call to PathSet. The path buffer enables you to build and edit a search path. When the path you are editing is complete, call PathGet to place the search path into a string variable so that you can pass it to other functions in your setup.

This function has no relationship to the path statement in the Autoexec.bat file or the path environment variable. It acts on the path buffer only, which helps you build, modify, and manipulate search paths. You can then add this temporary path string to the Autoexec.bat file using the appropriate batch file functions.



**Tip** • PathGet retrieves the search path from the path buffer and releases the memory allocated to the path buffer. A subsequent call to PathGet will fail unless the path buffer is reinitialized by a call to [PathSet](#).

## Syntax

```
PathGet ( svString );
```

## Parameters

**Table 125** • PathGet Parameters

Parameter	Description
<b>svString</b>	Returns the contents of the path buffer.

## Return Values

**Table 126** • PathGet Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully retrieved the path currently stored in the path buffer.
<b>&lt; 0</b>	Indicates that the function was unable to retrieve the path string currently stored in the temporary path string buffer.

## PathGet Example

To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the PathSet, PathFind, and PathGet functions.
*
* First, PathSet is called to place a search path into the path
```

```

* buffer. Then PathFind is then called to search the path
* buffer for instances of a specific path. Finally, PathGet
* is called to return the contents of the path buffer.
*
\*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_PathGet(HWND);

function ExFn_PathGet(hMSI)
    STRING szString, szMsg, svResult, svString, szDir;
    NUMBER nResult;
    BOOL bDir, bSearch;
begin

    // Set up the search path to pass as a parameter to PathSet.
    szString = "C:\\DOS;C:\\USERS\\BIN;C:\\MSC\\BIN;";

    // Place the search path into the path buffer
    if (PathSet (szString) < 0) then
        // Report an error; then terminate.
        MessageBox ("PathSet failed.", SEVERE);
        abort;
    else
        szMsg = "PathSet set the path buffer to: %s";
        sprintfBox (INFORMATION, "PathSet Example", szMsg, szString);
    endif;

    // Set PathFind variables.
    szDir = "BIN";

    // Search the path buffer for paths that include a folder named "BIN".
    nResult = PathFind(szDir, svResult, PARTIAL, RESTART);

    // Error check PathFind.
    if (nResult < 0) then
        MessageBox("PathFind failed.", SEVERE);
        abort;
    endif;

    // Loop through the string to find all occurrences of the szDir string.
    while (nResult = 0)
        sprintfBox(INFORMATION, "PathFind example",
            "Search for %s.\n\nFound: %s", szDir, svResult);
        nResult = PathFind(szDir, svResult, PARTIAL, CONTINUE);
    endwhile;

    // Get the contents of the path buffer.
    if (PathGet (svString) < 0) then
        MessageBox ("PathGet failed.", SEVERE);
    else
        // Display the path string.
        sprintfBox (INFORMATION, "Path Get Example", "Path is: %s", svString);
    endif;

end;

```

# PathMove

The **PathMove** function repositions a directory in the path buffer to another location. You can also use this function to position the directory relative to another directory or as the first or the last item in the path string.

This function has no relation to the PATH statement in the Autoexec.bat file or the PATH environment variable. It acts only on the path buffer, which helps you build, modify, and manipulate search paths. You can then add this temporary path string to Autoexec.bat using the various batch file functions.



---

**Tip** • Call [PathGet](#) to get the contents of the path buffer. Call [PathSet](#) to set contents of the path buffer.

## Syntax

```
PathMove ( szDir, szRefDir, bDir, bRefDir, bPosition );
```

## Parameters

**Table 127** • PathMove Parameters

Parameter	Description
<b>szDir</b>	Specifies a full or partial path to reposition in the path buffer.
<b>szRefDir</b>	Specifies the path in the path buffer relative to which the path in szDir will be moved. To move the path in szDir to the beginning or end of the path in the path buffer, pass a null string ("") in this parameter.
<b>bDir</b>	Specifies whether szDir contains a fully qualified or an unqualified directory name. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>● <b>FULL</b>—Specifies that szDir contains a fully qualified directory name.</li> <li>● <b>PARTIAL</b>—Specifies that szDir contains the directory name only.</li> </ul>
<b>bRefDir</b>	Specifies whether szRefDir contains a fully qualified or an unqualified directory name. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>● <b>FULL</b>—Specifies that szRefDir contains a fully qualified directory name.</li> <li>● <b>PARTIAL</b>—Specifies that szRefDir contains the directory name only.</li> </ul>
<b>bPosition</b>	Specifies the position relative to szRefDir to which szDir is to be moved. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>● <b>AFTER</b>—Specifies that szDir is to be positioned after szRefDir. If szRefDir specifies a null string (""), szDir is positioned at the end of the path in the path buffer.</li> <li>● <b>BEFORE</b>—Specifies that szDir is to be positioned before szRefDir. If szRefDir specifies a null string (""), szDir is positioned at the front of the path in the path buffer.</li> </ul>

## Return Values

**Table 128** • PathMove Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully repositioned a directory in the path buffer.
<b>&lt; 0</b>	Indicates that the function was unable to reposition a directory in the path buffer.

## PathMove Example

To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the PathMove functions.
 *
 * This example script shows how to reposition paths in the path
 * buffer. It begins by initializing the path buffer with a
 * search path. Then, it moves one path ahead of another.
 * Next, it gets the modified search path from the search buffer
 * and displays it.
 *
 * It then initializes the path buffer with the modified search
 * path and moves one path after another path. Finally, it gets
 * the modified search path from the search buffer and displays
 * it.
 *
 * Note: PathGet is called after the first PathMove statement
 *       only so the value of the path buffer can be displayed
 *       for demonstration purposes. In a production script,
 *       you would finish building the search path in the buffer
 *       before calling PathGet.
 *
\*-----*/

#define TITLE "Path buffer example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_PathMove(HWND);

function ExFn_PathMove(hMSI)
    STRING svSearchPath;
begin

    // Set up the search path to pass as a parameter to PathSet.
    svSearchPath = "C:\\MsOffice;C:\\DOS;C:\\Windows;C:\\App2;C:\\Temp";

    // Initialize the path buffer.
    PathSet (svSearchPath);

    // Display the initial search path.
    sprintfBox (INFORMATION, TITLE,
        "The starting search path is %s.",svSearchPath);

    // Move C:\\App2 before C:\\DOS.
    if (PathMove("C:\\App2", "C:\\DOS", FULL, FULL, BEFORE) < 0) then
        MessageBox ("Unable to move C:\\App2.", SEVERE);
        abort;
    endif;

    // Get the search path from the path buffer and release
    // the memory allocated for the path buffer.
    PathGet (svSearchPath);

    // Display the search path.
    // svSearchPath will contain C:\\MSOFFICE;C:\\APP2;C:\\DOS;C:\\WINDOWS;C:\\TEMP
    sprintfBox (INFORMATION, TITLE,

```

```

        "C:\\App2 moved before C:\\DOS.\\n\\nThe search path is %s.",
        svSearchPath);

// Initialize the path buffer with the search path.
PathSet (svSearchPath);

// Move C:\\DOS after C:\\Temp.
if (PathMove ("C:\\DOS", "TEMP", FULL, PARTIAL, AFTER) < 0) then
    MessageBox ("Unable to move C:\\DOS.", SEVERE);
endif;

// Get the search path from the path buffer and release
// the memory allocated for the path buffer.
PathGet (svSearchPath);

// Display the search path.
// svSearchPath will contain C:\\MSOFFICE;C:\\APP2;C:\\WINDOWS;C:\\TEMP;C:\\DOS
sprintfBox (INFORMATION, TITLE,
    "C:\\DOS moved after C:\\Temp.\\n\\nThe search path is %s.",
    svSearchPath);

end;

```

## PathSet

The **PathSet** function stores a search path string in the path buffer. You can then manipulate this buffer using the other path functions. The value of `szString` should be an absolute path (a path that includes a drive specification, for example, "C:\\Program Files\\AppName").

This function has no relation to the PATH statement in the Autoexec.bat file or the PATH environment variable. It acts only on the path buffer, which helps you build, modify, and manipulate search paths. You can then add this temporary path string to the Autoexec.bat file or PATH environment variable.

### Syntax

```
PathSet ( szString );
```

## Parameters

**Table 129** • PathSet Parameters

Parameter	Description
<b>szString</b>	Specifies a search path to store in the path buffer. The search path should be fully qualified; that is it should include a drive designation and the complete path to a directory.

## Return Values

**Table 130** • PathSet Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully stored a search path string in the path buffer.
<b>&lt; 0</b>	Indicates that the function was unable to store a search path string in the path buffer.

## PathSet Example

To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the PathSet, PathFind, and PathGet functions.
*
* First, PathSet is called to place a search path into the path
* buffer. Then PathFind is then called to search the path
* buffer for instances of a specific path. Finally, PathGet
* is called to return the contents of the path buffer.
*
\*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_PathSet(HWND);

function ExFn_PathSet(hMSI)
    STRING szString, szMsg, svResult, svString, szDir;
    NUMBER nResult;
    BOOL bDir, bSearch;
begin

    // Set up the search path to pass as a parameter to PathSet.
    szString = "C:\\DOS;C:\\USERS\\BIN;C:\\MSC\\BIN;";
  
```

```

// Place the search path into the path buffer
if (PathSet (szString) < 0) then
    // Report an error; then terminate.
    MessageBox ("PathSet failed.", SEVERE);
    abort;
else
    szMsg = "PathSet set the path buffer to: %s";
    sprintfBox (INFORMATION, "PathSet Example", szMsg, szString);
endif;

// Set PathFind variables.
szDir = "BIN";

// Search the path buffer for paths that include a folder named "BIN".
nResult = PathFind(szDir, svResult, PARTIAL, RESTART);

// Error check PathFind.
if (nResult < 0) then
    MessageBox("PathFind failed.", SEVERE);
    abort;
endif;

// Loop through the string to find all occurrences of the szDir string.
while (nResult = 0)
    sprintfBox(INFORMATION, "PathFind example",
        "Search for %s.\n\nFound: %s", szDir, svResult);
    nResult = PathFind(szDir, svResult, PARTIAL, CONTINUE);
endwhile;

// Get the contents of the path buffer.
if (PathGet (svString) < 0) then
    MessageBox ("PathGet failed.", SEVERE);
else
    // Display the path string.
    sprintfBox (INFORMATION, "Path Get Example", "Path is: %s", svString);
endif;

end;

```

## PlaceBitmap



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **PlaceBitmap** function inserts an image into the installation window. The image source is specified by szName; it can be a bitmap file (.bmp), metafile (.wmf file), or dynamic link library (.dll).

### Syntax

```
PlaceBitmap ( szName, nID_BITMAP, nDx, nDy, nDrawOp );
```




## Parameters

**Table 131** • PlaceBitmap Parameters

Parameter	Description
<b>szName</b>	<p>Specifies the fully qualified name of the bitmap file (.bmp), metafile (.wmf file), or dynamic link library (.dll) of the image to be displayed. InstallShield recognizes bitmap files and metafiles by their file extension. Bitmap files must have the extension .bmp. Metafiles must have the extension .wmf. Dynamic link libraries must have the extension .dll. If a file name is specified with no extension, InstallShield assumes the extension .dll.</p> <p>To specify an alternate transparent color, place a semicolon after the file name and follow it with a set of RGB color values. (RGB color is specified with three numeric values, separated by commas.) That color is then used as the transparent color for the bitmap specified by szName. Note that it does <i>not</i> affect bitmaps that are already displayed; nor does it become the default transparent color for bitmaps displayed by subsequent calls to PlaceBitmap.</p> <p>The parameter below specifies white as the transparent color:</p> <p><code>SUPPORTDIR ^ "Bitmap.bmp;255,255,255"</code> When nDrawOptions is set to REMOVE, this parameter is ignored.</p>
<b>nID_BITMAP</b>	<p>Specifies the bitmap's resource ID if the bitmap resides in a .dll. If the bitmap source is a metafile or bitmap file, specifies a value that is not in use for an image currently on display; images that are displayed simultaneously must have unique ID numbers. When nDrawOptions is set to REMOVE, this parameter must contain the ID of a displayed image.</p>
<b>nDx</b>	<p>Pass either a number or the CENTERED constant in this parameter:</p> <ul style="list-style-type: none"> <li>• Pass a number to specify the horizontal distance in pixels between the edge of installation window and the edge of the image when nDrawOp is set to LOWER_LEFT, LOWER_RIGHT, UPPER_LEFT, or UPPER_RIGHT.</li> <li>• Pass the CENTERED constant to center the image on the horizontal axis when nDrawOp is set to LOWER_LEFT, LOWER_RIGHT, UPPER_LEFT, or UPPER_RIGHT. The image will be offset from the upper or lower edge of the installation window by the number of pixels specified in nDy. Since the CENTERED constant — when passed in nDx — centers the image horizontally, the argument in nDrawOp will affect only the vertical placement of the image.</li> </ul>

**Table 131** • PlaceBitmap Parameters (cont.)

Parameter	Description
<b>nDy</b>	<p>Pass either a number or the <code>CENTERED</code> constant in this parameter:</p> <ul style="list-style-type: none"> <li>• Pass a number to specify the vertical distance in pixels between the edge of installation window and the edge of the image when <code>nDrawOp</code> is set to <code>LOWER_LEFT</code>, <code>LOWER_RIGHT</code>, <code>UPPER_LEFT</code>, or <code>UPPER_RIGHT</code>.</li> <li>• Pass the <code>CENTERED</code> constant to center the image on the vertical axis when <code>nDrawOp</code> is set to <code>LOWER_LEFT</code>, <code>LOWER_RIGHT</code>, <code>UPPER_LEFT</code>, or <code>UPPER_RIGHT</code>. The image will be offset from the right or left edge of the installation window by the number of pixels specified in <code>nDx</code>. Since the <code>CENTERED</code> constant — when passed in <code>nDy</code> — centers the image vertically, the argument in <code>nDrawOp</code> will affect only the horizontal placement of the image.</li> </ul>
	 <p><b>Note</b> • You can pass the <code>CENTERED</code> constant in both the <code>nDx</code> and <code>nDy</code> parameters to center the image in the installation window. This is equivalent to passing the <code>CENTERED</code> constant in the <code>nDrawOp</code> parameter.</p>

**Table 131** • PlaceBitmap Parameters (cont.)

Parameter	Description
<b>nDrawOp</b>	<p>Specifies the bitmap's placement location, sets placement options, or removes a previously placed bitmap. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li> <b>BITMAPICON</b>—Indicates that the bitmap has transparent parts. You can use the bitwise OR operator (   ) to combine this constant with any of the other constants except TILED, FULLSCREEN or FULLSCREENSIZE. When BITMAPICON is ORed with one of those constants, the bitwise operation is ignored and BITMAPICON is used. <p>BITMAPICON has no effect when szName specifies a metafile or a 24-bit bitmap. Note that when you specify BITMAPICON, the bitmap is displayed normally, even if a special effect has been enabled with SetDisplayEffect.</p> </li> <li> <b>TILED</b>—The bitmap is tiled across the main installation window. This constant is normally used to create an installation background. When this constant is specified, location options are ignored and the bitmap is displayed normally, even if a special effect has been enabled with SetDisplayEffect. </li> <li> <b>FULLSCREEN</b>—Draw the image to fill the entire installation window. The image is not resized when its drawn. If a bitmap image is smaller than the InstallShield main window, it is centered in the window and the background is filled with the current background color. The default value is teal; it can be changed using the <a href="#">SetColor</a> function. When this constant is specified, location options are ignored and the bitmap is displayed normally, even if a special effect has been enabled with SetDisplayEffect. </li> <li> <b>FULLSCREENSIZE</b>—Draw and stretch the image to fill the entire installation window. When this constant is specified, location options are ignored and the bitmap is displayed normally, even if a special effect has been enabled with SetDisplayEffect. </li> </ul>
<b>nDrawOp (continued)</b>	<ul style="list-style-type: none"> <li> <b>CENTERED</b>—Places the bitmap in the center of the installation window. </li> <li> <b>LOWER_LEFT</b>—Places the bitmap in the lower left corner of the InstallShield installation window. </li> <li> <b>LOWER_RIGHT</b>—Places the bitmap in the lower right corner of the InstallShield installation window. </li> <li> <b>UPPER_LEFT</b>—Places the bitmap in the upper left corner of the InstallShield installation window. </li> <li> <b>UPPER_RIGHT</b>—Places the bitmap in the upper right corner of the InstallShield installation window. </li> <li> <b>REMOVE</b>—Removes a previously placed bitmap or metafile. Any special display effects that have been enabled with SetDisplayEffect are ignored. </li> </ul>

## Return Values

**Table 132** • PlaceBitmap Return Values

Return Value	Description
0	Indicates that the function successfully found and placed the image.
< 0	Indicates that the function was unable to find or place the image.

## Comments

InstallShield supports 2-color, 16-color, 256-color and true color (24-bit) bitmaps. Two-color, 16-color and 256-color bitmaps can have transparent portions.

Transparent bitmaps are useful for displaying images that appear to be integrated with the background window. Pixels in the bitmap that match a specified transparent color are not displayed; the background pixel at that location remains visible. In setups, transparent bitmaps that incorporate the company name and its logo in an artful design are often used as titles in the installation window.

To specify a transparent bitmap, you must pass the constant `BITMAPICON` in the parameter `nDrawOp`. You must also consider which color in the bitmap is to be transparent. The default transparent color is purple (`RGB(255,0,255)`). To specify a different transparent color, use the parameter `szName` as described below.

Because metafiles are drawn rather than placed, they are intrinsically transparent. If `BITMAPICON` is specified for a metafile, that parameter is ignored.



**Note** • Many special display effects are available for non-transparent bitmaps by using the [SetDisplayEffect](#) function. That function also provides limited display effects for metafiles.

The location of the bitmap within the window can be specified in one of two ways:

- By passing one of the location constants in the parameter `nDrawOp`.
- By passing a vertical and horizontal offset from the edge of the installation window in `nDx` and `nDy`.
- By passing the `CENTERED` constant in either `nDx` or `nDy` in combination with a horizontal or vertical offset.

Always remove any bitmaps or metafiles that are no longer needed by calling `PlaceBitmap` with the constant `REMOVE` as the parameter `nDrawOp`. Removing an unneeded bitmap is recommended even if another bitmap covers that bitmap completely because the palette entries for the first bitmap will not be released until the bitmap is removed.



**Tip** • A true color bitmap that is displayed on a system running in 16-color or 256-color mode will use only those colors available in the color palette; no additional colors will be allocated for the bitmap, even when additional color palette entries are available. If you anticipate that a setup with 24-bit bitmaps will be run on 16- or 256-color systems, include 16- or 256-color versions of the bitmaps. Then call [GetSystemInfo](#) with the `COLORS` parameter to determine the current color mode before selecting the bitmap to display.

Call `SetDisplayEffect` to set special effects for non-tiled, full-screen, transparent bitmaps; you can also set limited special effects for metafiles.

InstallShield does not support 24-bit transparent bitmaps. If you include a transparent color in a 24-bit bitmap and specify the `BITMAPICON` constant, the color will be displayed normally.

When you place a 256-color bitmap on a system running in 256-color mode, InstallShield attempts to allocate the bitmap's color palette into the system color palette. If multiple 256-color bitmaps are placed, InstallShield attempts to merge the color palettes of all visible bitmaps into the system color palette, giving precedence to the most recently placed bitmap. This behavior may cause previously placed bitmaps to change colors when additional bitmaps are displayed.

On a system running in 256-color mode with a 256-color dithered background, bitmaps that include many colors may cause some of the color palette entries used for the background to be reallocated; this can cause a gradient effect to appear in the background. Setups with bitmaps that use many colors should not use a 256-color gradient background if they will run on 256-color systems.

System color palettes exist only on systems that are running in 256-color mode. Systems running in high color (16-bit) or true color (24-bit) modes and systems running under in 65535 (16-bit) color mode do not have a system color palette. On these systems there are no color palette handling issues to consider; colors are displayed directly using the RGB color value. See [Preventing Color Distortion](#) for more information.

Because metafiles are rendered, they do not include a custom color palette. When a metafile is displayed on a 256-color system, no color palette handling takes place; the metafile is drawn with the colors currently available in the color palette. For that reason, you should not use metafiles that display colors other than the standard 16 colors in setups that will run on 256-color systems.

## PlaceBitmap Example

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the PlaceBitmap function.
*
* PlaceBitmap is called to display and remove bitmaps on the
* screen. The SetDisplayEffect function sets the display
* effect for the bitmap.
*
* Note: Before running this script, set the constant BMP_PATH
*       so that it references an existing bitmap file on the
*       target system.
*
\*-----*/

#define BMP_PATH "C:\\Windows\\Bubbles.bmp"
#define BITMAP_ID_1 12
#define BITMAP_ID_2 13
#define BITMAP_ID_3 14

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_Placebitmap(HWND);

function ExFn_Placebitmap(hMSI)
begin
```

```

Enable ( BACKGROUND );

// Display the bitmap in the upper left corner.
PlaceBitmap (BMP_PATH, BITMAP_ID_1, 10, 12, UPPER_LEFT);

// Set bitmap reveal effect.
SetDisplayEffect (EFF_REVEAL);

// Display the bitmap in the lower right corner.
PlaceBitmap (BMP_PATH, BITMAP_ID_2, 10, 10, LOWER_RIGHT);

Delay(3);

// Remove the bitmap in the upper left corner.
PlaceBitmap ("", BITMAP_ID_2, 0, 0, REMOVE);

// Remove the bitmap in the lower right corner.
PlaceBitmap ("", BITMAP_ID_1, 0, 0, REMOVE);

// Set bitmap fade in effect.
SetDisplayEffect (EFF_FADE);

// Display the bitmap at the center of screen.
PlaceBitmap (BMP_PATH, BITMAP_ID_3, CENTERED, CENTERED, 0);
Delay (3);

// Remove the bitmap at the center of the screen.
PlaceBitmap ("", BITMAP_ID_3, 0, 0, REMOVE);
Delay (1);

end;

```

## PlaceWindow

The **PlaceWindow** function changes the position of user interface objects. This includes billboards, Adobe Flash application files, and AVI files that are displayed at run time through **PlayMMedia**. Specify the distance between the sides of the object and the edges of the screen in nDx and nDy.

When using this function, be aware that an installation runs on a variety of screen resolutions. You may want to determine the extents of the screen before you position the objects. The distance is measured in pixels and is between the edge of the object and the edge of the corner of the specified screen.



**Note** • The **PlaceWindow** function does not have any effect on the type of billboard that is displayed on a progress dialog. To learn more about the different types of billboards, see *Billboard Styles and File Types for InstallScript and InstallScript MSI Projects*.

### Restrictions

This function cannot be used to position message boxes or custom dialogs.

- Message boxes cannot be positioned with this function because they are created using the native Windows API. A message box's position is determined by the Windows API and is not under the control of an installation.


- Custom dialogs cannot be positioned with this function. **PlaceWindow** does not work in conjunction with the **AskOptions**, **AskPath**, **AskText**, or **EnterDisk** functions. By default, a dialog appears in the center of the desktop, unless the background window mode is enabled. If the installation is in window mode, the dialog appears in the center of the background window.

## Syntax

```
PlaceWindow ( nObject, nDx, nDy, nCorner );
```


## Parameters

**Table 133** • PlaceWindow Parameters

Parameter	Description
<b>nObject</b>	<p>Specifies the object whose position is to be changed. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>ASKOPTIONS</b>—Moves the <a href="#">AskOptions</a> dialog.</li> <li>• <b>ASKPATH</b>—Moves the <a href="#">AskPath</a> dialog.</li> <li>• <b>ASKTEXT</b>—Moves the <a href="#">AskText</a> dialog.</li> <li>• <b>BACKGROUND</b>—Moves the background window.</li> <li>• <b>BILLBOARD</b>—Sets the location of billboards used during the file transfer process.</li> <li>• <b>ENTERDISK</b>—Moves the <a href="#">EnterDisk</a> dialog.</li> <li>• <b>MMEDIA_AVI</b>—Sets the window position for the next .avi file to be played. By default, the .avi file is played at run time in a window in the left corner of the screen, 10 pixels from the left and 10 pixels from the top.</li> <li>• <b>MMEDIA_SWF</b>—Sets the window position for the Adobe Flash application file (.swf) to be played.</li> <li>• <b>STATUS</b>—Moves the progress indicator.</li> <li>• <b>STATUSDLG</b>—Moves the dialog style progress indicator.</li> <li>• <b>STATUSEX</b>—Moves the Setup Status dialog.</li> <li>• <b>STATUSOLD</b>—Moves the old style progress indicator.</li> </ul> <p> <b>Tip</b> • When you call <i>PlaceWindow</i> to move the progress indicator or status dialog, be sure to pass the correct constant for the feedback object you have enabled in your setup. For example, if you called <a href="#">Enable</a> (<b>STATUSOLD</b>), you must pass <b>STATUSOLD</b> to <i>PlaceWindow</i>.</p>
<b>nDx</b>	Specifies the distance in pixels between the appropriate edge of the object and the edge of the screen on the horizontal axis.
<b>nDy</b>	Specifies the distance in pixels between the appropriate edge of the object and the edge of the screen on the vertical axis.



**Table 133** • PlaceWindow Parameters (cont.)

Parameter	Description
<b>nCorner</b>	<p>Specifies from which corner to measure the distances expressed in nDx and nDy. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>LOWER_LEFT</b>—Measures nDx from the left and nDy from the bottom of the main InstallShield window.</li> <li>• <b>LOWER_RIGHT</b>—Measures the nDx from the right and nDy from the bottom of the main InstallShield window.</li> <li>• <b>UPPER_LEFT</b>—Measures the nDx from the left and nDy from the top of the main InstallShield window.</li> <li>• <b>UPPER_RIGHT</b>—Measures the nDx from the right and nDy from the top of the main InstallShield window.</li> <li>• <b>CENTERED</b>—Centers the user interface object within the window.</li> </ul> <p> <b>Note</b> • <i>CENTERED can also be placed in nDx or nDy to center an object only horizontally or only vertically.</i></p>

## Return Values

**Table 134** • PlaceWindow Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully changed the position of the object.
<b>&lt; 0</b>	Indicates that the function was unable to change the position of the object.

## Additional Information

Use the [PlayMMedia](#) function if you want your installation to play an Adobe Flash application file (.swf) or an AVI file.

## PlaceWindow Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the PlaceWindow function.
*
* PlaceWindow is called to place the background window 50
* pixels to the right and below the upper left-hand corner

```

```

    * of the display.
    *
    \*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_PlaceWindow(HWND);

function ExFn_PlaceWindow(hMSI)
begin

    Enable ( BACKGROUND );

    Enable ( DEFWINDOWMODE );

    MessageBox ("This is the default position of the background window.",
                INFORMATION);

    // Delay calling PlaceWindow for three seconds.
    Delay(3);

    // Change the position of the background window.
    if (PlaceWindow (BACKGROUND, 50, 50, UPPER_LEFT) < 0) then
        MessageBox ("PlaceWindow failed.", SEVERE);
    else
        MessageBox ("This is the new position of the background window.",
                    INFORMATION);
        // Delay exiting script for three seconds.
        Delay(3);
    endif;

end;

```

## PlayMMedia

The **PlayMMedia** function plays an Adobe Flash application file (.swf), an AVI file, or a sound file (MIDI or WAVE).



**Tip** • If you are using **PlayMMedia** to display a Flash file or an AVI file, the installation must display a background window. For more information, see *Displaying a Background Window in InstallScript and InstallScript MSI Installations*.


*InstallShield* has support for displaying a Flash file as a billboard for your installation without displaying a background window. To learn more, see *Billboard Styles and File Types for InstallScript and InstallScript MSI Projects*.

### Syntax

```
PlayMMedia (nType, szFileName, nOperation, nReserved);
```

## Parameters

**Table 135 • PlayMMedia Parameters**

Parameter	Description
<b>nType</b>	Specify the type of file that you want your installation to play. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>MMEDIA_AVI</b>—The file is an AVI file.</li> <li>• <b>MMEDIA_MIDI</b>—The file is of MIDI sound format.</li> <li>• <b>MMEDIA_SWF</b>—The file is an Adobe Flash application file (.swf).</li> <li>• <b>MMEDIA_WAVE</b>—The file is of WAVE sound format.</li> </ul>
<b>szFileName</b>	Specify the fully qualified name of the file to be played.
<b>nOperation</b>	Specify the play mode. Pass any of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>MMEDIA_PLAYSYNCH</b>—Play synchronously.</li> <li>• <b>MMEDIA_PLAYASYNCH</b>—Play asynchronously. This constant can be combined with <b>MMEDIA_PLAYCONTINUOUS</b> by using the OR operator ( ).</li> <li>• <b>MMEDIA_PLAYCONTINUOUS</b>—Play in a continuous loop. This value cannot be used when playing a sound/AVI file in synchronous mode. It can be used only with files being played in asynchronous mode. Combine it with <b>MMEDIA_PLAYASYNCH</b> by using the OR operator ( ).</li> <li>• <b>MMEDIA_STOP</b>—Stop playing.</li> </ul> <div>  <p><b>Note</b> • A Flash file plays only once asynchronously, regardless of whether any <b>MMEDIA_PLAY*</b> constants are passed in <b>nOperation</b>.</p> </div>
<b>nReserved</b>	Pass the number zero in this parameter. No other value is allowed.

## Return Values

**Table 136 • PlayMMedia Return Values**

Return Value	Description
<b>0</b>	The function successfully played the file.
<b>&lt; 0</b>	The function was unable to play the file.  One example of when this occurs is if you specified a Flash file for <b>PlayMMedia</b> but the Flash Player is not present on the target system.

## Additional Information

If you are using a Flash file or an AVI file, you can use **SizeWindow** and **PlaceWindow** to control the size and placement of the background window that displays the Flash or AVI file.

## PlayMMedia Example

To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the PlayMMedia function.
*
* This script plays an AVI file during the setup.
*
* Note: To run this example script, create a project (or
*       insert into project) with several features and/or
*       subfeatures with components containing files. Then
*       add an AVI file to Disk 1 in the Support Files view in
*       the IDE. Change the file name in #define SOURCE line
*       below to specify your AVI file.
*
* Warning: Since this example does not include uninstallation
*          functionality, use this example only with projects
*          that do not overwrite important files, install
*          shared files, or update the registry.
*
\*-----*/

#define SOURCE SRCDIR + "windy7(1).avi"
#define TITLE1 "Playing AVI synchronously..."
#define TITLE2 "Playing AVI asynchronously and continuously..."

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_PlayMMedia(HWND);

function ExFn_PlayMMedia(hMSI)
    NUMBER nvDisk;
begin
    Enable ( BACKGROUND );

    // First, play the AVI synchronously to demonstrate how this causes
    // it to play by itself, with no other events taking place.
    SetTitle (TITLE1, 16, YELLOW);
    PlaceWindow (MMEDIA_AVI, 10, 10, UPPER_RIGHT);

    if (PlayMMedia (MMEDIA_AVI, SOURCE, MMEDIA_PLAYSYNCH, 0) < 0) then
        MessageBox ("Unable to play AVI file.", WARNING);
    endif;

    // Now play the AVI asynchronously. The AVI continues executing
    // as the file transfer occurs.
    SetTitle (TITLE2, 16, YELLOW);
```

```

PlaceWindow (MMEDIA_AVI, 10, 10, LOWER_RIGHT);

if (PlayMMedia (MMEDIA_AVI, SOURCE,
                MMEDIA_PLAYASYNCH | MMEDIA_PLAYCONTINUOUS, 0) < 0) then
    MessageBox ("Unable to play AVI file.", WARNING);
endif;

Enable (STATUSDLG);
Enable (INDVFILESTATUS);

StatusUpdate (ON, 99);

// Transfer the files.
ComponentMoveData (MEDIA, nvDisk, 0);

Disable (INDVFILESTATUS);
Disable (STATUSDLG);

// The AVI will stop playing when the setup exits. But you can
// stop it explicitly like this:
PlayMMedia (MMEDIA_AVI, SOURCE, MMEDIA_STOP, 0);

end;

```

## PostShowComponentDlg



**Caution** • This function is not supported in InstallShield because it is no longer required. The functions that required *PreShowComponentDlg* in InstallShield Professional 2.03 are not supported for use in Basic MSI projects. If you want to use one of these functions in your setup project, you must convert your Basic MSI project to the InstallScript MSI project type and add the function to the script.

The **PostShowComponentDlg** function converts InstallShield Professional components back to InstallShield–Windows Installer Edition features. You must call *PostShowComponentDlg* after calling a component dialog or function in your script. This function selects Windows Installer features based upon component selection.

### Syntax

```
PostShowComponentDlg (hMSI);
```

## Parameters

**Table 137** • PostShowComponentDlg Parameters

Parameter	Description
<b>hMSI</b>	The handle to the Windows Installer (MSI) database that is passed to your entry-point function.

## Return Values

**Table 138** • PostShowComponentDlg Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully converted InstallShield Professional components back to Windows Installer features.
<b>&lt; 0</b>	Indicates that the function was unable to convert InstallShield Professional components back to Windows Installer features.

# PreShowComponentDlg



**Caution** • This function is not supported in InstallShield because it is no longer required. The functions that required *PreShowComponentDlg* in InstallShield Professional 2.03 are not supported for use in Basic MSI projects. If you want to use one of these functions in your setup project, you must convert your Basic MSI project to the InstallScript MSI project type and add the function to the script.

The **PreShowComponentDlg** function converts InstallShield–Windows Installer Edition features to InstallShield Professional components. It also initializes feature costing functionality.

## Syntax

```
PreShowComponentDlg (hMSI);
```

## Parameters

**Table 139** • PreShowComponentDlg Parameters

Parameter	Description
hMSI	The handle to the Windows Installer (MSI) database that is passed to your entry-point function.

## Return Values

**Table 140** • PreShowComponentDlg Parameters

Return Value	Description
0	Indicates that the function successfully converted Windows Installer features to InstallShield Professional components.
< 0	Indicates that the function was unable to convert Windows Installer features to InstallShield Professional components.

# ProgDefGroupType

The **ProgDefGroupType** function sets the value of the ALLUSERS system variable. For more details, see [ALLUSERS](#).

## Syntax

```
ProgDefGroupType ( nType );
```

## Parameters

**Table 141** • ProgDefGroupType Parameters

Parameter	Description
<b>nType</b>	Specifies the value to use for the InstallScript variable ALLUSERS. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li><b>PERSONAL</b>—Set ALLUSERS equal to FALSE.</li> <li><b>COMMON</b>—Set ALLUSERS equal to TRUE.</li> </ul>

## Return Values

**Table 142** • ProgDefGroupType Return Values

Return Value	Description
0	This function always returns zero.





# Built-In Functions (Q-R)

For a list of functions by category, see [Built-In Functions by Category](#).

## QueryProgItem

The [GetShortcutInfo](#) function supersedes the **QueryProgItem** function.

The **QueryProgItem** function checks for the existence of a specific program item or subfolder name. If the InstallScript engine finds the item or subfolder, **QueryProgItem** returns its attributes. The attributes include the product's command line, working directory, icon path, shortcut key, and minimize flag.

To use **QueryProgItem**, enter information in the parameters szFolderName and szItemName. The InstallScript engine fills the remaining parameters with the program item's attributes.

### Syntax

```
QueryProgItem ( szFolderName, szItemName, svCmdLine, svWrkDir, svIconPath, nvIconIndex, svShortCutKey,  
                nvMinimizeFlag );
```

## Parameters

**Table 1 • QueryProgItem Parameters**

Parameter	Description
<b>szFolderName</b>	<p>Specifies the name of the folder that contains the item or subfolder. You can specify a fully qualified path for szFolderName, such as:</p> <pre>"C:\\ProgramData\\Microsoft\\Windows\\Start Menu\\Programs\\Games"</pre> <p>If szFolderName is null, <b>QueryProgItem</b> searches the default Programs directory. If you do not specify an absolute path (a path that includes a drive specification, for example, "C:\\Program Files\\AppName") for szFolderName, <b>QueryProgItem</b> searches for a subfolder under the default Programs directory; the location depends on the value of the InstallScript variable ALLUSERS, as well as the version of Windows on the target system.</p> <p>You can also use an InstallScript system variable:</p> <ul style="list-style-type: none"> <li>● <b>FOLDER_DESKTOP</b>—Queries items in the Desktop folder.</li> <li>● <b>FOLDER_STARTUP</b>—Queries items in the Startup menu.</li> <li>● <b>FOLDER_STARTMENU</b>—Queries items in the Start menu.</li> <li>● <b>FOLDER_PROGRAMS</b>—Queries items in the Start\\Programs menu.</li> </ul> <p>Or you could use a relative path, such as:</p> <pre>FOLDER_PROGRAMS ^ "ACCESSORIES\\GAMES"</pre>
<b>szItemName</b>	Specifies the name of the program item or subfolder to find.
<b>svCmdLine</b>	Returns either the command line of the item's executable file or the complete path to the subfolder.
<b>svWrkDir</b>	Returns the full path of the working directory of the program item. (Not applicable if szItemName is a subfolder.)
<b>svIconPath</b>	Returns the fully qualified file name of the .ico file or .exe file. (Not applicable if szItemName is a subfolder.)
<b>nvIconIndex</b>	Returns the index of the icon used for the program item. (Not applicable if szItemName is a subfolder.)
<b>svShortcutKey</b>	Returns the item's shortcut key. (Not applicable if szItemName is a subfolder.)

**Table 1 •** QueryProgItem Parameters (cont.)

Parameter	Description
<b>nvMinimizeFlag</b>	<p>(Not applicable if szItemName is a subfolder.) Returns one of the following constants, indicating whether an application window is minimized when first displayed:</p> <ul style="list-style-type: none"> <li>• <b>NULL</b>—Indicates that the application's window is not minimized upon startup.</li> <li>• <b>RUN_MINIMIZED</b>—Indicates that the application's window is minimized upon startup.</li> </ul>

## Return Values

**Table 2 •** QueryProgItem Return Values

Return Value	Description
<b>IS_ITEM (0)</b>	Indicates szItemName is a program item or shortcut in szFolderName.
<b>IS_FOLDER (1)</b>	Indicates szItemName is a subfolder in szFolderName.
<b>&lt; 0</b>	<p>Indicates that the function was unable to find the program item or subfolder name.</p> <p>You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a>.</p>

## Additional Information

The location of the Start menu is different under different languages. The InstallScript engine automatically selects the correct path.

## QueryProgItem Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the QueryProgItem function.
*
* QueryProgItem is called to find the attributes of a target
* file or folder.
*
* Note: Before running this script, set the defined constants
*       FOLDER_NAME and ITEM_NAME so that they reference an
*       existing folder name and folder item.

```

```

*
\*-----*/
// Define constants to reference the file or folder name.
#define FOLDER_NAME "C:\\Windows\\Start Menu\\Programs"
#define ITEM_NAME "InstallShield"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_QueryProgItem(HWND);

function ExFn_QueryProgItem(hMSI)
    STRING svCmdLine, svWrkDir, svIconPath;
    STRING svShortCutKey, svGroupPath, szTitle, szMsg, szInfo, svMinFlag;
    STRING svMinimizeFlag;
    NUMBER nvIconIndex, nvMinimizeFlag, nResult, nvMinFlag;
    LIST listInfo, listID;
begin

    // Search for an item in the FOLDER_NAME folder.
    nResult = QueryProgItem (FOLDER_NAME, ITEM_NAME, svCmdLine, svWrkDir,
                            svIconPath, nvIconIndex, svShortCutKey,
                            nvMinimizeFlag);

    // Create string list.
    listInfo = ListCreate (STRINGLIST);

    // Error check QueryProgItem.
    if (nResult < 0) then
        // Report the error; then abort.
        MessageBox("QueryProgItem failed.", SEVERE);
        abort;
    // Check if the item is an application.
    elseif (nResult = IS_ITEM) then
        // Add the command line to the string list.
        Sprintf(szInfo, "The command line of %s: %s", ITEM_NAME, svCmdLine);
        ListAddString(listInfo, szInfo, AFTER);

        // Add the working directory to string list.
        Sprintf(szInfo, "The working directory of %s: %s", ITEM_NAME, svWrkDir);
        ListAddString(listInfo, szInfo, AFTER);

        // Add the icon path to string list.
        Sprintf(szInfo, "The icon path of %s: %s", ITEM_NAME, svIconPath);
        ListAddString(listInfo, szInfo, AFTER);

        // Add icon index to string list.
        Sprintf(szInfo, "The index of the icon: %d", nvIconIndex);
        ListAddString(listInfo, szInfo, AFTER);

        // Add shortcut key to string list.
        Sprintf (szInfo, "The shortcut key of %s: %s", ITEM_NAME,
                svShortCutKey);
        ListAddString(listInfo, szInfo, AFTER);

        // Check if the item is a folder.
        elseif (nResult = IS_FOLDER) then
            // Add a message to string list.
            Sprintf (szInfo, "The item is a subfolder. QueryProgItem does not " +

```

```

        "retrieve very much information about subfolders.");
    ListAddString(listInfo, szInfo, AFTER);
endif;

// Display the string list.
szTitle = "QueryProgItem Example";
szMsg   = "The following are attributes of the item:";
SdShowInfoList(szTitle, szMsg, listInfo);

// Destroy the list.
ListDestroy (listID);

end;

```

## QueryShellMgr

The **QueryShellMgr** function obtains the name of the program shell being used by Microsoft Windows. For example, if the program shell is Explorer, QueryShellMgr returns the string "Explorer.exe" in svShellMgrName.

### Syntax

```
QueryShellMgr ( svShellMgrName );
```

## Parameters

**Table 3 • QueryShellMgr Parameters**

Parameter	Description
<b>svShellMgrName</b>	Returns the unqualified name (that is, without the drive designation or path) of the shell manager that is currently running.

## Return Values

**Table 4 • QueryShellMgr Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully retrieved the name of the program shell.
<b>&lt; 0</b>	Indicates that the function was unable to retrieve the name of the program shell.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

## Additional Information

If the shell on the target system is not Explorer, you may need to launch that shell with the [LaunchApp](#) function. The InstallScript functions that create program folders and program icons use a DDE conversation with the shell to create the program folders and program items. Most alternate shells such as the Norton Desktop emulate the Explorer shell. Therefore, they can create program folders and items.

In shells that do not emulate the Explorer shell, InstallShield cannot use the program folder and program item functions to create or modify the program folders and program items. Check with the manufacturer of the shell to determine how it handles the creation of program folders and program items using Microsoft DDE specifications.

## QueryShellMgr Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the QueryShellMgr function.
*
* QueryShellMgr is called to find the name of the shell
```

```

* manager. The name is then displayed in a message box.
*
\*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_QueryShellMgr(HWND);

function ExFn_QueryShellMgr(hMSI)
    STRING svShellMgrName, szTitle, szMsg;
    NUMBER nReturn;
begin

    // Get the name of the program shell.
    nReturn = QueryShellMgr (svShellMgrName);

    if (nReturn < 0) then
        // Report an error.
        MessageBox ("Could not retrieve the program shell.", SEVERE);
    else
        // Display the name of the shell.
        MessageBox ("The shell manager is " + svShellMgrName + ".", INFORMATION);
    endif;

end;

```

## ReadArrayProperty



**Project** • This information applies to *InstallScript* projects.

The **ReadArrayProperty** function is called in an object script to read the value of a specified property whose value is an array.

### Syntax

```
ReadArrayProperty ( nPropertyBag, szPropertyName, ArrayPointer );
```

## Parameters

**Table 5 • ReadArrayProperty Parameters**

Parameter	Description
<b>nPropertyBag</b>	Specifies a reference to the object's property bag object, in which property values are stored. (The value of nPropertyBag is passed by the setup engine to the ReadProperties function block within which ReadBoolProperty is called. This value is placed in the object script when you use the Add New Property dialog in the InstallScript view.)
<b>szPropertyName</b>	Specifies the name of the property whose value you want to read.
<b>ArrayPointer</b>	Returns a pointer to the specified array property.

## Return Values

**Table 6 • ReadArrayProperty Return Values**

Return Value	Description
<b>0</b>	This function always returns zero (0).

# ReadBoolProperty



**Project** • This information applies to InstallScript projects.

The **ReadBoolProperty** function is called in an object script to read the value of a specified property whose value is a Boolean.

## Syntax

```
ReadBoolProperty ( nPropertyBag, szPropertyName, bvPropertyValue );
```



## Parameters

**Table 7 • ReadBoolProperty Parameters**

Parameter	Description
<b>nPropertyBag</b>	Specifies a reference to the object's property bag object, in which property values are stored. (The value of nPropertyBag is passed by the setup engine to the ReadProperties function block within which ReadBoolProperty is called. This value is placed in the object script when you use the Add New Property dialog in the InstallScript view.)
<b>szPropertyName</b>	Specifies the name of the property whose value you want to read.
<b>bvPropertyValue</b>	Returns the value of the specified property.

## Return Values

**Table 8 • ReadBoolProperty Return Values**

Return Value	Description
<b>0</b>	This function always returns zero (0).

# ReadBytes

The **ReadBytes** function reads a specific number of bytes from a file starting at the current file pointer location. When this function returns, InstallShield relocates the file pointer to the new position at the end of the bytes read from the file.



**Note** • Before you can read from the file, which may be a file on the Internet, you must open the file in binary mode by calling [OpenFileMode](#) and [OpenFile](#).

The parameter nIndex is an index into the value specified by svString. Use the parameter nBytes to specify how many bytes beyond the parameter nIndex to read from the file. If the nIndex plus nBytes is a value larger than the length of the svString, only the number of bytes from the index of the string to the end of the string are read from the file. For example, if svString is declared to be 100 bytes long, the parameter nIndex is declared as 50 bytes and the parameter nBytes is 75 bytes, only the bytes between 49 and 99 (50 bytes instead of 75 bytes) are read.

## Syntax

```
ReadBytes ( nFileHandle, svString, nIndex, nBytes );
```

## Parameters

**Table 9 • ReadBytes Parameters**

Parameter	Description
<b>nFileHandle</b>	Specifies the file handle to a file opened in binary mode.
<b>svString</b>	Returns the bytes read from the file. The variable passed in this parameter must have been declared with an explicit size, and it must be large enough to accommodate the number of bytes specified by nBytes.
<b>nIndex</b>	Specifies an index into svString; data from the file is inserted into the string at this location.
<b>nBytes</b>	Specifies the number of bytes to read from the file. Bytes are read starting from the current location of the file pointer. InstallShield relocates the file pointer as the bytes are read.

## Return Values

**Table 10 • ReadBytes Return Values**

Return Value	Description
<b>X</b>	Indicates that the function successfully read bytes from the file, where X is the actual number of bytes returned in svString.
<b>&lt; 0</b>	Indicates that the function was unable to successfully read from the file.

## ReadBytes Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ReadBytes and SeekBytes functions.
*
* SeekBytes is called to position a file pointer to a
* specific location in a file that has been opened in binary
```

```

* mode. ReadBytes then reads a specific number of bytes,
* starting at this location. The bytes are read into a string,
* which is then displayed in a message box.
*
* Note: The defined constants EXAMPLE_DIR and EXAMPLE_BIN must
*       be set to an existing directory and file on the target
*       system.
*
\*-----*/

#define EXAMPLE_DIR "C:\\\"
#define EXAMPLE_BIN "Example.bin"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ReadBytes(HWND);

function ExFn_ReadBytes(hMSI)
    STRING svString;
    NUMBER nvFileHandle;
begin
    // Set the file mode to read/write.
    OpenFileMode (FILE_MODE_BINARY);

    // Open a binary file.
    if (OpenFile (nvFileHandle, EXAMPLE_DIR, EXAMPLE_BIN) < 0) then
        // Report an error; then abort.
        sprintfBox (SEVERE, "CopyBytes Example", "Could not open %s.",
            EXAMPLE_BIN);
        abort;
    endif;

    // Set the file pointer to the 16th byte in the file.
    SeekBytes (nvFileHandle, 15, FILE_BIN_START);

    // Read the next twenty-eight bytes into svString.
    if (ReadBytes (nvFileHandle, svString, 0, 28) < 0) then
        // Report an error.
        MessageBox ("ReadBytes failed.", SEVERE);
    else
        // Display the string.
        sprintfBox (INFORMATION, "ReadBytes Example", "Bytes read: %s",
            svString);
    endif;

    // Close the file.
    CloseFile (nvFileHandle);

end;

```

## ReadNumberProperty



**Project** • This information applies to InstallScript projects.

The **ReadNumberProperty** function is called in an object script to read the value of a specified property whose value is a number.

## Syntax

```
ReadNumberProperty ( nPropertyBag, szPropertyName, nvPropertyValue );
```

## Parameters

**Table 11** • ReadNumberProperty Parameters

Parameter	Description
<b>nPropertyBag</b>	Specifies a reference to the object's property bag object, in which property values are stored. (The value of nPropertyBag is passed by the setup engine to the ReadProperties function block within which ReadBoolProperty is called. This value is placed in the object script when you use the Add New Property dialog in the InstallScript view.)
<b>szPropertyName</b>	Specifies the name of the property whose value you want to read.
<b>nvPropertyValue</b>	Returns the value of the specified property.

## Return Values

**Table 12** • ReadNumberProperty Return Values

Return Value	Description
<b>0</b>	This function always returns zero (0).

# ReadStringProperty



**Project** • This information applies to InstallScript projects.

The **ReadStringProperty** function is called in an object script to read the value of a specified property whose value is a string.

## Syntax

```
ReadStringProperty ( nPropertyBag, szPropertyName, svPropertyValue );
```

## Parameters

**Table 13 • ReadStringProperty Parameters**

Parameter	Description
<b>nPropertyBag</b>	Specifies a reference to the object's property bag object, in which property values are stored. (The value of nPropertyBag is passed by the setup engine to the ReadProperties function block within which ReadBoolProperty is called. This value is placed in the object script when you use the Add New Property dialog in the InstallScript view.)
<b>szPropertyName</b>	Specifies the name of the property whose value you want to read.
<b>svPropertyValue</b>	Returns the value of the specified property.

## Return Values

**Table 14 • ReadStringProperty Return Values**

Return Value	Description
<b>0</b>	This function always returns zero (0).

# RebootDialog



**Project •** This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **RebootDialog** function displays a message box that enables end users to specify whether they want to restart the computer. The selected option is performed at the end of the installation.

When you call a function with the SHAREDFILE or LOCKEDFILE option and locked .dll or .exe files are encountered, updated versions of the locked files are copied to the target system and the system variable BATCH\_INSTALL is set to TRUE. **RebootDialog** automatically commits the locked files for update when the system is restarted, unless the user selects the **No, I will restart my computer later** option.

The InstallScript engine makes every attempt not to restart the system when other instances of the installation are running. Because of this, you must make sure all other instances of the installation are shut down before calling **RebootDialog**. Your message to end users should request that they ensure all other applications are shut down before restarting the system.



**Note** • An alternative to the **RebootDialog** function is [SdFinishReboot](#), which has a better look and feel than the **RebootDialog** dialog.

## Syntax

```
RebootDialog ( szTitle, szMsg, nDefChoice );
```

## Parameters

**Table 15** • RebootDialog Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title Restarting Windows, pass a null string ("" ) in this parameter.
<b>szMsg</b>	Specifies the message to display in the dialog. To display the default instructions for this dialog, pass a null string ("" ) in this parameter.
<b>nDefChoice</b>	Specifies the default option button selection. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li><b>SYS_BOOTMACHINE</b>—The option to reboot the computer (Yes, I want to restart my computer now) is the default option button selection.</li> <li><b>0</b>—The option to not restart the computer (No, I will restart my computer later) is the default option button selection.</li> </ul>

## Return Values

**Table 16** • RebootDialog Return Values

Return Value	Description
<b>WILL_REBOOT</b>	Indicates that the end user selected the Yes, I want to restart my computer now option button.
<b>0</b>	Indicates that the end user selected the No, I will restart my computer later option button.

## Additional Information

The message box that is displayed by the **RebootDialog** function cannot be displayed with a skin; it appears the same regardless of whether you have specified a skin.

## RebootDialog Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the RebootDialog function.
 *
 * This script calls RebootDialog to display a dialog that
 * asks the user whether or not to reboot the computer. The call
 * to RebootDialog passes a null string in parameter 2 to display
 * the default message and 0 in parameter 3 to make the default
 * selection "No, I will restart my computer later."
 *
 * Warning: If the end user selects Yes in this dialog, the
 *         computer is rebooted.
 *
\*-----*/

#define TITLE_TEXT "RebootDialog Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_RebootDialog(HWND);

function ExFn_RebootDialog(hMSI)
    NUMBER nvDefChoice;
begin

    // Query to reboot computer.
    RebootDialog (TITLE_TEXT, "", 0);

end;

```

## RegDBConnectRegistry



**Project** • For *InstallScript MSI* and *Basic MSI* projects, it is recommended that you use the *Registry* view in *InstallShield* instead of creating registry keys and values through *InstallScript* code. Handling all of your registry changes in this way allows for a clean uninstallation through the Windows Installer service.

The **RegDBConnectRegistry** function creates a connection to a remote registry. After you have opened the connection, you can create, delete, or retrieve registry keys, value names, and value pairs on a remote registry much as you would on a local registry. This functionality is also supported on 64-bit systems with some [limitations](#).



**Note** • This function supports the 64-bit parts of the registry by using the `REGDB_OPTION_WOW64_64KEY` option. For more information, see [REGDB\\_OPTIONS](#).

RegDBConnectRegistry allows you to edit only one registry root key each time the remote registry is opened, and you can edit only keys and values under either HKEY\_LOCAL\_MACHINE or HKEY\_USERS. When you call RegDBConnectRegistry, you must specify which root key you want to be able to edit. If you want to edit the other root key or one its subkeys, you must close and re-open the connection.



---

**Caution** • Because you set the root key by calling *RegDBConnectRegistry*, you cannot call *RegDBSetDefaultRoot* after you have established a connection to a remote registry. When you call *RegDBDisconnectRegistry*, all calls to registry-related functions affect the local registry, and you can then call *RegDBSetDefaultRoot* to change the root key.



---

**Note** • If you are trying to open a registry on a remote Windows system, you must have administrator privileges. This function is intended for use by system administrators for network installations.

## Syntax

```
RegDBConnectRegistry ( szRemoteSystem, nKeyType, nReserved );
```



## Parameters

**Table 17 • RegDBConnectRegistry Parameters**

Parameter	Description
<b>szRemoteSystem</b>	Specifies the name of the system with which to connect, such as "RemoteSys". If you pass a null string ("") in this parameter, the function creates a connection to the local registry.
<b>nKeyType</b>	Specifies one of the following constants: HKEY_LOCAL_MACHINE or HKEY_USERS.
<b>nReserved</b>	Pass zero in this parameter. No other value is allowed.

## Return Values

**Table 18 • RegDBConnectRegistry Return Values**

Return Value	Description
<b>0</b>	Indicates this function successfully established a connection to the system registry.
<b>REGDB_ERR_CONNECTIONEXISTS (-6)</b>	A connection to a remote registry already exists. It must be closed using RegDBDisconnectRegistry before you can call RegDBConnectRegistry again.
<b>REGDB_ERR_CORRUPTEDREGISTRY (-4)</b>	Indicates that the remote registry is corrupted and cannot be accessed.
<b>REGDB_ERR_INITIALIZATION (-2)</b>	Indicates that the registry services could not be initialized. Make sure Remote Administration is enabled and that you have appropriate privileges to be able to write to the registry.
<b>REGDB_ERR_INVALIDHANDLE (-5)</b>	The key name provided for the remote registry is not allowed.
<b>REGDB_ERR_INVALIDNAME (-3)</b>	Indicates that the system in szRemoteSystem could not be found. Check the name and try again.
<b>-1</b>	Other error.

You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling [FormatMessage](#).

## Additional Information

By default, any text that is surrounded by angle brackets—for example, "<my registry entry text>"—in this function's string arguments is interpreted as a text substitution and is processed accordingly. To disable text substitution processing for the string arguments of registry functions, call **Disable** with the REGISTRYFUNCTIONS\_USETEXTSUBS argument.

## RegDBConnectRegistry Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the RegDBConnectRegistry function and
 * RegDBDisconnectRegistry function.
 *
 * Note: In order for this script to run properly, you must set
 *       the preprocessor constants to a valid remote computer
 *       with remote administration enabled. Both computers
 *       must also have the remote-registry service enabled.
 *
 \*-----*/

#define REMOTE "IShield_NT1"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_RegDBConnectRegistry(HWND);

function ExFn_RegDBConnectRegistry(hMSI)
    STRING szRemoteMachine, szKey, szTitle, szMsg;
    NUMBER nKeyType, nReturn;
begin

    szTitle          = "RegDBConnectRegistry & RegDBDisconnectRegistry";
    szRemoteMachine = REMOTE;
    nKeyType          = HKEY_LOCAL_MACHINE;
    szMsg             = "Setup will now connect to %s.";

    sprintfBox (INFORMATION, szTitle, szMsg, szRemoteMachine);

    // Connect to the remote computer's registry. All registry-related
    // function calls will now alter only the remote computer.
    nReturn = RegDBConnectRegistry (szRemoteMachine, nKeyType, 0);

    if (nReturn < 0) then
        szMsg = "RegDBConnectRegistry failed.\n\nCould not connect to remote " +
            "system.";
        MessageBox (szMsg, SEVERE);
        abort;
    else
        szMsg = "Successfully connected to %s.";
        sprintfBox (INFORMATION, szTitle, szMsg, szRemoteMachine);
    endif;

    // Create a key on the remote computer.
    szKey  = "SOFTWARE\\InstallShield\\Test Key";
    nReturn = RegDBCreateKeyEx(szKey, "");
```

```

if (nReturn < 0) then
    szMsg = "RegDBCreateKeyEx failed.\n\nCould not create key on remote " +
        "machine.";
    MessageBox (szMsg , SEVERE);
else
    szMsg = "Successfully created %s on %s.";
    sprintfBox (INFORMATION, szTitle, szMsg, szKey, szRemoteMachine);

    // Verify that the key now exists in the remote registry.
    nReturn = RegDBKeyExist(szKey);

    if (nReturn < 0) then
        szMsg = "RegDBKeyExist failed.\n\nRemote key does not exist.";
        MessageBox (szMsg, SEVERE);
    else
        szMsg = "%s exists.";
        sprintfBox (INFORMATION, szTitle, szMsg, szKey);
    endif;
endif;

// Delete the key that was created on the remote computer.
nReturn = RegDBDeleteKey(szKey);

if (nReturn < 0) then
    MessageBox("RegDBDeleteKey failed.\n\nRemote key could not be deleted.",
        INFORMATION);
else
    szMsg = "Successfully deleted %s on %s.";
    sprintfBox(INFORMATION, szTitle, szMsg, szKey, szRemoteMachine);
endif;

// Disconnect from the remote registry. All registry-related functions
// will now alter only the local registry.
nReturn = RegDBDisconnectRegistry(0);

if (nReturn < 0) then
    MessageBox("RegDBDisconnectRegistry failed.\n\nRemote registry still " +
        "connected.", SEVERE);
else
    MessageBox("RegDBDisconnectRegistry successful.\n\nRemote registry " +
        "disconnected.", INFORMATION);
endif;

end;

```

## RegDBCopYKeys



**Project** • For InstallScript MSI and Basic MSI projects, it is recommended that you use the Registry view in InstallShield instead of creating registry keys and values through InstallScript code. Handling all of your registry changes in this way allows for a clean uninstallation through the Windows Installer service.

The **RegDBCopYKeys** function copies the registry keys and values under the key specified by szSourceKey to the key specified by szTargetKey.



---

**Note** • This function supports the 64-bit parts of the registry by using the `REGDB_OPTION_WOW64_64KEY` option. For more information, see [REGDB\\_OPTIONS](#).

## Syntax

```
RegDBCopyKeys ( szSourceKey, szTargetKey, nRootKeySource, nRootKeyTarget );
```

## Parameters

**Table 19 • RegDBCpyKeys Parameters**

Parameter	Description
<b>szSourceKey</b>	Specifies the name of the key whose subkeys and values are to be copied. Separate different levels in the subkey with a double backslash (\\).
<b>szTargetKey</b>	Specifies the name of the key to be copied to. Separate different levels in the subkey with a double backslash (\\). If this key does not exist, RegDBCpyKeys creates it. If this key does exist, any existing values under the key that have the same names as values under szSourceKey are overwritten; this includes values under identically named subkeys.
<b>nRootKeySource</b>	Specifies the root key of szSourceKey. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• HKEY_CLASSES_ROOT</li> <li>• HKEY_CURRENT_USER</li> <li>• HKEY_LOCAL_MACHINE</li> <li>• HKEY_USERS</li> <li>• HKEY_CURRENT_CONFIG</li> <li>• HKEY_DYN_DATA</li> </ul>
<b>nRootKeyTarget</b>	Specifies the root key of szTargetKey. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• HKEY_CLASSES_ROOT</li> <li>• HKEY_CURRENT_USER</li> <li>• HKEY_LOCAL_MACHINE</li> <li>• HKEY_USERS</li> <li>• HKEY_CURRENT_CONFIG</li> <li>• HKEY_DYN_DATA</li> </ul>

## Return Values

**Table 20 • RegDBCpyKeys Return Values**

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the function successfully copied the keys and values.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was unable to copy the keys and values.  You can obtain the error message text associated with a large negative return value—for example, -2147024891(0x80070005)—by calling <a href="#">FormatMessage</a> .

## Additional Information

By default, any text that is surrounded by angle brackets—for example, "<my registry entry text>"—in this function's string arguments is interpreted as a text substitution and is processed accordingly. To disable text substitution processing for the string arguments of registry functions, call **Disable** with the `REGISTRYFUNCTIONS_USETEXTSUBS` argument.

# RegDBCopyValues



**Project** • For InstallScript MSI and Basic MSI projects, it is recommended that you use the Registry view in InstallShield instead of creating registry keys and values through InstallScript code. Handling all of your registry changes in this way allows for a clean uninstallation through the Windows Installer service.

The **RegDBCopyValues** function copies the registry values under the key specified by `szSourceKey` to the key specified by `szTargetKey`.



**Note** • This function supports the 64-bit parts of the registry by using the `REGDB_OPTION_WOW64_64KEY` option. For more information, see [REGDB\\_OPTIONS](#).

## Syntax

```
RegDBCopyValues ( szSourceKey, szTargetKey, nRootKeySource, nRootKeyTarget );
```

## Parameters

**Table 21** • RegDBCpyValues Parameters

Parameter	Description
<b>szSourceKey</b>	Specifies the name of the key whose values are to be copied. Separate different levels in the subkey with a double backslash (\\).
<b>szTargetKey</b>	Specifies the name of the key to be copied to. Separate different levels in the subkey with a double backslash (\\). If this key does not exist, RegDBCpyKeys creates it. If this key does exist, any existing values under the key that have the same names as values under szSourceKey are overwritten.
<b>nRootKeySource</b>	Specifies the root key of szSourceKey. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• HKEY_CLASSES_ROOT</li> <li>• HKEY_CURRENT_USER</li> <li>• HKEY_LOCAL_MACHINE</li> <li>• HKEY_USERS</li> <li>• HKEY_CURRENT_CONFIG</li> <li>• HKEY_DYN_DATA</li> </ul>
<b>nRootKeyTarget</b>	Specifies the root key of szTargetKey. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• HKEY_CLASSES_ROOT</li> <li>• HKEY_CURRENT_USER</li> <li>• HKEY_LOCAL_MACHINE</li> <li>• HKEY_USERS</li> <li>• HKEY_CURRENT_CONFIG</li> <li>• HKEY_DYN_DATA</li> </ul>

## Return Values

**Table 22** • RegDBCpyValues Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the function successfully copied the values.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was unable to copy the values.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

## Additional Information

By default, any text that is surrounded by angle brackets—for example, "<my registry entry text>"—in this function's string arguments is interpreted as a text substitution and is processed accordingly. To disable text substitution processing for the string arguments of registry functions, call **Disable** with the `REGISTRYFUNCTIONS_USETEXTSUBS` argument.

# RegDBCreateKeyEx



**Project** • For InstallScript MSI and Basic MSI projects, it is recommended that you use the Registry view in InstallShield instead of creating registry keys and values through InstallScript code. Handling all of your registry changes in this way allows for a clean uninstallation through the Windows Installer service.

The **RegDBCreateKeyEx** function creates a key in the registry. You can also associate a class object with the newly created key (advanced users only). The newly created key does not have a value associated with it.

When logging is enabled, the InstallScript engine logs each key in the path that is passed through `szKey` to **RegDBCreateKeyEx**. For example, you might pass the following for `szKey`:

**"Software" ^ IFX\_COMPANY\_NAME ^ IFX\_PRODUCT\_NAME**

In this case, the InstallScript engine logs several keys:

- Software
- The value of the `IFX_COMPANY_NAME` variable
- The value of the `IFX_PRODUCT_NAME` variable

The log flags a key as "created" if it did not already exist. In the aforementioned example, the Software key already exists on all target systems (`HKEY_LOCAL_MACHINE` or `HKEY_CURRENT_USER`). The Software key is logged, but its "created" flag would be false, indicating that the key should not be removed during uninstallation. Any keys that are created and logged (because they did not already exist at the time the installation was running) are removed during uninstallation.

When a key is uninstalled, all of its subkeys are also uninstalled. Therefore, if you use **RegDBCreateKeyEx** to create a key or keys under a key that is already logged for uninstallation, the keys that you create will be uninstalled when the higher-level key is uninstalled. This behavior occurs regardless of whether logging is enabled when the installation creates your keys and regardless of the order in which the installation creates the keys. Therefore, in the aforementioned example, if logging is also enabled for a second installation that likewise uses **RegDBCreateKeyEx** to create a key for a different product under the `IFX_COMPANY_NAME` key that the first installation created, and the then end user uninstalls the first product, the entire `IFX_COMPANY_NAME` key, with both of the product subkeys, are removed. This may make the second product behave unexpectedly.

If you want to be able to share registry keys among multiple installations, it is recommended that you use the Registry view to configure the registry entries, instead of using the **RegDBCreateKeyEx** function. In the Registry view of an InstallScript project, you can mark a registry keys as shared (by right-clicking the key and then clicking **Shared among several applications**). During the uninstallation, the InstallScript engine removes a shared key only if no other logged installations that share the key still exist on the machine.

To view all of the registry keys that were logged during an installation and find out how each of their "created" flags were set, use the InstallShield Cabinet and Log File Viewer.



For more information about logging, see [InstallScript Functions that Are Logged for Uninstallation](#).



**Note** • This function supports the 64-bit parts of the registry by using the `REGDB_OPTION_WOW64_64KEY` option. For more information, see [REGDB\\_OPTIONS](#).

## Syntax

```
RegDBCreateKeyEx ( szKey, szClass );
```

## Parameters

**Table 23** • RegDBCreateKeyEx Parameters

Parameter	Description
<b>szKey</b>	Specify the name of the key to create. Separate different levels in the subkey with a double backslash (\\).  If the root key is <code>HKEY_CLASSES_ROOT</code> , you do not need to specify the <code>HKEY_CLASSES_ROOT</code> in this parameter. Unless you specify otherwise, the InstallScript engine creates the key as a subkey of <code>HKEY_CLASSES_ROOT</code> . To specify a different root key, you can call <a href="#">RegDBSetDefaultRoot</a> before calling <b>RegDBCreateKeyEx</b> .
<b>szClass</b>	Specify the class name to associate with this key.

## Return Values

**Table 24** • RegDBCreateKeyEx Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully created the subkey.
<b>&lt; 0</b>	Indicates that the function was unable to create the subkey.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

## Additional Information

**RegDBCreateKeyEx** is a general registry-related function, designed to work with all registry keys, including those handled by the special registry-related functions. For more information on special registry-related functions, see [Special Registry-Related Functions](#).



**Note** • Windows does not allow the creation of a key directly under `HKEY_LOCAL_MACHINE` or `HKEY_USERS`.

By default, any text that is surrounded by angle brackets—for example, "<my registry entry text>"—in this function's string arguments is interpreted as a text substitution and is processed accordingly. To disable text substitution processing for the string arguments of registry functions, call **Disable** with the `REGISTRYFUNCTIONS_USETEXTSUBS` constant.

## RegDBCreateKeyEx Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the RegDBCreateKeyEx and RegDBKeyExist functions.
*
* First, RegDBCreateKeyEx is called to create a subkey with no
* class value in the HKEY_CLASSES_ROOT key. Then, RegDBKeyExist
* is then called to check if the key was created.
*
* RegDBCreateKeyEx is called again to create a multi-level subkey
* with a class value associated with it under HKEY_CLASSES_ROOT.
* Then RegDBKeyExist is called again to check for the existence
* of the new key.
*
\*-----*/

#define TITLE_TEXT "RegDBCreateKeyEx & RegDBKeyExist"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_RegDBCreateKeyEx(HWND);

function ExFn_RegDBCreateKeyEx(hMSI)
    STRING szKey, szClass, szKeyRoot, szMsg, svLogFile;
    NUMBER nResult1, nResult2;
begin
    // Create a key with no class value.
    szKey = "CreateKeyExample";
    szClass = "";

    if (RegDBCreateKeyEx(szKey, szClass) < 0) then
        MessageBox ("First call to RegDBCreateKeyEx failed.", SEVERE);
        abort;
    else
        sprintfBox (INFORMATION, TITLE_TEXT, "Successfully created: %s", szKey);

        // Check to see if the key just created exists.
        if (RegDBKeyExist (szKey) < 0) then
            MessageBox ("First call to RegDBKeyExist failed.", SEVERE);
        else
            sprintfBox (INFORMATION, TITLE_TEXT, "%s exists.", szKey);
```

```

        endif;
    endif;

    if (RegDBDeleteKey (szKey) < 0) then
        MessageBox ("RegDBDeleteKey failed.", SEVERE);
    endif;

    // Create a key with more than one sublevel and a class value.
    szKey      = "ShareWare\\Games\\CoolChess";
    szClass    = "LastPlayed";
    szKeyRoot = "ShareWare";

    if (RegDBCreateKeyEx(szKey, szClass) < 0) then
        MessageBox ("Second call to RegDBCreateKeyEx failed.", SEVERE);
        abort;
    else
        sprintfBox (INFORMATION, TITLE_TEXT, "Successfully created: %s", szKey);

        // Check if the newly created multi-level key exists.
        if (RegDBKeyExist (szKeyRoot) < 0) then
            MessageBox ("Second call to RegDBKeyExist failed.", SEVERE);
        else
            sprintfBox (INFORMATION, TITLE_TEXT, "%s exists.", szKey);
        endif;
    endif;

    if (RegDBDeleteKey (szKey) < 0) then
        MessageBox ("RegDBDeleteKey failed.", SEVERE);
    endif;

end;

```

## RegDBDeleteItem



**Project** • For InstallScript MSI and Basic MSI projects, it is recommended that you use the Registry view in InstallShield instead of creating registry keys and values through InstallScript code. Handling all of your registry changes in this way allows for a clean uninstallation through the Windows Installer service.

**RegDBDeleteItem** is a special registry-related function designed to work with certain predefined registry keys. The **RegDBDeleteItem** function deletes values under the per application paths key or the application uninstallation key, depending on the value of nItem. Calling **RegDBDeleteItem** with either the REGDB\_APPPATH or the REGDB\_APPPATH\_DEFAULT option results in the deletion of the per application paths key.



**Note** • The InstallScript engine currently does not support writing or reading Add or Remove Programs information for a product in the 64-bit part of the registry. Therefore, using the REGDB\_OPTION\_WOW64\_64KEY option with the [REGDB\\_OPTIONS](#) system variable is not supported for this registry function. Enabling the REGDB\_OPTION\_WOW64\_64KEY option has no effect on where registry entries are created by this function.

### Syntax

```
RegDBDeleteItem( nItem );
```

## Parameters


**Table 25 •** RegDBDeleteItem Parameters

Parameter	Description
<b>nItem</b>	<p>Specifies the item to delete. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>REGDB_APPPATH</b>—The data of the Path value under the per-application paths key. This key is deleted by the installation as a result of calling <b>CreateInstallationInfo</b>. You must call <b>CreateInstallationInfo</b> to create this key before calling <b>RegDBDeleteItem</b>. (In an event-based script, <b>CreateInstallationInfo</b> is called in the default <b>OnMoveData</b> event handler code.)</li> <li>• <b>REGDB_APPPATH_DEFAULT</b>—The data of the DefaultPath value under the per-application paths key. This key is created by the installation as a result of calling <b>CreateInstallationInfo</b>. You must call <b>CreateInstallationInfo</b> to create this key before calling <b>RegDBDeleteItem</b>. (In an event-based script, <b>CreateInstallationInfo</b> is called in the default <b>OnMoveData</b> event handler code.)</li> <li>• <b>REGDB_UNINSTALL_COMMENTS</b>—The data of the Comments value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_CONTACT</b>—The data of the Contact value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_DISPLAY_VERSION</b>—The data of the DisplayVersion value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_DISPLAYICON</b>—The data of the DisplayIcon value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_HELPLINK</b>—The data of the HelpLink value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_HELPTELEPHONE</b>—The data of the HelpTelephone value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_INSTALLDATE</b>—The data of the InstallDate value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_INSTALLLOC</b>—The data of the InstallLocation value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_INSTALLSOURCE</b>—The data of the InstallSource value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_LANGUAGE</b>—The data of the Language value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_LOGFILE</b>—The data of the LogFile value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_MAINT_OPTION</b>—The data of the LogMode value under the uninstallation key.</li> </ul>

**Table 25 •** RegDBDeleteltem Parameters (cont.)

Parameter	Description
<b>nItem (cont.)</b>	<ul style="list-style-type: none"> <li>• <b>REGDB_UNINSTALL_MAJOR_VERSION</b>—The data of the VersionMajor value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_MAJOR_VERSION_OLD</b>—The data of the MajorVersion value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_MINOR_VERSION</b>—The data of the VersionMinor value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_MINOR_VERSION_OLD</b>—The data of the MinorVersion value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_MODIFYPATH</b>—The data of the ModifyPath value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_NAME</b>—The data of the DisplayName value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_NOMODIFY</b>—The data of the NoModify value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_NOREMOVE</b>—The data of the NoRemove value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_NOREPAIR</b>—The data of the NoRepair value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_PRODUCTGUID</b>—The data of the ProductGuid value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_PRODUCTID</b>—The data of the ProductId value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_PUBLISHER</b>—The data of the Publisher value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_README</b>—The data of the Readme value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_REGCOMPANY</b>—The data of the RegCompany value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_REGOWNER</b>—The data of the RegOwner value under the uninstallation key.</li> </ul>

**Table 25** • RegDBDeleteItem Parameters (cont.)

Parameter	Description
<b>nItem</b> <b>(cont.)</b>	<ul style="list-style-type: none"> <li>• <b>REGDB_UNINSTALL_STRING</b>—The data of the UninstallString value under the uninstallation key. The UninstallString value is created when the <a href="#">MaintenanceStart</a> or <a href="#">DeinstallStart</a> functions are called.</li> <li>• <b>REGDB_UNINSTALL_SYSTEMCOMPONENT</b>—The data of the SystemComponent value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_URLINFOABOUT</b>—The data of the URLInfoAbout value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_URLUPDATEINFO</b>—The data of the URLUpdateInfo value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_VERSION</b>—The data of the Version value under the uninstallation key. When this constant is used, szValue specifies the installed product's version number expressed as a string, for example, "16777216".</li> </ul>
 <p><b>Important</b> • Do not use <code>REGDB_WINCURRVER_REGOWNER</code> or <code>REGDB_WINCURRVER_REGORGANIZATION</code> with this function. Windows sets these values, and they should not be deleted by an installation.</p>	

## Return Values

**Table 26** • RegDBDeleteItem Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully set the value.
<b>&lt; 0</b>	Indicates that the function failed.

## Additional Information

- By default, any text that is surrounded by angle brackets—for example, "<my registry entry text>"—in this function's string arguments is interpreted as a text substitution and is processed accordingly. To disable text substitution processing for the string arguments of registry functions, call **Disable** with the `REGISTRYFUNCTIONS_USETEXTSUBS` argument.
- **RegDBDeleteItem** does not write anything to the uninstall log file; thus, calling this function does not have any effect how the application is uninstalled.

# RegDBDeleteKey



**Project** • For InstallScript MSI and Basic MSI projects, it is recommended that you use the Registry view in InstallShield instead of creating registry keys and values through InstallScript code. Handling all of your registry changes in this way allows for a clean uninstallation through the Windows Installer service.

The **RegDBDeleteKey** function deletes a specific key and its associated value from the registry. All subkeys of the deleted key are also deleted, along with their associated values.

InstallShield assumes the key specified in `szSubKey` is a subkey of `HKEY_CLASSES_ROOT`. You can use `RegDBSetDefaultRoot` to specify another root key.

`RegDBDeleteKey` is a general registry-related function, designed to work with all registry keys, including those handled by the special registry-related functions. For more information on special registry-related functions, see [Special Registry-Related Functions](#).



**Note** • This function supports the 64-bit parts of the registry by using the `REGDB_OPTION_WOW64_64KEY` option. For more information, see [REGDB\\_OPTIONS](#).

## Syntax

```
RegDBDeleteKey ( szSubKey );
```

## Parameters

**Table 27** • RegDBDeleteKey Parameters

Parameter	Description
<b>szSubKey</b>	Specifies the name of the key to delete. Separate different levels in the subkey with a double backslash (\\).

## Return Values

**Table 28** • RegDBDeleteKey Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully deleted the key.
<b>&lt; 0</b>	Indicates that the function was unable to delete the key.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

## Additional Information

- By default, any text that is surrounded by angle brackets—for example, "<my registry entry text>"—in this function's string arguments is interpreted as a text substitution and is processed accordingly. To disable text substitution processing for the string arguments of registry functions, call **Disable** with the `REGISTRYFUNCTIONS_USETEXTSUBS` argument.
- RegDBDeleteKey** does not write anything to the uninstall log file; thus, calling this function does not have any effect how the application is uninstalled.

## RegDBDeleteKey Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the RegDBDeleteKey function.
*
* This example creates and then deletes a registry key.
* Almost all of the examples provided with the registry functions
* use this function to delete a key. Please refer to those
* examples for more information.
*
\*-----*/
```



```

#define TITLE_TEXT "RegDBDeleteKey Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_RegDBDeleteKey(HWND);

function ExFn_RegDBDeleteKey(hMSI)
    STRING szKey, szClass, szKeyRoot, szMsg, svLogFile;
    NUMBER nResult1, nResult2;
begin
    // Set up parameters for call to RegDBCreateKeyEx.
    szKey = "DeleteMeKey";
    szClass = "";

    // Create a key with no class value.
    if (RegDBCreateKeyEx (szKey, szClass) < 0) then
        MessageBox ("RegDBCreateKeyEx failed.", SEVERE);
        abort;
    else
        sprintfBox (INFORMATION, TITLE_TEXT, "%s successfully created.", szKey);
    endif;

    // Call RegDBDeleteKey to delete the key just created.
    if (RegDBDeleteKey (szKey) < 0) then
        MessageBox ("RegDBDeleteKey failed.", SEVERE);
    else
        sprintfBox (INFORMATION, TITLE_TEXT, "%s successfully deleted.", szKey);
    endif;

end;

```

## RegDBDeleteValue



**Project** • For InstallScript MSI and Basic MSI projects, it is recommended that you use the Registry view in InstallShield instead of creating registry keys and values through InstallScript code. Handling all of your registry changes in this way allows for a clean uninstallation through the Windows Installer service.

The **RegDBDeleteValue** function deletes a value from a specific key in the registry. InstallShield assumes that the key specified in szSubKey is a subkey of HKEY\_CLASSES\_ROOT. You must use RegDBSetDefaultRoot to specify another root key.

RegDBDeleteKey is a general registry-related function, designed to work with all registry keys, including those handled by the special registry-related functions. For more information on special registry-related functions, see [Special Registry-Related Functions](#).



**Note** • This function supports the 64-bit parts of the registry by using the REGDB\_OPTION\_WOW64\_64KEY option. For more information, see [REGDB\\_OPTIONS](#).

## Syntax

```
RegDBDeleteValue ( szSubKey, szValue );
```

## Parameters

**Table 29** • RegDBDeleteValue Parameters

Parameter	Description
<b>szSubKey</b>	Specifies the name of the registry key that contains the value name to delete. Separate different levels in the subkey with a double backslash (\\).
<b>szValue</b>	Specifies the name of the value you want to delete.

## Return Values

**Table 30** • RegDBDeleteValue Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully deleted the value.
<b>&lt; 0</b>	Indicates that the function was unable to delete the value.

## Additional Information

- By default, any text that is surrounded by angle brackets—for example, "<my registry entry text>"—in this function's string arguments is interpreted as a text substitution and is processed accordingly. To disable text substitution processing for the string arguments of registry functions, call **Disable** with the `REGISTRYFUNCTIONS_USETEXTSUBS` argument.
- RegDBDeleteValue** does not write anything to the uninstall log file; thus, calling this function does not have any effect how the application is uninstalled.

## RegDBDeleteValue Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the RegDBDeleteValue function.
*
* RegDBDeleteValue is called to delete the value name "Cursive"
* from the following registry key:
*
* "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Fonts".
```

```

*
* Note: Before running this script, set the preprocessor
*       constants so that they reference an existing subkey
*       and value on the target system.
*
\*-----*/

#define SUBKEY "\\Software\\Microsoft\\Windows\\CurrentVersion\\Fonts"
#define VALUE  "Cursive"
#define TITLE  "RegDBDeleteValue Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_RegDBDeleteValue(HWND);

function ExFn_RegDBDeleteValue(hMSI)
    STRING szSubKey, szValue, szTitle;
    NUMBER nReturn;
begin

    // Set the root key.
    RegDBSetDefaultRoot (HKEY_LOCAL_MACHINE);

    // Set the name of the subkey.
    szSubKey = SUBKEY;
    szValue  = VALUE;

    // Delete the subkey.
    nReturn = RegDBDeleteValue (szSubKey, szValue);

    // Report the results of the deletion.
    if (nReturn < 0) then
        MessageBox ("RegDBDeleteValue failed.", SEVERE);
    else
        SprintfBox (INFORMATION, TITLE, "%s successfully deleted.", szValue);
    endif;

end;

```

## RegDBDisconnectRegistry



**Project** • For InstallScript MSI and Basic MSI projects, it is recommended that you use the Registry view in InstallShield instead of creating registry keys and values through InstallScript code. Handling all of your registry changes in this way allows for a clean uninstallation through the Windows Installer service.

The **RegDBDisconnectRegistry** function closes a connection to a remote registry that you established by calling **RegDBConnectRegistry**.

After calling **RegDBDisconnectRegistry**, all calls to the InstallScript registry-related functions affect the local system's registry. For more information on special registry-related functions, see [Special Registry-Related Functions](#).



**Note** • This function supports the 64-bit parts of the registry by using the `REGDB_OPTION_WOW64_64KEY` option. For more information, see [REGDB\\_OPTIONS](#).

Syntax

```
RegDBDisconnectRegistry ( nReserved );
```

Parameters

Table 31 • RegDBDisconnectRegistry Parameters

Parameter	Description
nReserved	Pass zero in this parameter. No other value is allowed.

Return Values

Table 32 • RegDBDisconnectRegistry Return Values

Return Value	Description
0	Indicates that this function successfully closed a connection to the registry on the remote system.
< 0	Indicates that this function failed to close the registry connection.

RegDBDisconnectRegistry Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the RegDBConnectRegistry function and
* RegDBDisconnectRegistry function.
*
* Note: In order for this script to run properly, you must set
*       the preprocessor constants to a valid remote computer
*       with remote administration enabled. Both computers
*       must also have the remote-registry service enabled.
*
*\-----*/

#define REMOTE "IShield_NT1"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"
```

```

export prototype ExFn_RegDBDisConnectRegistry(HWND);

function ExFn_RegDBDisConnectRegistry(hMSI)
    STRING szRemoteMachine, szKey, szTitle, szMsg;
    NUMBER nKeyType, nReturn;
begin

    szTitle          = "RegDBConnectRegistry & RegDBDisConnectRegistry";
    szRemoteMachine = REMOTE;
    nKeyType          = HKEY_LOCAL_MACHINE;
    szMsg             = "Setup will now connect to %s.";

    sprintfBox (INFORMATION, szTitle, szMsg, szRemoteMachine);

    // Connect to the remote computer's registry. All registry-related
    // function calls will now alter only the remote computer.
    nReturn = RegDBConnectRegistry (szRemoteMachine, nKeyType, 0);

    if (nReturn < 0) then
        szMsg = "RegDBConnectRegistry failed.\n\nCould not connect to remote " +
            "system.";
        MessageBox (szMsg, SEVERE);
        abort;
    else
        szMsg = "Successfully connected to %s.";
        sprintfBox (INFORMATION, szTitle, szMsg, szRemoteMachine);
    endif;

    // Create a key on the remote computer.
    szKey = "SOFTWARE\\InstallShield\\Test Key";
    nReturn = RegDBCreateKeyEx(szKey, "");

    if (nReturn < 0) then
        szMsg = "RegDBCreateKeyEx failed.\n\nCould not create key on remote " +
            "machine.";
        MessageBox (szMsg, SEVERE);
    else
        szMsg = "Successfully created %s on %s.";
        sprintfBox (INFORMATION, szTitle, szMsg, szKey, szRemoteMachine);

        // Verify that the key now exists in the remote registry.
        nReturn = RegDBKeyExist(szKey);

        if (nReturn < 0) then
            szMsg = "RegDBKeyExist failed.\n\nRemote key does not exist.";
            MessageBox (szMsg, SEVERE);
        else
            szMsg = "%s exists.";
            sprintfBox (INFORMATION, szTitle, szMsg, szKey);
        endif;
    endif;

    // Delete the key that was created on the remote computer.
    nReturn = RegDBDeleteKey(szKey);

    if (nReturn < 0) then
        MessageBox("RegDBDeleteKey failed.\n\nRemote key could not be deleted.",
            INFORMATION);
    endif;
endfunction

```

```

else
    szMsg = "Successfully deleted %s on %s.";
    sprintfBox(INFORMATION, szTitle, szMsg, szKey, szRemoteMachine);
endif;

// Disconnect from the remote registry. All registry-related functions
// will now alter only the local registry.
nReturn = RegDBDisconnectRegistry(0);

if (nReturn < 0) then
    MessageBox("RegDBDisconnectRegistry failed.\n\nRemote registry still " +
        "connected.", SEVERE);
else
    MessageBox("RegDBDisconnectRegistry successful.\n\nRemote registry " +
        "disconnected.", INFORMATION);
endif;

end;

```

## RegDBGetAppInfo



**Project** • For InstallScript MSI and Basic MSI projects, it is recommended that you use the Registry view in InstallShield instead of creating registry keys and values through InstallScript code. Handling all of your registry changes in this way allows for a clean uninstallation through the Windows Installer service.

The **RegDBGetAppInfo** function retrieves from the registry the value of a particular value name under the application information key of your main application. RegDBGetAppInfo is a special registry-related function, designed to work with certain predefined registry keys.



**Note** • The InstallScript engine currently does not support writing or reading Add or Remove Programs information for a product in the 64-bit part of the registry. Therefore, using the REGDB\_OPTION\_WOW64\_64KEY option with the [REGDB\\_OPTIONS](#) system variable is not supported for this registry function. Enabling the REGDB\_OPTION\_WOW64\_64KEY option has no effect on where registry entries are created by this function.

### Syntax

```
RegDBGetAppInfo ( szName, nvType, svValue, nvSize );
```

## Parameters

**Table 33** • RegDBGetAppInfo Parameters

Parameter	Description
<b>szName</b>	Specifies the value name of the value to retrieve.
<b>nvType</b>	Returns one of the following predefined constants, which identifies the type of data returned in svValue: <ul style="list-style-type: none"> <li>• <b>REGDB_STRING</b>—String variable, no newline characters allowed.</li> <li>• <b>REGDB_STRING_EXPAND</b>—String variable holding an expandable environment variable expression, such as "%MYPATH%".</li> <li>• <b>REGDB_STRING_MULTI</b>—String variable, newline characters allowed.</li> <li>• <b>REGDB_NUMBER</b>—Number expressed as a string and passed in a string variable.</li> <li>• <b>REGDB_BINARY</b>—Binary data stored in a string.</li> </ul>
<b>svValue</b>	Returns the value of the value name specified in szName.
<b>nvSize</b>	Returns the size—in bytes—of the return value.

## Return Values

**Table 34** • RegDBGetAppInfo Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully retrieved the value.
<b>&lt; 0</b>	Indicates that the function was unable to retrieve the value.

## Additional Information

By default, any text that is surrounded by angle brackets—for example, "<my registry entry text>"—in this function's string arguments is interpreted as a text substitution and is processed accordingly. To disable text substitution processing for the string arguments of registry functions, call **Disable** with the REGISTRYFUNCTIONS\_USETEXTSUBS argument.

## RegDBGetAppInfo Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
```

```

*
* Demonstrates the RegDBSetAppInfo and RegDBGetAppInfo functions.
*
* Before calling either of these functions, you must call
* InstallationInfo to set the application information.
*
\*-----*/

#define COMPANY_NAME      "Example_Company"
#define PRODUCT_NAME      "Example_App"
#define PRODUCT_VERSION   "5.0"
#define PRODUCT_KEY       "EXAMPLE.EXE"
#define DEINSTALL_KEY     "Example_DeinstKey"
#define UNINSTALL_NAME    "Example_App_5.0"
#define DEFAULT_LOG_PATH  "EXAMPLE"
#define TITLE             "RegDBGetAppInfo"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_RegDBGetAppInfo(HWND);

function ExFn_RegDBGetAppInfo(hMSI)
    STRING  szStrName, szStrValue, svStrValue, szTitle, szMsg, svLogFile;
    NUMBER  nvSize, nvType;
begin
    // Set the root key.
    RegDBSetDefaultRoot (HKEY_LOCAL_MACHINE);

    // Set the name to be used with REGDB_STRING.
    szStrName = "ExampleStringValue";
    szStrValue = "ExampleStringSetting";

    // Set up the application information prior to using RegDBSetAppInfo
    // and RegDBGetAppInfo.
    InstallationInfo (COMPANY_NAME, PRODUCT_NAME, PRODUCT_VERSION, PRODUCT_KEY);
    DeinstallStart(DEFAULT_LOG_PATH, svLogFile, DEINSTALL_KEY, 0);

    // Set value of type REGDB_STRING
    if (RegDBSetAppInfo (szStrName, REGDB_STRING, szStrValue, -1) < 0) then
        MessageBox ("Failed to set key and value of REGDB_STRING type.", SEVERE);
        abort;
    endif;

    // RegDBGetAppInfo is called to return the values and compare all the setup
    // parameters.
    if (RegDBGetAppInfo (szStrName, nvType, svStrValue, nvSize) < 0) then
        MessageBox ("Failed to get application information value.", SEVERE);
        abort;
    else
        // Check to see if the value retrieved is the same as the value set.
        if (nvType != REGDB_STRING) then
            MessageBox ("Type comparison failed.", WARNING);
        endif;

        if (szStrValue != svStrValue) then
            MessageBox ("Sub key value comparison Failed.", WARNING);
        else

```



```

        szMsg = "Set values: %s = %s\n\nReturn values: %s = %s";
        sprintfBox (INFORMATION, TITLE, szMsg, szStrName, szStrValue,
                    szStrName, svStrValue);
    endif;

    if (nvSize != StrLength(szStrValue)) then
        MessageBox ("Size Comparison failed.", WARNING);
    else
        szMsg = "Size in bytes entered: %d\n\nSize in bytes returned: %d";
        sprintfBox (INFORMATION, TITLE, szMsg,
                    nvSize, StrLength(szStrValue) + 1);
    endif;
endif;

end;

```

## RegDBGetDefaultRoot



**Project** • For InstallScript MSI and Basic MSI projects, it is recommended that you use the Registry view in InstallShield instead of creating registry keys and values through InstallScript code. Handling all of your registry changes in this way allows for a clean uninstallation through the Windows Installer service.

The **RegDBGetDefaultRoot** function returns the root key that is used by the general registry-related functions.



**Note** • This function supports the 64-bit parts of the registry by using the `REGDB_OPTION_WOW64_64KEY` option. For more information, see [REGDB\\_OPTIONS](#).

### Syntax

```
RegDBGetDefaultRoot ( );
```

### Parameters

None.

### Return Values

- HKEY\_CLASSES\_ROOT
- HKEY\_CURRENT\_USER
- HKEY\_LOCAL\_MACHINE
- HKEY\_USERS
- HKEY\_CURRENT\_CONFIG
- HKEY\_DYN\_DATA
- HKEY\_USER\_SELECTABLE



**Note** • The return value `HKEY_USER_SELECTABLE` indicates that a subsequent registry function call uses `HKEY_LOCAL_MACHINE` as the root key if the `ALLUSERS` system variable is non-zero when the function is called, or uses `HKEY_CURRENT_USER` as the root key if `ALLUSERS` is `FALSE` when the function is called.

## RegDBGetItem



**Project** • For InstallScript MSI and Basic MSI projects, it is recommended that you use the Registry view in InstallShield instead of creating registry keys and values through InstallScript code. Handling all of your registry changes in this way allows for a clean uninstallation through the Windows Installer service.

The **RegDBGetItem** function retrieves values under the per-application paths key or the application uninstallation key, depending on the value of `nItem`. **RegDBGetItem** is a special registry-related function, designed to work with certain predefined registry keys.



**Note** • The InstallScript engine currently does not support writing or reading Add or Remove Programs information for a product in the 64-bit part of the registry. Therefore, using the `REGDB_OPTION_WOW64_64KEY` option with the `REGDB_OPTIONS` system variable is not supported for this registry function. Enabling the `REGDB_OPTION_WOW64_64KEY` option has no effect on where registry entries are created by this function.

### Syntax

```
RegDBGetItem ( nItem, svValue );
```

## Parameters

**Table 35 •** RegDBGetItem Parameters

Parameter	Description
<b>nItem</b>	<p>Specifies which item to retrieve. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>REGDB_APPPATH</b>—The data of the Path value under the per application paths key.</li> <li>• <b>REGDB_APPPATH_DEFAULT</b>—The data of the DefaultPath value under the per application paths key.</li> <li>• <b>REGDB_UNINSTALL_COMMENTS</b>—The data of the Comments value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_CONTACT</b>—The data of the Contact value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_DISPLAY_VERSION</b>—The data of the DisplayVersion value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_DISPLAYICON</b>—The data of the DisplayIcon value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_HELPLINK</b>—The data of the HelpLink value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_HELPTELEPHONE</b>—The data of the HelpTelephone value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_INSTALLDATE</b>—The data of the InstallDate value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_INSTALLLOC</b>—The data of the InstallLocation value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_INSTALLSOURCE</b>—The data of the InstallSource value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_LANGUAGE</b>—The data of the Language value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_LOGFILE</b>—The data of the LogFile value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_MAINT_OPTION</b>—The data of the LogMode value under the uninstallation key.</li> </ul>

**Table 35 •** RegDBGetItem Parameters (cont.)

Parameter	Description
<b>nItem (cont.)</b>	<ul style="list-style-type: none"> <li>● <b>REGDB_UNINSTALL_MAJOR_VERSION</b>—The data of the VersionMajor value under the uninstallation key, if it is present. If it is not present, the function checks the data of the MajorVersion value under the uninstallation key.</li> <li>● <b>REGDB_UNINSTALL_MAJOR_VERSION_OLD</b>—The data of the MajorVersion value under the uninstallation key.</li> <li>● <b>REGDB_UNINSTALL_MINOR_VERSION</b>—The data of the VersionMinor value under the uninstallation key, if it is present. If it is not present, the function checks the data of the MinorVersion value under the uninstallation key.</li> <li>● <b>REGDB_UNINSTALL_MINOR_VERSION_OLD</b>—The data of the MinorVersion value under the uninstallation key.</li> <li>● <b>REGDB_UNINSTALL_MODIFYPATH</b>—The data of the ModifyPath value under the uninstallation key.</li> <li>● <b>REGDB_UNINSTALL_NAME</b>—The data of the DisplayName value under the uninstallation key. When this constant is used, szValue specifies the application name shown in the list of uninstallable applications in the Control Panel.</li> <li>● <b>REGDB_UNINSTALL_NOMODIFY</b>—The data of the NoModify value under the uninstallation key.</li> <li>● <b>REGDB_UNINSTALL_NOREMOVE</b>—The data of the NoRemove value under the uninstallation key.</li> <li>● <b>REGDB_UNINSTALL_NOREPAIR</b>—The data of the NoRepair value under the uninstallation key.</li> <li>● <b>REGDB_UNINSTALL_PRODUCTGUID</b>—The data of the ProductGuid value under the uninstallation key.</li> <li>● <b>REGDB_UNINSTALL_PRODUCTID</b>—The data of the ProductId value under the uninstallation key.</li> <li>● <b>REGDB_UNINSTALL_PUBLISHER</b>—The data of the Publisher value under the uninstallation key.</li> <li>● <b>REGDB_UNINSTALL_README</b>—The data of the Readme value under the uninstallation key.</li> <li>● <b>REGDB_UNINSTALL_REGCOMPANY</b>—The data of the RegCompany value under the uninstallation key.</li> <li>● <b>REGDB_UNINSTALL_REGOWNER</b>—The data of the RegOwner value under the uninstallation key.</li> </ul>

**Table 35** • RegDBGetItem Parameters (cont.)

Parameter	Description
<b>nItem (cont.)</b>	<ul style="list-style-type: none"> <li>• <b>REGDB_UNINSTALL_STRING</b>—The data of the UninstallString value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_SYSTEMCOMPONENT</b>—The data of the SystemComponent value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_URLINFOABOUT</b>—The data of the URLInfoAbout value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_URLUPDATEINFO</b>—The data of the URLUpdateInfo value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_VERSION</b>—The data of the Version value under the uninstallation key. When this constant is used, svValue returns the installed product's version number expressed as a string, for example, "16777216".</li> <li>• <b>REGDB_WINCURRVER_REGORGANIZATION</b>—The data of the RegisteredOrganization value under the current version key.</li> <li>• <b>REGDB_WINCURRVER_REGOWNER</b>—The data of the RegisteredOwner value under the current version key.</li> </ul>
<b>svValue</b>	Returns the value of the item.

## Return Values

**Table 36** • RegDBGetItem Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully retrieved the value of the item.
<b>&lt; 0</b>	Indicates that the function was unable to retrieve the value of the item.

## Additional Information

By default, any text that is surrounded by angle brackets—for example, "<my registry entry text>"—in this function's string arguments is interpreted as a text substitution and is processed accordingly. To disable text substitution processing for the string arguments of registry functions, call **Disable** with the **REGISTRYFUNCTIONS\_USETEXTSUBS** argument.



**Project** • In an InstallScript installation, calling **RegDBSetItem** with **REGDB\_APPPATH** or **REGDB\_APPPATH\_DEFAULT** before calling **CreateInstallationInfo** has no net effect. This is because **CreateInstallationInfo** overwrites registry information that is created by **RegDBSetItem**, thus nullifying the call. Calling **RegDBSetItem** with any other constant before calling **MaintenanceStart** has no net effect. This is because **MaintenanceStart** overwrites the registry data created by **RegDBSetItem**, thus nullifying the call. (In an event-based script, **CreateInstallationInfo** and **MaintenanceStart** are called in the default **OnMoveData** event handler code.)

In an InstallScript MSI installation, the uninstallation information is created during file transfer by the Windows Installer. Therefore, if you call **RegDBSetItem** in an InstallScript MSI installation, you should call it only after file transfer.

## RegDBGetItem Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the RegDBSetItem and RegDBGetItem functions.
*
* This script sets several registry keys; then it gets those
* keys and displays their current values.
*
\*-----*/

#define COMPANY_NAME      "ExampleCompany"
#define PRODUCT_NAME      "ExampleProduct"
#define VERSION_NUMBER    "5.00.00"
#define PRODUCT_KEY       "EXAMPLE.EXE"
#define DEINST_KEY        "ExampleDeinstKey"
#define APP_DEF_LOG_PATH  "C:\\EXAMPLE\\TEMP"
#define APP_PATH           "C:\\EXAMPLE"
#define APP_DEF_PATH      "C:\\EXAMPLE\\TARGET"
#define UNINSTALL_NAME    "ExampleUninstallName"
#define TITLE              "RegDBSetItem Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_RegDBGetItem(HWND);

function ExFn_RegDBGetItem(hMSI)
    STRING svLogFile, svValue, szTitle;
begin

    // Set the root key.
    RegDBSetDefaultRoot (HKEY_LOCAL_MACHINE);

    // Set installation and uninstallation information in
    // order to call RegDBSetItem and RegDBGetItem.
    InstallationInfo (COMPANY_NAME, PRODUCT_NAME, VERSION_NUMBER, PRODUCT_KEY);
    DeinstallStart (APP_DEF_LOG_PATH, svLogFile, DEINST_KEY, 0);

    // Set the value of the application path key in the
    // registry to the value of szAppPath.
    if (RegDBSetItem (REGDB_APPPATH, APP_PATH) < 0) then
        MessageBox ("Unable to set application path key.", SEVERE);
    else
        sprintfBox (INFORMATION, TITLE, "RegDBSetItem set the application " +
            "path key to %s.", APP_PATH);
```

```

endif;

// Set the value of the application default path key in
// the registry to the value of szAppDefPath.
if (RegDBSetItem(REGDB_APPPATH_DEFAULT, APP_DEF_PATH) < 0) then
    MessageBox ("Unable to set application default path key.", SEVERE);
else
    sprintfBox (INFORMATION, TITLE, "RegDBSetItem set the application " +
        "default path key to %s.", APP_DEF_PATH);
endif;

// Set the value of the uninstall name key in the
// registry to the value of szUninstallName.
if (RegDBSetItem (REGDB_UNINSTALL_NAME, UNINSTALL_NAME) < 0) then
    MessageBox ("Unable to set uninstall name key.", SEVERE);
else
    sprintfBox (INFORMATION, TITLE, "RegDBSetItem set the uninstall " +
        "name key to %s.", UNINSTALL_NAME);
endif;

// Set up title parameter for call to sprintfBox.
szTitle = "RegDBGetItem";

// Get the value of the application path key from the registry.
if (RegDBGetItem (REGDB_APPPATH, svValue) < 0) then
    MessageBox ("Unable to get value of application path key.", SEVERE);
else
    sprintfBox (INFORMATION, TITLE, "RegDBGetItem retrieved the value " +
        "of the application path key: %s.", svValue);
endif;

// Get the value of the application default path key in
// the registry.
if (RegDBGetItem (REGDB_APPPATH_DEFAULT, svValue) < 0) then
    MessageBox ("Unable to get application default path key", SEVERE);
else
    sprintfBox (INFORMATION, TITLE, "RegDBGetItem retrieved the value " +
        "of the application default path key: %s.", svValue);
endif;

// Get the value of the uninstall name key from the registry.
if (RegDBGetItem (REGDB_UNINSTALL_NAME, svValue) < 0) then
    MessageBox ("Unable to get application uninstall name key.", SEVERE);
else
    sprintfBox (INFORMATION, TITLE, "RegDBGetItem retrieved the value " +
        "of the uninstallation name key: %s.", svValue);
endif;

end;

```

## RegDBGetKeyValueEx



**Project** • For InstallScript MSI and Basic MSI projects, it is recommended that you use the Registry view in InstallShield instead of creating registry keys and values through InstallScript code. Handling all of your registry changes in this way allows for a clean uninstallation through the Windows Installer service.

The **RegDBGetKeyValueEx** function retrieves the value of a particular value name under a specified key in the registry. By default, InstallShield assumes this key is a subkey of HKEY\_CLASSES\_ROOT. You can use **RegDBSetDefaultRoot** to specify another root key.

**RegDBGetKeyValueEx** is a general registry-related function, designed to work with all registry keys, including those handled by the special registry-related functions.



---

**Note** • This function supports the 64-bit parts of the registry by using the `REGDB_OPTION_WOW64_64KEY` option. For more information, see [REGDB\\_OPTIONS](#).


## Syntax

```
RegDBGetKeyValueEx ( szKey, szName, nvType, svValue, nvSize );
```



## Parameters

**Table 37** • RegDBGetKeyValueEx Parameters

Parameter	Description
<b>szKey</b>	Specifies the name of the key whose value is to be retrieved. Separate different levels in the subkey with a double backslash (\\).
<b>szName</b>	Specifies the value name under szKey of the value to retrieve. To retrieve the default value of the key, pass a null string ("").
<b>nvType</b>	<p>Returns one of the following predefined constants, which identifies the type of data returned in svValue:</p> <ul style="list-style-type: none"> <li>• <b>REGDB_STRING</b>—String variable, no newline characters allowed.</li> <li>• <b>REGDB_STRING_EXPAND</b>—String variable holding an expandable environment variable expression such as "%MYPATH%".</li> <li>• <b>REGDB_STRING_MULTI</b>—String variable, newline characters allowed.</li> <li>• <b>REGDB_NUMBER</b>—Number expressed as a string and passed in a string variable.</li> <li>• <b>REGDB_BINARY</b>—Binary data stored in a string.</li> </ul> <p> <b>Note</b> • When data type REGDB_STRING_MULTI is retrieved, use <b>StrGetTokens</b> with null string ("" ) to parse the multiple null terminated strings into a list of strings. That is, if svValue has the resulting multiple strings after a call to <b>RegDBGetKeyValueEx</b>, StrGetTokens( listID, svValue, "" ) can be used to parse the strings and put them in a string list (listID).</p>
<b>svValue</b>	Returns the value that was specified by szKey and svName. Note that a number value is returned as a string.
<b>nvSize</b>	Returns the size—in bytes—of the value returned in svValue.

## Return Values

**Table 38** • RegDBGetKeyValueEx Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully retrieved the value.
<b>&lt; 0</b>	Indicates that the function was unable to retrieve the value.

## RegDBGetKeyValueEx Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function,

*execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.*

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the RegDBSetKeyValueEx and RegDBGetKeyValueEx
 * functions.
 *
 * RegDBCreateKeyEx is called to create a test subkey in the
 * HKEY_CLASSES_ROOT key. The value of a key is set in integer
 * form using the REGDB_NUMBER option of the RegDBSetKeyValueEx
 * function. After this value is set, it is retrieved using
 * the RegDBGetKeyValueEx function and verified.
 *
\*-----*/

#define TITLE "RegDBSetKeyValueEx & RegDBGetKeyValueEx"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_RegDBGetKeyValueEx(HWND);

function ExFn_RegDBGetKeyValueEx(hMSI)
    STRING szKey, szNumName, szNumValue, svNumValue, szTitle, szMsg;
    NUMBER nType, nSize, nvType, nvSize;
begin

    // Create a key to test.
    szKey = "TestKey";

    if (RegDBCreateKeyEx (szKey, "") < 0) then
        MessageBox ("RegDBCreateKeyEx failed.", SEVERE);
        abort;
    endif;

    // Set up parameters for call to RegDBSetKeyValueEx.
    szNumName = "TestValue";
    szNumValue = "12345";
    nType      = REGDB_NUMBER;
    nSize      = -1;

    // Set a key name and a value associated with it.
    if (RegDBSetKeyValueEx (szKey, szNumName, nType, szNumValue,
                           nSize) < 0) then
        MessageBox ("RegDBSetKeyValueEx failed.", SEVERE);
        abort;
    else
        // Display what RegDBSetKeyValueEx has done.
        szMsg = "%s set to: %s";
        sprintfBox (INFORMATION, TITLE, szMsg, szNumName, szNumValue);
    endif;

    // Retrieve key value information.
    if (RegDBGetKeyValueEx (szKey, szNumName, nvType, svNumValue,
                           nvSize) < 0) then
        MessageBox ("RegDBGetKeyValueEx failed.", SEVERE);
    else

```

```

    // Check to see if the value returned is the same as the value set.
    if (nvType != REGDB_NUMBER) then
        MessageBox ("Type comparison failed.", SEVERE);
    endif;

    if (svNumValue != szNumValue) then
        MessageBox ("Subkey value comparison failed.", SEVERE);
    endif;

    // Display what RegDBGetKeyValueEx retrieved.
    szMsg = "%s has value: %s\n\nThis data is %d bytes.";
    sprintfBox (INFORMATION, TITLE, szMsg, szNumName, svNumValue, nvSize);
endif;

// Delete the created test key.
if (RegDBDeleteKey (szKey) < 0) then
    MessageBox ("RegDBDeleteKey failed.", SEVERE);
endif;

end;

```

## RegDBGetUninstCmdLine



**Project** • For InstallScript MSI and Basic MSI projects, it is recommended that you use the Registry view in InstallShield instead of creating registry keys and values through InstallScript code. Handling all of your registry changes in this way allows for a clean uninstallation through the Windows Installer service.

The **RegDBGetUninstCmdLine** function gets the registered command line for the uninstallation that is specified by szUninstallKey and returns the command line in svUninstCmdLine.



**Note** • The InstallScript engine currently does not support writing or reading Add or Remove Programs information for a product in the 64-bit part of the registry. Therefore, using the REGDB\_OPTION\_WOW64\_64KEY option with the [REGDB\\_OPTIONS](#) system variable is not supported for this registry function. Enabling the REGDB\_OPTION\_WOW64\_64KEY option has no effect on where registry entries are created by this function.

### Syntax

```
RegDBGetUninstCmdLine ( szUninstallKey, svUninstCmdLine );
```

## Parameters

**Table 39** • RegDBGetUninstCmdLine Parameters

Parameter	Description
<b>szUninstallKey</b>	Specifies the name of a subkey under the target registry's <root key>\Software\Microsoft\Windows\CurrentVersion\Uninstall key; the function first checks for the subkey under the root key HKEY_CURRENT_USER, and if it does not find the subkey there, it checks under HKEY_LOCAL_MACHINE. For setups created with InstallShield Professional 5.53 or earlier, this is typically the name of the application; for setups created with InstallShield Professional 6.0 or later, this is the application's product GUID including the surrounding braces {}.
<b>svUninstCmdLine</b>	Returns the uninstallation command line that is specified in szUninstallKey's UninstallString value's data.

## Return Values

**Table 40** • RegDBGetUninstCmdLine Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	The function successfully got the command line.
<b>&lt; ISERR_SUCCESS</b>	The function failed to get the command line.

## Additional Information

By default, any text that is surrounded by angle brackets—for example, "<my registry entry text>"—in this function's string arguments is interpreted as a text substitution and is processed accordingly. To disable text substitution processing for the string arguments of registry functions, call **Disable** with the REGISTRYFUNCTIONS\_USETEXTSUBS argument.

# RegDBKeyExist



**Project** • For InstallScript MSI and Basic MSI projects, it is recommended that you use the Registry view in InstallShield instead of creating registry keys and values through InstallScript code. Handling all of your registry changes in this way allows for a clean uninstallation through the Windows Installer service.

The **RegDBKeyExist** function checks for the existence of a specific key in the registry. By default, InstallShield assumes this key is a subkey of HKEY\_CLASSES\_ROOT. If you want to use a different main key, use RegDBSetDefaultRoot to specify another root key.

RegDBKeyExist is a general registry-related function, designed to work with all registry keys, including those handled by the special registry-related functions. For more information on special registry-related functions, see [Special Registry-Related Functions](#).



**Note** • This function supports the 64-bit parts of the registry by using the `REGDB_OPTION_WOW64_64KEY` option. For more information, see [REGDB\\_OPTIONS](#).

## Syntax

```
RegDBKeyExist ( szSubKey );
```

## Parameters

**Table 41** • RegDBKeyExist Parameters

Parameter	Description
<b>szSubKey</b>	Specifies the name of the key to find. You do not have to include the <code>HKEY_CLASSES_ROOT</code> key (or another root key you specified) in this parameter. Separate different levels in the subkey with a double backslash ( <code>\\</code> ).

## Return Values

**Table 42** • RegDBKeyExist Return Values

Return Value	Description
<b>1</b>	Indicates that the function found the key name in the registry.
<b>&lt; 0</b>	Indicates that the function was unable to find the key name in the registry.

This function never returns zero (0).

## Additional Information

By default, any text that is surrounded by angle brackets—for example, "<my registry entry text>"—in this function's string arguments is interpreted as a text substitution and is processed accordingly. To disable text substitution processing for the string arguments of registry functions, call **Disable** with the `REGISTRYFUNCTIONS_USETEXTSUBS` argument.

## RegDBKeyExist Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
```

```

* InstallShield Example Script
*
* Demonstrates the RegDBCreateKeyEx and RegDBKeyExist functions.
*
* First, RegDBCreateKeyEx is called to create a subkey with no
* class value in the HKEY_CLASSES_ROOT key. Then, RegDBKeyExist
* is then called to check if the key was created.
*
* RegDBCreateKeyEx is called again to create a multi-level subkey
* with a class value associated with it under HKEY_CLASSES_ROOT.
* Then RegDBKeyExist is called again to check for the existence
* of the new key.
*
\*-----*/

#define TITLE_TEXT "RegDBCreateKeyEx & RegDBKeyExist"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_RegDBKeyExist(HWND);

function ExFn_RegDBKeyExist(hMSI)
    STRING szKey, szClass, szKeyRoot, szMsg, svLogFile;
    NUMBER nResult1, nResult2;
begin

    // Create a key with no class value.
    szKey = "CreateKeyExample";
    szClass = "";

    if (RegDBCreateKeyEx(szKey, szClass) < 0) then
        MessageBox ("First call to RegDBCreateKeyEx failed.", SEVERE);
        abort;
    else
        sprintfBox (INFORMATION, TITLE_TEXT, "Successfully created: %s", szKey);

        // Check to see if the key just created exists.
        if (RegDBKeyExist (szKey) < 0) then
            MessageBox ("First call to RegDBKeyExist failed.", SEVERE);
        else
            sprintfBox (INFORMATION, TITLE_TEXT, "%s exists.", szKey);
        endif;
    endif;

    if (RegDBDeleteKey (szKey) < 0) then
        MessageBox ("RegDBDeleteKey failed.", SEVERE);
    endif;

    // Create a key with more than one sublevel and a class value.
    szKey = "ShareWare\\Games\\CoolChess";
    szClass = "LastPlayed";
    szKeyRoot = "ShareWare";

    if (RegDBCreateKeyEx(szKey, szClass) < 0) then
        MessageBox ("Second call to RegDBCreateKeyEx failed.", SEVERE);
        abort;
    else
        sprintfBox (INFORMATION, TITLE_TEXT, "Successfully created: %s", szKey);
    endif;
endfunction

```

```

    // Check if the newly created multi-level key exists.
    if (RegDBKeyExist (szKeyRoot) < 0) then
        MessageBox ("Second call to RegDBKeyExist failed.", SEVERE);
    else
        sprintfBox (INFORMATION, TITLE_TEXT, "%s exists.", szKey);
    endif;
endif;

if (RegDBDeleteKey (szKey) < 0) then
    MessageBox ("RegDBDeleteKey failed.", SEVERE);
endif;

end;

```

## RegDBQueryKey



**Project** • For InstallScript MSI and Basic MSI projects, it is recommended that you use the Registry view in InstallShield instead of creating registry keys and values through InstallScript code. Handling all of your registry changes in this way allows for a clean uninstallation through the Windows Installer service.

The **RegDBQueryKey** function allows users to query a key for its subkeys and value names. The keys can be enumerated dynamically at run time using this function. RegDBQueryKey is a general registry-related function, designed to work with all registry keys, including those handled by the special registry-related functions.



**Note** • This function supports the 64-bit parts of the registry by using the `REGDB_OPTION_WOW64_64KEY` option. For more information, see [REGDB\\_OPTIONS](#).

### Syntax

```
RegDBQueryKey ( szSubKey, nItem, listResults );
```

## Parameters

**Table 43 •** RegDBQueryKey Parameters

Parameter	Description
<b>szSubKey</b>	Specifies subkeys under one of the root keys that were set by a previous call to <a href="#">RegDBSetDefaultRoot</a> . Use backslashes to specify deeper levels of subkeys. To retrieve the root key, pass a null string ("").
<b>nItem</b>	Specifies which items should be placed in the list. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>REGDB_KEYS</b>—The string list returned in listResults will contain a list of all the subkeys under this key.</li> <li>• <b>REGDB_NAMES</b>—The string list returned in listResults will contain the names of all named values for this key.</li> </ul>
<b>listResults</b>	Returns the results of the query in a string list. The list identified by listResults must already have been initialized by a call to <a href="#">ListCreate</a> .

## Return Values

**Table 44 •** RegDBQueryKey Return Values

Return Value	Description
<b>0</b>	Indicates function was successful.
<b>&lt; 0</b>	Indicates function failed.

## Additional Information

By default, any text that is surrounded by angle brackets—for example, "<my registry entry text>"—in this function's string arguments is interpreted as a text substitution and is processed accordingly. To disable text substitution processing for the string arguments of registry functions, call **Disable** with the `REGISTRYFUNCTIONS_USETEXTSUBS` argument.

## RegDBQueryKey Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the RegDBQueryKey function.
```



```

*
* First, RegDBQueryKey is called to query the subkeys under the
* key KEY1. The list returned by RegDBQueryKey is displayed
* in a dialog.
*
* Then RegDBQueryKey is called to query the subkeys under the
* key KEY2. This list is also displayed in a dialog.
*
\*-----*/

#define KEY1  "SOFTWARE"
#define KEY2  "SOFTWARE\\Microsoft"
#define TITLE "RegDBQueryKey Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_RegDBQueryKey(HWND);

function ExFn_RegDBQueryKey(hMSI)
    STRING szMsg;
    NUMBER nReturn, nItem;
    LIST listSubKeys, listNames;
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Create the lists to hold values returned by RegDBQueryKey.
    listSubKeys = ListCreate(STRINGLIST);
    listNames = ListCreate(STRINGLIST);

    if ((listNames = LIST_NULL) || (listSubKeys = LIST_NULL)) then
        MessageBox ("Unable to create necessary lists.", SEVERE);
        abort;
    endif;

    RegDBSetDefaultRoot(HKEY_LOCAL_MACHINE);

    // Get the list of subkeys.
    nReturn = RegDBQueryKey(KEY1, REGDB_KEYS, listSubKeys );

    if (nReturn < 0) then
        MessageBox("First call to RegDBQueryKey failed.", SEVERE);
    else
        szMsg = "Subkeys under " + KEY1 + " key:";
        SdShowInfoList(TITLE, szMsg, listSubKeys );
    endif;

    // Get the list of subkeys.
    nReturn = RegDBQueryKey(KEY2, REGDB_NAMES, listNames);

    if (nReturn < 0) then
        MessageBox("Second call to RegDBQueryKey failed.", SEVERE);
    else
        szMsg = "Named values under " + KEY2 + " key";
        SdShowInfoList(TITLE, szMsg, listNames);
    endif;
end;

```

```
// Remove the lists from memory.  
ListDestroy (listNames);  
ListDestroy (listSubKeys );  
  
end;
```

## RegDBQueryKeyCount



---

**Project** • For InstallScript MSI and Basic MSI projects, it is recommended that you use the Registry view in InstallShield instead of creating registry keys and values through InstallScript code. Handling all of your registry changes in this way allows for a clean uninstallation through the Windows Installer service.

The **RegDBQueryKeyCount** function returns the number of subkeys or values under szKey. nItem specifies whether subkeys or values are counted.



---

**Note** • This function supports the 64-bit parts of the registry by using the REGDB\_OPTION\_WOW64\_64KEY option. For more information, see [REGDB\\_OPTIONS](#).

### Syntax

```
RegDBQueryKeyCount ( szKey, nItem );
```

## Parameters

**Table 45** • RegDBQueryKeyCount Parameters

Parameter	Description
<b>szKey</b>	Specifies a key under the root key that was set by a previous call to <a href="#">RegDBSetDefaultRoot</a> . Use backslashes to specify deeper levels of subkeys. To specify the root key, pass a null string ("").
<b>nItem</b>	Specifies which items should be counted. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>REGDB_KEYS</b>—The number of subkeys is returned.</li> <li>• <b>REGDB_NAMES</b>—The number of values is returned.</li> </ul>

## Return Values

**Table 46** • RegDBQueryKeyCount Return Values

Return Value	Description
<b>X</b>	The number of items of the specified type under the specified key.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function could not determine the number of items.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

## Additional Information

- **RegDBQueryKeyCount** is a general registry-related function, designed to work with all registry keys, including those handled by the special registry-related functions. For more information on special registry-related functions, see [Special Registry-Related Functions](#).
- By default, any text that is surrounded by angle brackets—for example, "<my registry entry text>"—in this function's string arguments is interpreted as a text substitution and is processed accordingly. To disable text substitution processing for the string arguments of registry functions, call **Disable** with the **REGISTRYFUNCTIONS\_USETEXTSUBS** argument.

# RegDBQueryStringMultiStringCount



**Project** • For InstallScript MSI and Basic MSI projects, it is recommended that you use the Registry view in InstallShield instead of creating registry keys and values through InstallScript code. Handling all of your registry changes in this way allows for a clean uninstallation through the Windows Installer service.

The **RegDBQueryStringMultiStringCount** function returns the number of strings contained in the multistring value specified by szValue under the key specified by szKey.



**Note** • This function supports the 64-bit parts of the registry by using the `REGDB_OPTION_WOW64_64KEY` option. For more information, see [REGDB\\_OPTIONS](#).

Syntax

```
RegDBQueryStringMultiStringCount ( szKey, szValue );
```

Parameters

Table 47 • RegDBQueryStringMultiStringCount Parameters

Parameter	Description
szKey	Specifies a key under the root key that was set by a previous call to <a href="#">RegDBSetDefaultRoot</a> . Use backslashes to specify deeper levels of subkeys. To specify the root key, pass a null string ("").
szValue	Specifies the multistring value under szKey to check.

Return Values

Table 48 • RegDBQueryStringMultiStringCount Return Values

Return Value	Description
X	The number of substrings in the specified multistring value under the specified key.
< ISERR_SUCCESS	Indicates that the function could not determine the number of substrings.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

Additional Information

- **RegDBQueryStringMultiStringCount** is a general registry-related function, designed to work with all registry keys, including those handled by the special registry-related functions. For more information on special registry-related functions, see [Special Registry-Related Functions](#).
- By default, any text that is surrounded by angle brackets—for example, "<my registry entry text>"—in this function's string arguments is interpreted as a text substitution and is processed accordingly. To disable text substitution processing for the string arguments of registry functions, call **Disable** with the `REGISTRYFUNCTIONS_USETEXTSUBS` argument.

# RegDBSetAppInfo



**Project** • For InstallScript MSI and Basic MSI projects, it is recommended that you use the Registry view in InstallShield instead of creating registry keys and values through InstallScript code. Handling all of your registry changes in this way allows for a clean uninstallation through the Windows Installer service.

The **RegDBSetAppInfo** function sets the value of a particular value name under the application information key in the registry. RegDBSetAppInfo is a special registry-related function, designed to work with certain predefined registry keys.



**Note** • The InstallScript engine currently does not support writing or reading Add or Remove Programs information for a product in the 64-bit part of the registry. Therefore, using the `REGDB_OPTION_WOW64_64KEY` option with the [REGDB\\_OPTIONS](#) system variable is not supported for this registry function. Enabling the `REGDB_OPTION_WOW64_64KEY` option has no effect on where registry entries are created by this function.

## Syntax

```
RegDBSetAppInfo ( szName, nType, szValue, nSize );
```

## Parameters

**Table 49** • RegDBSetAppInfo Parameters

Parameter	Description
<b>szName</b>	Specifies the value name whose information is to be set.
<b>nType</b>	<p>Specifies the type of data you are setting. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>REGDB_STRING</b>—String variable, no newline characters allowed.</li> <li>• <b>REGDB_STRING_EXPAND</b>—String variable holding an expandable environment variable expression, such as "%MYPATH%".</li> <li>• <b>REGDB_STRING_MULTI</b>—String variable, newline characters allowed.</li> <li>• <b>REGDB_NUMBER</b>—Number expressed as a string and passed in a string variable.</li> <li>• <b>REGDB_BINARY</b>—Binary data stored in a string.</li> </ul>
<b>szValue</b>	Specifies the value to set for the value name.
<b>nSize</b>	Specifies the size, in bytes, of the data passed to RegDBSetAppInfo. Pass -1 in this parameter to indicate that InstallShield should determine the size of the data.

## Return Values

**Table 50** • RegDBSetAppInfo Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully assigned the value to the value name.
<b>&lt; 0</b>	Indicates that the function was unable to assign the value.

## Additional Information

By default, any text that is surrounded by angle brackets—for example, "<my registry entry text>"—in this function's string arguments is interpreted as a text substitution and is processed accordingly. To disable text substitution processing for the string arguments of registry functions, call **Disable** with the **REGISTRYFUNCTIONS\_USETEXTSUBS** argument.

## RegDBSetAppInfo Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the RegDBSetAppInfo and RegDBGetAppInfo functions.
 *
 * Before calling either of these functions, you must call
 * InstallationInfo to set the application information.
 *
 \*-----*/

#define COMPANY_NAME      "Example_Company"
#define PRODUCT_NAME      "Example_App"
#define PRODUCT_VERSION   "5.0"
#define PRODUCT_KEY       "EXAMPLE.EXE"
#define DEINSTALL_KEY     "Example_DeinstKey"
#define UNINSTALL_NAME    "Example_App_5.0"
#define DEFAULT_LOG_PATH  "EXAMPLE"
#define TITLE              "RegDBGetAppInfo"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_RegDBSetAppInfo(HWND);

function ExFn_RegDBSetAppInfo(hMSI)
    STRING  szStrName, szStrValue, svStrValue, szTitle, szMsg, svLogFile;
    NUMBER  nvSize, nvType;
begin

    // Set the root key.
    RegDBSetDefaultRoot (HKEY_LOCAL_MACHINE);

    // Set the name to be used with REGDB_STRING.
    szStrName = "ExampleStringValue";
    szStrValue = "ExampleStringSetting";

    // Set up the application information prior to using RegDBSetAppInfo
    // and RegDBGetAppInfo.
    InstallationInfo (COMPANY_NAME, PRODUCT_NAME, PRODUCT_VERSION, PRODUCT_KEY);
    DeinstallStart(DEFAULT_LOG_PATH, svLogFile, DEINSTALL_KEY, 0);

    // Set value of type REGDB_STRING
    if (RegDBSetAppInfo (szStrName, REGDB_STRING, szStrValue, -1) < 0) then
        MessageBox ("Failed to set key and value of REGDB_STRING type.", SEVERE);
        abort;
    endif;

    // RegDBGetAppInfo is called to return the values and compare all the setup
    // parameters.
    if (RegDBGetAppInfo (szStrName, nvType, svStrValue, nvSize) < 0) then
```

```

        MessageBox ("Failed to get application information value.", SEVERE);
        abort;
    else
        // Check to see if the value retrieved is the same as the value set.
        if (nvType != REGDB_STRING) then
            MessageBox ("Type comparison failed.", WARNING);
        endif;

        if (szStrValue != svStrValue) then
            MessageBox ("Sub key value comparison Failed.", WARNING);
        else
            szMsg = "Set values: %s = %s\n\nReturn values: %s = %s";
            sprintfBox (INFORMATION, TITLE, szMsg, szStrName, szStrValue,
                        szStrName, svStrValue);
        endif;

        if (nvSize != StrLength(szStrValue)) then
            MessageBox ("Size Comparison failed.", WARNING);
        else
            szMsg = "Size in bytes entered: %d\n\nSize in bytes returned: %d";
            sprintfBox (INFORMATION, TITLE, szMsg,
                        nvSize, StrLength(szStrValue) + 1);
        endif;
    endif;
end;
end;

```

## RegDBSetDefaultRoot



**Project** • For InstallScript MSI and Basic MSI projects, it is recommended that you use the Registry view in InstallShield instead of creating registry keys and values through InstallScript code. Handling all of your registry changes in this way allows for a clean uninstallation through the Windows Installer service.

The **RegDBSetDefaultRoot** function sets the root key that is used by the general registry-related functions. Most InstallScript registry functions work on the HKEY\_CLASSES\_ROOT as the default registry hive. Using this function, you can specify another key, such as HKEY\_LOCAL\_MACHINE, HKEY\_CURRENT\_USER, or HKEY\_USERS, as the root key.

You cannot use **RegDBSetDefaultRoot** to change the root key of keys created or handled using the special registry-related functions. For more information, see [Special Registry-Related Functions](#).



**Note** • This function supports the 64-bit parts of the registry by using the REGDB\_OPTION\_WOW64\_64KEY option. For more information, see [REGDB\\_OPTIONS](#).


### Syntax

```
RegDBSetDefaultRoot ( nRootKey );
```



## Parameters

**Table 51** • RegDBSetDefaultRoot Parameters

Parameter	Description
<b>nRootKey</b>	<p>Specify one of the following constants for the root key:</p> <ul style="list-style-type: none"> <li>• HKEY_CLASSES_ROOT</li> <li>• HKEY_CURRENT_USER</li> <li>• HKEY_LOCAL_MACHINE</li> <li>• HKEY_USERS</li> <li>• HKEY_CURRENT_CONFIG</li> <li>• HKEY_DYN_DATA</li> <li>• HKEY_USER_SELECTABLE</li> </ul> <p>If you pass HKEY_USER_SELECTABLE as the argument, a subsequent registry function call uses HKEY_LOCAL_MACHINE as the root key if the <a href="#">ALLUSERS</a> system variable is non-zero when the function is called, or uses HKEY_CURRENT_USER as the root key if ALLUSERS is FALSE when the function is called.</p> <p> <b>Note</b> • Windows does not allow the creation of a key directly under HKEY_LOCAL_MACHINE or HKEY_USERS.</p>

## Return Values

**Table 52** • RegDBSetDefaultRoot Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully set the key.
<b>&lt; 0</b>	Indicates that the function was unable to set the key.

## RegDBSetDefaultRoot Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the RegDBSetDefaultRoot function.
 *
 * RegDBSetDefaultRoot is called to set the default root key for
```

```

* the RegDBCreateKeyEx function.
*
* Note: Windows does not allow the creation of a key
*       directly under HKEY_LOCAL_MACHINE or HKEY_USERS.
*
\*-----*/

#define TITLE "RegDBSetDefaultRoot Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_RegDBSetDefaultRoot(HWND);

function ExFn_RegDBSetDefaultRoot(hMSI)
    STRING szKey, szClass, szMsg, szTitle;
    NUMBER nRootKey;
begin

    // Create a subkey in the HKEY_CLASSES_ROOT key (default).
    szKey = "Test Key";
    szClass = "";

    if (RegDBCreateKeyEx (szKey, szClass) < 0) then
        MessageBox ("RegDBCreateKeyEx failed.", SEVERE);
        abort;
    else
        szMsg = "Successfully created %s in HKEY_CLASSES_ROOT.";
        sprintfBox (INFORMATION, TITLE, szMsg, szKey);
    endif;

    // Set the root key to HKEY_LOCAL_MACHINE.
    nRootKey = HKEY_LOCAL_MACHINE;

    if (RegDBSetDefaultRoot (nRootKey) < 0) then
        MessageBox ("First call to RegDBSetDefaultRoot failed.", SEVERE);
    else
        MessageBox ("Root key successfully set to HKEY_LOCAL_MACHINE.",
                    INFORMATION);
    endif;

    // Create a subkey in the HKEY_LOCAL_MACHINE key.
    if (RegDBCreateKeyEx (szKey, szClass) < 0) then
        MessageBox ("RegDBCreateKeyEx failed.", SEVERE);
        abort;
    else
        szMsg = "Successfully created %s in HKEY_LOCAL_MACHINE";
        sprintfBox (INFORMATION, TITLE, szMsg, szKey);
    endif;

    // Delete the example subkey in HKEY_LOCAL_MACHINE.
    if (RegDBDeleteKey (szKey) < 0) then
        MessageBox ("RegDBDeleteKey failed.", SEVERE);
    else
        szMsg = "Successfully deleted %s in HKEY_LOCAL_MACHINE";
        sprintfBox (INFORMATION, TITLE, szMsg, szKey);
    endif;

    // Set the root key to HKEY_CLASSES_ROOT again.

```

```

nRootKey = HKEY_CLASSES_ROOT;

if (RegDBSetDefaultRoot (nRootKey) < 0) then
    MessageBox ("Second call to RegDBSetDefaultRoot failed.", SEVERE);
else
    MessageBox ("Root key successfully set to HKEY_CLASSES_ROOT.",
        INFORMATION);
endif;

if (RegDBDeleteKey (szKey) < 0) then
    MessageBox ("RegDBDeleteKey failed.", SEVERE);
else
    szMsg = "Successfully deleted %s in HKEY_CLASSES_ROOT";
    sprintfBox (INFORMATION, TITLE, szMsg, szKey);
endif;

end;

```

## RegDBSetItem



**Project** • For InstallScript MSI and Basic MSI projects, it is recommended that you use the Registry view in InstallShield instead of creating registry keys and values through InstallScript code. Handling all of your registry changes in this way allows for a clean uninstallation through the Windows Installer service.

**RegDBSetItem** is a special registry-related function, designed to work with certain predefined registry keys. The **RegDBSetItem** function assigns values under the per application paths key or the application uninstallation key, depending on the value of `nItem`. Calling **RegDBSetItem** with either the `REGDB_APPPATH` or the `REGDB_APPPATH_DEFAULT` option results in the creation of the per application paths key.



**Note** • The InstallScript engine currently does not support writing or reading Add or Remove Programs information for a product in the 64-bit part of the registry. Therefore, using the `REGDB_OPTION_WOW64_64KEY` option with the [REGDB\\_OPTIONS](#) system variable is not supported for this registry function. Enabling the `REGDB_OPTION_WOW64_64KEY` option has no effect on where registry entries are created by this function.

### Syntax

```
RegDBSetItem ( nItem, szValue );
```

## Parameters


**Table 53 •** RegDBSetItem Parameters

Parameter	Description
<b>nItem</b>	<p>Specifies the item to set. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>REGDB_APPPATH</b>—The data of the Path value under the per-application paths key. This key is created by the installation as a result of calling <b>CreateInstallationInfo</b>. You must call <b>CreateInstallationInfo</b> to create this key before calling <b>RegDBSetItem</b>. (In an event-based script, <b>CreateInstallationInfo</b> is called in the default <b>OnMoveData</b> event handler code.)</li> <li>• <b>REGDB_APPPATH_DEFAULT</b>—The data of the DefaultPath value under the per-application paths key. This key is created by the installation as a result of calling <b>CreateInstallationInfo</b>. You must call <b>CreateInstallationInfo</b> to create this key before calling <b>RegDBSetItem</b>. (In an event-based script, <b>CreateInstallationInfo</b> is called in the default <b>OnMoveData</b> event handler code.)</li> <li>• <b>REGDB_UNINSTALL_COMMENTS</b>—The data of the Comments value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_CONTACT</b>—The data of the Contact value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_DISPLAY_VERSION</b>—The data of the DisplayVersion value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_DISPLAYICON</b>—The data of the DisplayIcon value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_HELPLINK</b>—The data of the HelpLink value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_HELPTELEPHONE</b>—The data of the HelpTelephone value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_INSTALLDATE</b>—The data of the InstallDate value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_INSTALLLOC</b>—The data of the InstallLocation value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_INSTALLSOURCE</b>—The data of the InstallSource value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_LANGUAGE</b>—The data of the Language value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_LOGFILE</b>—The data of the LogFile value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_MAINT_OPTION</b>—The data of the LogMode value under the uninstallation key.</li> </ul>

**Table 53** • RegDBSetItem Parameters (cont.)

Parameter	Description
<b>nItem (cont.)</b>	<ul style="list-style-type: none"> <li>• <b>REGDB_UNINSTALL_MAJOR_VERSION</b>—The data of the VersionMajor value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_MAJOR_VERSION_OLD</b>—The data of the MajorVersion value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_MINOR_VERSION</b>—The data of the VersionMinor value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_MINOR_VERSION_OLD</b>—The data of the MinorVersion value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_MODIFYPATH</b>—The data of the ModifyPath value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_NAME</b>—The data of the DisplayName value under the uninstallation key. When this constant is used, szValue specifies the application name shown in the list of uninstallable applications in the Control Panel.  In an InstallScript or InstallScript MSI project, an easier way to set this value is to set the system variable <b>UNINSTALL_DISPLAYNAME</b> directly.</li> <li>• <b>REGDB_UNINSTALL_NOMODIFY</b>—The data of the NoModify value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_NOREMOVE</b>—The data of the NoRemove value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_NOREPAIR</b>—The data of the NoRepair value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_PRODUCTGUID</b>—The data of the ProductGuid value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_PRODUCTID</b>—The data of the ProductId value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_PUBLISHER</b>—The data of the Publisher value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_README</b>—The data of the Readme value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_REGCOMPANY</b>—The data of the RegCompany value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_REGOWNER</b>—The data of the RegOwner value under the uninstallation key.</li> </ul>

**Table 53** • RegDBSetItem Parameters (cont.)

Parameter	Description
<b>nItem</b> <b>(cont.)</b>	<ul style="list-style-type: none"> <li>• <b>REGDB_UNINSTALL_STRING</b>—The data of the UninstallString value under the uninstallation key. The UninstallString value is created when the <a href="#">MaintenanceStart</a> or <a href="#">DeinstallStart</a> functions are called.</li> <li>• <b>REGDB_UNINSTALL_SYSTEMCOMPONENT</b>—The data of the SystemComponent value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_URLINFOABOUT</b>—The data of the URLInfoAbout value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_URLUPDATEINFO</b>—The data of the URLUpdateInfo value under the uninstallation key.</li> <li>• <b>REGDB_UNINSTALL_VERSION</b>—The data of the Version value under the uninstallation key. When this constant is used, szValue specifies the installed product's version number expressed as a string, for example, "16777216".</li> </ul>
 <p><b>Important</b> • Do not use REGDB_WINCURRVER_REGOWNER or REGDB_WINCURRVER_REGORGANIZATION with this function. Windows sets these values, and they should not be set by an installation.</p>	
<b>szValue</b>	Specifies the value to assign to the specified item.

## Return Values

**Table 54** • RegDBSetItem Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully set the value.
<b>&lt; 0</b>	Indicates that the function failed.

## Additional Information

By default, any text that is surrounded by angle brackets—for example, "<my registry entry text>"—in this function's string arguments is interpreted as a text substitution and is processed accordingly. To disable text substitution processing for the string arguments of registry functions, call **Disable** with the REGISTRYFUNCTIONS\_USETEXTSUBS argument.

## RegDBSetItem Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the RegDBSetItem and RegDBGetItem functions.
*
* This script sets several registry keys; then it gets those
* keys and displays their current values.
*
\*-----*/

#define COMPANY_NAME      "ExampleCompany"
#define PRODUCT_NAME      "ExampleProduct"
#define VERSION_NUMBER    "5.00.00"
#define PRODUCT_KEY       "EXAMPLE.EXE"
#define DEINST_KEY        "ExampleDeinstKey"
#define APP_DEF_LOG_PATH  "C:\\EXAMPLE\\TEMP"
#define APP_PATH           "C:\\EXAMPLE"
#define APP_DEF_PATH       "C:\\EXAMPLE\\TARGET"
#define UNINSTALL_NAME    "ExampleUninstallName"
#define TITLE              "RegDBSetItem Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_RegDBSetItem(HWND);

function ExFn_RegDBSetItem(hMSI)
    STRING svLogFile;
    STRING svValue, szTitle;
begin

    // Set the root key.
    RegDBSetDefaultRoot (HKEY_LOCAL_MACHINE);

    // Set installation and uninstallation information in
    // order to call RegDBSetItem and RegDBGetItem.
    InstallationInfo (COMPANY_NAME, PRODUCT_NAME, VERSION_NUMBER, PRODUCT_KEY);
    DeinstallStart (APP_DEF_LOG_PATH, svLogFile, DEINST_KEY, 0);

    // Set the value of the application path key in the
    // registry to the value of szAppPath.
    if (RegDBSetItem (REGDB_APPPATH, APP_PATH) < 0) then
        MessageBox ("Unable to set application path key.", SEVERE);
    else
        sprintfBox (INFORMATION, TITLE, "RegDBSetItem set the application " +
                    "path key to %s.", APP_PATH);
    endif;

    // Set the value of the application default path key in
    // the registry to the value of szAppDefPath.
    if (RegDBSetItem (REGDB_APPPATH_DEFAULT, APP_DEF_PATH) < 0) then
        MessageBox ("Unable to set application default path key.", SEVERE);
    else
        sprintfBox (INFORMATION, TITLE, "RegDBSetItem set the application " +
                    "default path key to %s.", APP_DEF_PATH);
    endif;

    // Set the value of the uninstall name key in the

```

```

// registry to the value of szUninstallName.
if (RegDBSetItem (REGDB_UNINSTALL_NAME, UNINSTALL_NAME) < 0) then
    MessageBox ("Unable to set uninstall name key.", SEVERE);
else
    sprintfBox (INFORMATION, TITLE, "RegDBSetItem set the uninstall " +
        "name key to %s.", UNINSTALL_NAME);
endif;

// Set up title parameter for call to sprintfBox.
szTitle = "RegDBGetItem";

// Get the value of the application path key from the registry.
if (RegDBGetItem (REGDB_APPPATH, svValue) < 0) then
    MessageBox ("Unable to get value of application path key.", SEVERE);
else
    sprintfBox (INFORMATION, TITLE, "RegDBGetItem retrieved the value " +
        "of the application path key: %s.", svValue);
endif;

// Get the value of the application default path key in
// the registry.
if (RegDBGetItem (REGDB_APPPATH_DEFAULT, svValue) < 0) then
    MessageBox ("Unable to get application default path key", SEVERE);
else
    sprintfBox (INFORMATION, TITLE, "RegDBGetItem retrieved the value " +
        "of the application default path key: %s.", svValue);
endif;

// Get the value of the uninstall name key from the registry.
if (RegDBGetItem (REGDB_UNINSTALL_NAME, svValue) < 0) then
    MessageBox ("Unable to get application uninstall name key.", SEVERE);
else
    sprintfBox (INFORMATION, TITLE, "RegDBGetItem retrieved the value " +
        "of the uninstallation name key: %s.", svValue);
endif;

end;

```

## RegDBSetKeyValueEx



**Project** • For InstallScript MSI and Basic MSI projects, it is recommended that you use the Registry view in InstallShield instead of creating registry keys and values through InstallScript code. Handling all of your registry changes in this way allows for a clean uninstallation through the Windows Installer service.

The **RegDBSetKeyValueEx** function sets the value of a particular value name under a specified key in the registry.

If the value name does not already exist, **RegDBSetKeyValueEx** creates it. If the value data already exists, **RegDBSetKeyValueEx** overwrites it.

If the key does not already exist, **RegDBSetKeyValueEx** calls [RegDBCreateKeyEx](#), passing the key from the szKey parameter to create it.

**RegDBSetKeyValueEx** is a general registry-related function, designed to work with all registry keys, including those handled by the special registry-related functions.





**Note** • This function supports the 64-bit parts of the registry by using the `REGDB_OPTION_WOW64_64KEY` option. For more information, see [REGDB\\_OPTIONS](#).

## Syntax

```
RegDBSetKeyValueEx ( szKey, szName, nType, szValue, nSize );
```

## Parameters

**Table 55 • RegDBSetKeyValueEx Parameters**

Parameter	Description
<b>szKey</b>	<p>Specifies the name of the key. Separate different levels in the subkey with a double backslash (\\).</p> <p>If the root key is HKEY_CLASSES_ROOT, you do not need to specify the HKEY_CLASSES_ROOT in this parameter. Unless you specify otherwise, the InstallScript engine assumes that the value that is being set is under the key HKEY_CLASSES_ROOT. To specify a different root key, you can call <a href="#">RegDBSetDefaultRoot</a> before calling <b>RegDBSetKeyValueEx</b>.</p>
<b>szName</b>	Specifies the value name for the value data to be set. To set the default value of the key specified in szKey, pass a null string ("" ) in this parameter.
<b>nType</b>	<p>Specifies the type of data to be set. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>REGDB_STRING</b>—A string variable; no newline characters allowed.</li> <li>• <b>REGDB_STRING_EXPAND</b>—A string variable holding an expandable environment variable expression, such as "%MYPATH%".</li> <li>• <b>REGDB_STRING_MULTI</b>—A string variable; newline characters allowed.</li> <li>• <b>REGDB_NUMBER</b>—A number expressed as a string and passed in string variable. Creates data of type DWORD. When nType is REGDB_NUMBER, passing a decimal string in szValue results in a numeric value being stored in the registry. This numeric value is then displayed as a hexadecimal number followed by its decimal equivalent in parentheses. You cannot pass a hexadecimal number in the string in szValue—it <i>must</i> be a decimal number.</li> <li>• <b>REGDB_BINARY</b>—Binary data stored in a string.</li> </ul>
<b>szValue</b>	Specifies the value to associate with the value name. All values must be passed as string variables. Numbers must be expressed as strings. The InstallScript engine converts them to numbers internally.
<b>nSize</b>	Specifies the size—in bytes—of the data to be set. You can specify -1 in this parameter when nType is REGDB_STRING, REGDB_STRING_EXPAND, or REGDB_NUMBER, and the InstallScript engine will set the size. However, with REGDB_BINARY and REGDB_STRING_MULTI, you must always specify the number of bytes of binary data that you are storing.

## Return Values

**Table 56 •** RegDBSetKeyValueEx Return Values

Return Value	Description
0	Indicates that the function successfully set the key.
< 0	Indicates that the function was unable to set the key.

## Additional Information

By default, any text that is surrounded by angle brackets—for example, "<my registry entry text>"—in this function's string arguments is interpreted as a text substitution and is processed accordingly. To disable text substitution processing for the string arguments of registry functions, call **Disable** with the REGISTRYFUNCTIONS\_USETEXTSUBS argument.

## RegDBSetKeyValueEx Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the RegDBSetKeyValueEx and RegDBGetKeyValueEx
 * functions.
 *
 * RegDBCreateKeyEx is called to create a test subkey in the
 * HKEY_CLASSES_ROOT key. The value of a key is set in integer
 * form using the REGDB_NUMBER option of the RegDBSetKeyValueEx
 * function. After this value is set, it is retrieved using
 * the RegDBGetKeyValueEx function and verified.
 *
 \*-----*/

#define TITLE "RegDBSetKeyValueEx & RegDBGetKeyValueEx"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_RegDBSetKeyValueEx(HWND);

function ExFn_RegDBSetKeyValueEx(hMSI)
    STRING szKey, szKeyName, szNumValue, svNumValue, szTitle, szMsg;
    NUMBER nType, nSize, nvType, nvSize;
begin

    // Create a key to test.
    szKey = "TestKey";
    if (RegDBCreateKeyEx (szKey, "") < 0) then
        MessageBox ("RegDBCreateKeyEx failed.", SEVERE);
```

```

        abort;
    endif;

    // Set up parameters for call to RegDBSetKeyValueEx.
    szNumName = "TestValue";
    szNumValue = "12345";
    nType      = REGDB_NUMBER;
    nSize      = -1;

    // Set a key name and a value associated with it.
    if (RegDBSetKeyValueEx (szKey, szNumName, nType, szNumValue,
                           nSize) < 0) then
        MessageBox ("RegDBSetKeyValueEx failed.", SEVERE);
        abort;
    else
        // Display what RegDBSetKeyValueEx has done.
        szMsg = "%s set to: %s";
        sprintfBox (INFORMATION, TITLE, szMsg, szNumName, szNumValue);
    endif;

    // Retrieve key value information.
    if (RegDBGetKeyValueEx (szKey, szNumName, nvType, svNumValue,
                           nvSize) < 0) then
        MessageBox ("RegDBGetKeyValueEx failed.", SEVERE);
    else
        // Check to see if the value returned is the same as the value set.
        if (nvType != REGDB_NUMBER) then
            MessageBox ("Type comparison failed.", SEVERE);
        endif;

        if (svNumValue != szNumValue) then
            MessageBox ("Subkey value comparison failed.", SEVERE);
        endif;

        // Display what RegDBGetKeyValueEx retrieved.
        szMsg = "%s has value: %s\n\nThis data is %d bytes.";
        sprintfBox (INFORMATION, TITLE, szMsg, szNumName, svNumValue, nvSize);
    endif;

    // Delete the created test key.
    if (RegDBDeleteKey (szKey) < 0) then
        MessageBox ("RegDBDeleteKey failed.", SEVERE);
    endif;

end;

```

## RegDBSetVersion



**Project** • For InstallScript MSI and Basic MSI projects, it is recommended that you use the Registry view in InstallShield instead of creating registry keys and values through InstallScript code. Handling all of your registry changes in this way allows for a clean uninstallation through the Windows Installer service.

The **RegDBSetVersion** function places the value of the system variable [IFX\\_PRODUCT\\_VERSION](#) as the data for the Version value under the application uninstallation registry key, creating the registry value if it does not already exist. If IFX\_PRODUCT\_VERSION is not in packed DWORD format, the function fails.



**Note** • This function supports the 64-bit parts of the registry by using the `REGDB_OPTION_WOW64_64KEY` option. For more information, see [REGDB\\_OPTIONS](#).

## Syntax

```
RegDBSetVersion ( );
```

## Parameters

None.

## Return Values

**Table 57** • RegDBSetVersion Return Values

Return Value	Description
<code>&gt;= ISERR_SUCCESS</code>	The function successfully placed the data in the registry value.
<code>&lt; ISERR_SUCCESS</code>	The function failed to place data in the registry.

## Additional Information

RegDBSetVersion is called by the default code for the OnMoveData event handler function.

# RegisterFontResource

The **RegisterFontResource** function registers or unregisters the font resource specified by szFileName. This function is called by the default code for the [OnInstalledFontFile](#) and [OnUninstallingFontFile](#) event handler functions.

## Syntax

```
RegisterFontResource ( szFileName, svFontTitle, bRegister, nOptions );
```

Parameters

Table 58 • RegisterFontResource Parameters

Parameter	Description
<b>szFileName</b>	<p>Specifies the fully qualified file name of the font file (.fnt, .fon, .fot, .mmm, .otf, .pfb, .pfm, .ttc, or .ttf file) to be registered or unregistered. Typically you would install a font file to FOLDER_FONTS. Only a font file that exists on the target system can be registered, and you must unregister a font file before you remove it from the system.</p>
<b>svFontTitle</b>	<p>This parameter must be a variable name and not a literal value. When RegisterFontResource is called, this parameter specifies the title for the font, if its value is not a null string (""); when RegisterFontResource returns, this parameter contains the title that was used when registering or unregistering the font file.</p> <p>If the font file being registered or unregistered is a TrueType font file, you can specify in this parameter a string variable whose value is a null string. In this case RegisterFontResource attempts to get the title by internally calling <a href="#">GetTrueTypeFontFileInfo</a>; if this fails, the file name of the font file is used as the title.</p> <p>If the REGFONT_OPTION_DONTUPDATeregistry option is specified in this function's fourth argument, svFontTitle is ignored.</p>
<b>bRegister</b>	<p>Specifies whether to register the font resource (TRUE) or unregister the font resource (FALSE).</p>

**Table 58** • RegisterFontResource Parameters (cont.)

Parameter	Description
<b>nOptions</b>	<p>Specifies various options. Pass one of the following constants in this parameter, or combine these constants using the OR operator ( ):</p> <ul style="list-style-type: none"> <li>• <b>REGFONT_OPTION_DEFAULT</b>—Specifies that the function has its default behavior.</li> <li>• <b>REGFONT_OPTION_DONTBROADCASTFONTCHANGE MSG</b>—Specifies that the function does not send a message to top level windows indicating that font information has changed after registering or registering the font. You should specify this flag only if your installation sends this message manually after calling this function by calling the Windows API function PostMessage as in the following example:</li></ul> <pre>PostMessage( HWND_BROADCAST, WM_FONTCHANGE, 0, 0 );</pre> <p>You would typically use this flag if you are calling RegisterFontResource multiple times to update multiple fonts and then calling PostMessage.</p> <ul style="list-style-type: none"> <li>• <b>REGFONT_OPTION_DONTUPDATeregistry</b>—Specifies that the function does not add the font registration information to the registry or remove the font information from the registry. If you specify this option for font registration, the font is available only until the next time the system is rebooted. Note that if this option is specified svFontTitle is ignored.</li></ul>

## Return Values

**Table 59** • RegisterFontResource Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the function successfully registered or unregistered the font.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was unable to register or unregister the font.

## Additional Information

- Font information is written to HKEY\_LOCAL\_MACHINE in all cases, so typically in order to install a font the end user must have administrator privileges on the system.
- The function calls PostMessage instead of SendMessage to send the WM\_FONTCHANGE message. This is because SendMessage (when used with HWND\_BROADCAST) waits for all open windows to respond to the

message before returning, which could cause the installation to freeze if an open window on the system fails to respond to the message. Since the function calls `PostMessage`, the function could return before Windows has processed the message and released the font from the font cache; so when using this function to uninstall fonts, it is recommended that you wait a few seconds (by calling [Delay](#)) before removing the font file.

- If you call this function in a feature event or a file install event (such as `OnInstallingFile` or `OnInstalledFile`) the font registration is associated with the corresponding feature and the font is automatically unregistered before the file or feature is uninstalled.

## RegisterFontResource Example

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the RegisterFontResource function.
*
\*-----*/

function OnBegin()
    string szFileName, svFontTitle;
begin
    szFileName = FOLDER_FONTS ^ "Estre.ttf";
    svFontTitle = "Estrangelo Edessa";
    RegisterFontResource ( szFileName, svFontTitle, TRUE, REGFONT_OPTION_DEFAULT );
    // If you cut and paste this sample script into a project
    // and run it, the following line aborts execution of the script.
    abort;
end;
```

## ReleaseDialog

The **ReleaseDialog** function frees all memory associated with the custom dialog identified in `szDialogName`. Call this function after calling [EndDialog](#). Call this function outside the message-handling case statement.

### Syntax

```
ReleaseDialog ( szDialogName );
```



## Parameters

**Table 60** • ReleaseDialog Parameters

Parameter	Description
<b>szDialogName</b>	Specifies the name of the dialog to destroy.

## Return Values

**Table 61** • ReleaseDialog Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully freed all memory associated with the custom dialog.
<b>DLG_ERR (-1)</b>	Function failed. The dialog name may be invalid.
<b>DLG_ERR_ENDDLG (-2)</b>	ReleaseDialog was called before EndDialog. You must call EndDialog first to remove the dialog.

## ReleaseDialog Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the DefineDialog, EndDialog, and ReleaseDialog
* functions.
*
* This script opens a simple custom dialog that displays
* a bitmap. The dialog can be closed with any of three
* buttons: Back, Next, or Cancel.
*
* The "custom" dialog used in this script is actually the
* InstallShield Sd dialog that is displayed by the built-in
* function SdBitmap. Because this dialog is stored in
* the file _isres.dll, which is already compressed in
* the installation, it can be used in a script as a custom
* dialog.
*
* In order to use this dialog as a custom dialog, the
* script first defines it by calling DefineDialog. It then
* displays the dialog by calling WaitOnDialog. When an event
* ends dialog processing, EndDialog is called to close the
* dialog. Then the dialog is released from memory by
* a call to ReleaseDialog.
```

```

*
\*-----*/

// Dialog and control IDs.
#define RES_DIALOG_ID      12027 // ID of dialog itself
#define RES_PBUT_NEXT      1 // ID of Next button
#define RES_PBUT_CANCEL    9 // ID of Cancel button
#define RES_PBUT_BACK      12 // ID of Back button

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ReleaseDialog(HWND);

function ExFn_ReleaseDialog(hMSI)
    STRING  szDialogName, szDLLName, szDialog;
    NUMBER  nDialog, nResult, nCmdValue;
    BOOL     bDone;
    HWND     hInstance, hwndParent, hwndDlg;
begin

    // Define the name of a dialog to pass as first
    // parameter to DefineDialog.
    szDialogName = "ExampleDialog";

    // DefineDialog's second parameter will be 0 because the
    // .dll file is in _isres.dll.
    hInstance = 0;

    // DefineDialog's third parameter will be null; installation will
    // search for the dialog in _isuser.dll and _isres.dll.
    szDLLName = "";

    // DefineDialog's fifth parameter will be null because the
    // dialog is identified by its ID in the fourth parameter.
    szDialog = "";

    // This value is reserved and must be 0.
    hwndParent = 0;

    // Define the dialog. The installation's main window will own the
    // dialog (indicated by HWND_INSTALL in parameter 7).
    nResult = DefineDialog (szDialogName, hInstance, szDLLName,
                           RES_DIALOG_ID, szDialog, hwndParent,
                           HWND_INSTALL, DLG_MSG_STANDARD|DLG_CENTERED);

    // Check for an error.
    if (nResult < 0) then
        MessageBox ("An error occurred while defining the dialog.", SEVERE);
        bDone = TRUE;
        abort;
    endif;

    // Initialize the indicator used to control the while loop.
    bDone = FALSE;

    // Loop until done.
    repeat

```

```

// Display the dialog and return the next dialog event.
nCmdValue = WaitOnDialog(szDialogName);

// Respond to the event.
switch (nCmdValue)
case DLG_CLOSE:
    // The user clicked the window's Close button.
    Do (EXIT);
case DLG_ERR:
    MessageBox ("Unable to display dialog. Setup canceled.", SEVERE);
    abort;
case DLG_INIT:
    // Initialize the back, next, and cancel button enable/disable states
    // for this dialog and replace %P, %VS, %VI with
    // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
    // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
    hwndDlg = CmdGetHwndDlg(szDialogName);
    SdGeneralInit(szDialogName, hwndDlg, 0, "");
case RES_PBUT_CANCEL:
    // The user clicked the Cancel button.
    Do (EXIT);
case RES_PBUT_NEXT:
    bDone = TRUE;
case RES_PBUT_BACK:
    bDone = TRUE;
endswitch;

until bDone;

// Close the dialog.
EndDialog (szDialogName);

// Free the dialog from memory.
ReleaseDialog (szDialogName);

end;

```

## RenameFile

The **RenameFile** function changes the name of a file or directory and/or moves a file or directory (and its subdirectories and files) from one parent directory to another.



**Note** • Note the following:

- If *szSource* and *szTarget* reference the same directory or—when using unqualified names, if *SRCDIR* and *TARGETDIR* reference the same directory for an InstallScript installation (or if *SRCDIR* and *INSTALLDIR* reference the same directory for a Basic MSI or InstallScript MSI installation)—the file or directory that is specified by *szSource* is renamed, unless a file or directory with the name that is specified by *szTarget* already exists in that directory.
- If *szSource* and *szTarget* reference different directories or—when using unqualified names, if *SRCDIR* and *TARGETDIR* reference different directories for an InstallScript installation (or if *SRCDIR* and *INSTALLDIR* reference different directory for a Basic MSI or InstallScript MSI installation)—the file or directory that is specified by


*szSource is moved to the new directory and given the name that is specified by szTarget, unless a file or directory with that name already exists in that directory.*

## Syntax

```
RenameFile ( szSource, szTarget );
```

## Parameters

**Table 62 • RenameFile Parameters**

Parameter	Description
<b>szSource</b>	Specifies the name of the file or directory to be renamed or moved. If szFileOld specifies a fully qualified file name or directory (it includes a path), <b>RenameFile</b> renames or moves the file or directory in the specified directory. If szSource contains an unqualified file name or directory name (without path information), <b>RenameFile</b> renames or moves the file or directory in the directory specified by the system variable <a href="#">SRCDIR</a> .   <b>Note •</b> Wildcard characters are not permitted. Only one file or one directory (including its subdirectories and folders) can be renamed or moved with each call to <b>RenameFile</b> .
<b>szTarget</b>	Specifies a new name and/or location for the file or directory. If szFileNew specifies a fully qualified file name or directory (it includes a path), <b>RenameFile</b> renames the file or moves it to the specified directory. If szFileNew contains an unqualified file name or directory name (without path information), <b>RenameFile</b> either renames the file or directory, or moves the file or directory to the directory specified by the system variable TARGETDIR (for an InstallScript installation) or the system variable INSTALLDIR (for a Basic MSI or InstallScript MSI installation).

## Return Values

**Table 63 • RenameFile Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully changed the name of the file or directory or moved the file or directory.
<b>&lt; 0</b>	Indicates that the function was unable to change the name of the file or folder or move the file or folder.

## RenameFile Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the RenameFile function.
```

```

*
* First, RenameFile is called to rename FILENAME1 to FILENAME2.
* It is then called again to move the FILENAME2 file to the
* TARGET directory.
*
* This can also be done in one call to RenameFile. The third
* call to RenameFile demonstrates this by renaming and moving
* the FILENAME2 file from the TARGET directory to the SOURCE
* directory, with the FILENAME1 file name.
*
* Note: Before running this script, set the preprocessor
*       constants so that they specify valid file names and
*       paths on the target system.
*
\*-----*/

#define FILENAME1 "ISExamp1.txt"
#define FILENAME2 "ISExamp1.bak"
#define SOURCE_DIR "C:\\ISExamp1\\Source"
#define TARGET_DIR "C:\\ISExamp1\\Target"
#define TITLE      "RenameFile Example"

#include "ifx.h"

function OnBegin()
begin

    // Set up system variables for rename operation.
    SRCDIR    = SOURCE_DIR;
    TARGETDIR = SOURCE_DIR;

    // Rename FILENAME1 to FILENAME2.
    if (RenameFile (FILENAME1, FILENAME2) < 0) then
        MessageBox("First call to RenameFile failed.", SEVERE);
        abort;
    else
        szMsg = "%s successfully renamed to %s.";
        sprintfBox(INFORMATION, szTitle, szMsg, FILENAME1, FILENAME2);
    endif;

    // Set up system variables to move a file from one directory to another.
    SRCDIR    = SOURCE_DIR;
    TARGETDIR = TARGET_DIR;

    // Move the file from the SOURCE to the TARGET directory.
    if (RenameFile(FILENAME2, FILENAME2) < 0) then
        MessageBox("Second call to RenameFile failed.", SEVERE);
        abort;
    else
        szMsg = "%s successfully moved to %s.";
        sprintfBox(INFORMATION, TITLE, szMsg, FILENAME2, TARGETDIR);
    endif;

    // Set up system variables to move the file back to its original location.
    SRCDIR    = TARGET_DIR;
    TARGETDIR = SOURCE_DIR;

    // Rename the file and move it from the TARGET directory

```

```

// to the SOURCE directory.
if (RenameFile(FILENAME2, FILENAME1) < 0) then
    MessageBox("Third call to RenameFile failed.", SEVERE);
    abort;
else
    szMsg = "%s successfully renamed %s and moved to the directory %s.";
    sprintfBox(INFORMATION, TITLE, szMsg, FILENAME2, FILENAME1, TARGETDIR);
endif;

end;

```

## ReplaceFolderIcon

The [ReplaceShortcut](#) function supersedes the **ReplaceFolderIcon** function.

The **ReplaceFolderIcon** function replaces a shortcut in a specified folder. You must specify an existing folder, either one you have created with the [CreateProgramFolder](#) function or one that already exists on the user's system.

### Syntax

```

ReplaceFolderIcon ( szProgramFolder, szItemName, szNewItem, szCmdLine, szWorkingDir, szIconPath, nIcon,
    szShortCutKey, nFlag );

```

## Parameters

**Table 64 •** ReplaceFolderIcon Parameters

Parameter	Description
<b>szProgramFolder</b>	Specify the name of the folder that contains the shortcut to replace.
<b>szItemName</b>	Specify the name of the shortcut to replace.
<b>szNewItem</b>	Specify the name of the shortcut as it should appear after the replacement.
<b>szCmdLine</b>	Specify one of the following: <ul style="list-style-type: none"> <li>The fully qualified name of the executable associated with the icon, including any command-line parameters.</li> <li>The fully qualified path if szItemName is a subfolder.</li> </ul>
<b>szWorkingDir</b>	Specify the directory where the application's program files are located. (Not applicable if szItemName is a subfolder.) To make the directory that contains the program file the working directory, pass a null string ("") in this parameter.
<b>szIconPath</b>	Specify the name of an icon file or a valid Windows executable that contains the new icon.
<b>nlcon</b>	If you specified an executable file for szIconPath, specify the icon index in the executable file. Otherwise, enter the number 0 for nlcon.
<b>szShortCutKey</b>	Specify the string that contains the shortcut key sequence the user can press to start the program. For example, if you wanted the user to be able to open the application by depressing the "Ctrl," the "Alt," and then the "1" key, specify "Ctrl + Alt + 1" in this parameter.
<b>nFlag</b>	Specify one or more options. Pass the following predefined constants in this parameter. To specify more than one option, combine constants with the OR ( ) operator. <ul style="list-style-type: none"> <li><b>NULL</b>—Indicates no options.</li> <li><b>REPLACE</b>—Indicates that the existing icon should be replaced with the new icon.</li> <li><b>RUN_MAXIMIZED</b>—Indicates that the program should be maximized when launched.</li> <li><b>RUN_MINIMIZED</b>—Indicates that the program should be minimized when launched.</li> </ul>



## Return Values

**Table 65** • ReplaceFolderIcon Return Values

Return Value	Description
0	Indicates that the function successfully replaced the shortcut.
< 0	Indicates that the function was unable to replace the icon.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

## ReplaceFolderIcon Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ReplaceFolderIcon function.
*
* Note: In order for this script to run correctly, you must set
*       preprocessor constants to valid file names and a path
*       on the target system. To easily create this example
*       folder and icon, run the AddFolderIcon example #3.
*
\*-----*/

#define FOLDER      "C:\\Windows\\"
#define NEW_PROGRAM "C:\\WINDOWS\\WRITE.EXE"
#define NEW_PARAM   "C:\\WINDOWS\\README.TXT"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ReplaceFolderIcon(HWND);

function ExFn_ReplaceFolderIcon(hMSI)
    STRING szProgramFolder, szItemName, szNewItem, szCmdLine, szWorkingDir;
    STRING szShortCutKey, szIconPath, szProgram, szParam;
    NUMBER nIcon, nFlag;
begin

    szProgramFolder = FOLDER ^ "Example folder";
    szItemName      = "Notepad Example";
    szNewItem       = "New Wordpad Example";

    // Make sure the space is not seen as a delimiter.
    szProgram = NEW_PROGRAM;
    LongPathToQuote (szProgram, TRUE);
```

```

szParam = NEW_PARAM;
LongPathToShortPath(szParam);

szCmdLine      = szProgram + " " + szParam;
szWorkingDir   = "";
szIconPath     = "";
nIcon          = 0;
szShortCutKey  = "";
nFlag          = REPLACE|RUN_MAXIMIZED;

// Display the folder on the screen.
ShowProgramFolder (szProgramFolder, SW_SHOW);

// Replace the "Notepad Example" icon with "New Wordpad Example".
if (ReplaceFolderIcon (szProgramFolder, szItemName, szNewItem, szCmdLine,
                      szWorkingDir, szIconPath, nIcon, szShortCutKey,
                      nFlag) < 0) then
    MessageBox ("ReplaceFolderIcon failed.", SEVERE);
else
    MessageBox ("Icon successfully replaced.", INFORMATION);
endif;

end;

```

## ReplaceProfString

In Windows Installer based projects, all .ini file changes should be created in the INI Files view of the IDE. Handling all of your .ini file changes in this way allows for a clean uninstallation through the Windows Installer service.

The **ReplaceProfString** function replaces a profile string in an .ini file. This function can replace values of duplicate keys (non-unique keys) such as those found in the [386Enh] section of the System.ini file (device = ...). The function searches for a szKeyName = szOrigValue, and replaces the line. If it is not found, it adds the szKeyName = szReplaceValue line to the beginning of the szSectionName section.

If you are adding unique keys (that is, keys that are all different for a given section), use the [WriteProfString](#) function.

Use this function to replace only non-unique key names, such as the device= line in the System.ini file.

### Syntax

```
ReplaceProfString ( szFileName, szSectionName, szKeyName, szOrigValue, szReplaceValue );
```

## Parameters

**Table 66** • ReplaceProfString Parameters

Parameter	Description
<b>szFileName</b>	Specifies the name of the .ini file in which the profile string is to be replaced. If the file name is unqualified (that is, if a drive designation and path are <i>not</i> included), InstallShield searches for the file in the Windows folder. If the file does not exist, it is created in the specified folder; if a path is not included in file name, the file is created in the Windows folder. Note that if the file name is qualified with a path that does not exist, ReplaceProfString will fail.
<b>szSectionName</b>	Specifies the name of the .ini file section to search for szKeyName. The section name should <i>not</i> be enclosed within delimiting brackets ( [ ] ). The search for this name is <i>not</i> case sensitive.
<b>szKeyName</b>	Specifies the name of the key to replace. If the key does not exist, it is created.
<b>szOrigValue</b>	Specifies the current value of the key specified by szKeyName.
<b>szReplaceValue</b>	Specifies the new value to assign to szKeyName. The value of this parameter will appear to the right of the equal sign in the profile string (szKeyName = szValue).

## Return Values

**Table 67** • ReplaceProfString Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully replaced or added the profile string.
<b>&lt; 0</b>	Indicates that the function was unable to replace or add the profile string.

## Additional Information

Changes made to .ini files can be logged for uninstallation. However, there are some important restrictions to be aware of. For more information, see Uninstalling Initialization (.ini) File Entries.

## ReplaceProfString Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ReplaceProfString function.
*
* ReplaceProfString is called for the first time to replace
* the szKeyName key value szOrigValue with the value
* szReplaceValue. Then ReplaceProfString is called again to
* replace the newly set value of szKeyName, szReplaceValue,
* with szOrigValue.
*
* NOTE: In order for this script to run properly, you must set
*       the constant EXAMPLE_INI to reference an existing
*       initialization file on the target system. That file
*       should include the following lines:
*
*       [Old Section]
*       Old Key=Old value
*
*-----*/

#define EXAMPLE_INI "C:\\ISExempl.ini"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_ReplaceProfString(HWND);

function ExFn_ReplaceProfString(hMSI)
    STRING szSectionName, szKeyName, szOrigValue, szReplaceValue;
begin

    szSectionName = "Old Section";
    szKeyName     = "Old Key";
    szOrigValue   = "Old value";
    szReplaceValue = "New value";

    // Replace szOrigValue with szReplaceValue.
    if (ReplaceProfString(EXAMPLE_INI, szSectionName, szKeyName, szOrigValue,
        szReplaceValue) < 0) then
        MessageBox("ReplaceProfString failed.", SEVERE);
        abort;
    else
        SprintfBox (INFORMATION, "Replacement Successful",
            "Original: %s\nNew: %s", szOrigValue, szReplaceValue);
    endif;

    // Replace szReplaceValue with szOrigValue.
    if (ReplaceProfString(EXAMPLE_INI, szSectionName, szKeyName, szReplaceValue,
        szOrigValue) < 0) then;
```

```

        MessageBox("ReplaceProfString failed.", SEVERE);
    else
        sprintfBox (INFORMATION, "Replacement Successful",
                    "Original: %s\nNew:  %s", szReplaceValue, szOrigValue);
    endif;

end;

```

## ReplaceShortcut

The **ReplaceShortcut** function replaces a shortcut in a specified folder. You must specify an existing folder, either one that you have created with the [CreateShortcutFolder](#) function or one that already exists on the end user's system.

### Syntax

```

ReplaceShortcut (szShortcutFolder, szName, szNewItem, szCmdLine, szWorkingDir, szIconPath, nIcon,
                szShortCutKey, nFlag);

```

## Parameters

**Table 68 • ReplaceShortcut Parameters**

Parameter	Description
<b>szShortcutFolder</b>	Specify the name of the folder that contains the shortcut that you want to replace.
<b>szName</b>	Specify the name of the shortcut that you want to replace.
<b>szNewItem</b>	Specify the name of the shortcut as it should appear after the replacement.
<b>szCmdLine</b>	Specify one of the following: <ul style="list-style-type: none"> <li>• The fully qualified name of the executable file that is associated with the icon, including any command-line parameters.</li> <li>• The fully qualified path if szName is a subfolder.</li> </ul>
<b>szWorkingDir</b>	<p>If szName is not a subfolder, specify the directory that contains the product's program files.</p> <p>To make the directory that contains the product's program files the working directory, pass a null string ("") in this parameter.</p>
<b>szIconPath</b>	Specify the name of an alternate icon file or a valid executable file that contains the new icon.
<b>nlcon</b>	If you specified an executable file for szIconPath, specify the icon index in the executable file. Otherwise, enter the number 0 for nlcon.
<b>szShortCutKey</b>	Specify the string that contains the shortcut key sequence the end user can press to start the program. For example, if you want the end user to be able to open the application by pressing the "Ctrl," the "Alt," and then the "1" key, specify "Ctrl + Alt + 1" in this parameter.

**Table 68** • ReplaceShortcut Parameters (cont.)

Parameter	Description
<b>nFlag</b>	<p>Pass one or more of the following predefined constants in this parameter. To pass two or more predefined constants in this parameter, combine those constants with the bitwise OR operator (   ).</p> <ul style="list-style-type: none"> <li>• <b>CS_OPTION_FLAG_REPLACE_EXISTING</b>—Replace an existing shortcut.</li> <li>• <b>CS_OPTION_FLAG_RUN_MAXIMIZED</b>—The target of the shortcut is maximized when launched.</li> <li>• <b>CS_OPTION_FLAG_RUN_MINIMIZED</b>—The target of the shortcut is minimized when launched.</li> <li>• <b>CS_OPTION_FLAG_PREVENT_PINNING</b>—Do not allow the shortcut to be pinned to the Start menu or taskbar on Windows Vista or later systems. This option hides the context menu commands that enable end users to pin the shortcut to the taskbar and to the Start menu.  You may want to prevent pinning for shortcuts that are for tools and secondary products that are part of your installation.</li> <li>• <b>CS_OPTION_FLAG_NO_NEW_INSTALL_HIGHLIGHT</b>—Do not highlight the shortcut as newly installed after end users install your product on Windows Vista or later systems. This has the same effect as clearing the <b>Highlight newly installed programs</b> check box in the Customize Start Menu dialog box for an individual item on a target system.  You may want to use this option for shortcuts that are for tools and secondary products that are part of your installation.</li> <li>• <b>CS_OPTION_FLAG_NO_STARTSCREEN_PIN</b>—Do not pin the shortcut to the Start screen by default on Windows 8 target systems. If you pass this constant, the installation sets a Windows Shell property that was introduced in Windows 8.  You may want to prevent pinning for shortcuts that are for tools and secondary products that are part of your installation.</li> <li>• <b>NULL</b>—Indicates no options.</li> </ul> <p>For more information on CS_OPTION_FLAG_PREVENT_PINNING and CS_OPTION_FLAG_NO_STARTSCREEN_PIN, see the <a href="#">Additional Information section</a>.</p>

## Return Values

**Table 69** • ReplaceShortcut Return Values

Return Value	Description
0	Indicates that the function successfully replaced the shortcut.
< 0	Indicates that the function was unable to replace the shortcut.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

## Additional Information

Note the following details about two of the nFlag constants.

### CS\_OPTION\_FLAG\_PREVENT\_PINNING

If you configure the shortcut to prevent pinning to the taskbar and the Start menu, the target of the shortcut is ineligible for inclusion in the most frequently used list on the Start menu.

Shortcuts that contain certain strings cannot be pinned to the taskbar or the Start menu, and they cannot be displayed in the most frequently used list. Examples are:

- Documentation
- Help
- Install
- Remove
- Setup
- Support

### CS\_OPTION\_FLAG\_NO\_STARTSCREEN\_PIN

Note that Windows 8 maintains information about shortcut pinning to the Start screen after a shortcut is removed by uninstalling the application. Therefore, the CS\_OPTION\_FLAG\_NO\_STARTSCREEN\_PIN constant has no effect on the target system if the shortcut has already been installed on it. Thus, when you are testing this functionality, ensure that you test on a clean machine—one on which this shortcut and its target have never been installed.

## ReplaceShortcut Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ReplaceShortcut function.
```



```

*
* Note: In order for this script to run correctly, you must set
*       preprocessor constants to valid file names and a path
*       on the target system. To easily create this example
*       shortcut, run the CreateShortcut example 3.
*
\*-----*/

#define FOLDER      "C:\\Windows\\"
#define NEW_PROGRAM "C:\\WINDOWS\\WRITE.EXE"
#define NEW_PARAM   "C:\\WINDOWS\\README.TXT"

function OnFirstUIAfter()
    STRING szShortcutFolder, szName, szNewItem, szCmdLine, szWorkingDir;
    STRING szIconPath, szShortCutKey, szProgram, szParam;
    NUMBER nIcon, nFlag;
begin

    szShortcutFolder = FOLDER ^ "Example folder 3";
    szName           = "Notepad Example 3";
    szNewItem        = "New Wordpad Example";

    // Make sure the space is not seen as a delimiter.
    szProgram = NEW_PROGRAM;
    LongPathToQuote (szProgram, TRUE);

    szParam = NEW_PARAM;
    LongPathToShortPath(szParam);

    szCmdLine    = szProgram + " " + szParam;
    szWorkingDir = "";
    szIconPath   = "";
    nIcon        = 0;
    szShortCutKey = "";
    nFlag        = CS_OPTION_FLAG_REPLACE_EXISTING|CS_OPTION_FLAG_RUN_MAXIMIZED;

    // Display the folder on the screen.
    ShowProgramFolder (szShortcutFolder, SW_SHOW);

    // Replace the Notepad Example 3 shortcut with New Wordpad Example.
    if (ReplaceShortcut (szShortcutFolder, szName, szNewItem, szCmdLine,
                        szWorkingDir, szIconPath, nIcon, szShortCutKey,
                        nFlag) < 0) then
        MessageBox ("ReplaceShortcut failed.", SEVERE);
    else
        MessageBox ("Shortcut successfully replaced.", INFORMATION);
    endif;

end;

```

## Resize

The **Resize** function resizes an InstallScript array.

### Syntax

```
Resize ( Array , nNewSize );
```

## Parameters

**Table 70** • Resize Parameters

Parameter	Description
<b>Array</b>	Specifies the name of the array variable.
<b>nNewSize</b>	Specifies the new size to be given to the array.

## Return values

The **Resize** function returns the new size of the array.

# RGB

The **RGB** function creates a custom color value that can be used with [SetColor](#) and [SetTitle](#).

## Syntax

```
RGB ( constRed, constGreen, constBlue );
```

## Parameters

**Table 71** • RGB Parameters

Parameter	Description
<b>constRed</b>	Specifies a numeric constant, ranging in value from 0 to 255, that indicates the amount of red in the custom color.
<b>constGreen</b>	Specifies a numeric constant, ranging in value from 0 to 255, that indicates the amount of green in the custom color.
<b>constBlue</b>	Specifies a numeric constant, ranging in value from 0 to 255, that indicates the amount of green in the custom color.

## Return Values

This function returns a number value for a custom color that can be used when calling [SetColor](#) and [SetTitle](#).

## RGB Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the RGB function.
 *
 * The first call to RGB returns the value of the background
 * color to Grey; the second call returns the background color
 * to red; the third call returns the background color to blue.
 * The values returned by the calls to RGB are passed to
 * SetColor to change the background color.
 *
 \*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_RGB(HWND);

function ExFn_RGB(hMSI)
begin

    Enable ( BACKGROUND );

    // Change the background color to light grey.
    SetColor (BACKGROUND, RGB(198,198,198));

    Delay (3);

    // Change the background color to red.
    SetColor (BACKGROUND, RGB(255,0,0));

    Delay (3);

    // Change the background color to blue.
    SetColor (BACKGROUND, RGB(0, 0, 255));

    Delay (3);

end;

```



# Built-In Functions (S-T)

For a list of functions by category, see [Built-In Functions by Category](#).

## SdAskDestPath



---

**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdAskDestPath** function creates a dialog that allows the end user to select an alternate destination path. When the end user clicks Browse in this dialog, the [SelectDir](#) function is called to open a second dialog that enables the end user either to select an existing folder or to enter a new folder name.

### Syntax

```
SdAskDestPath ( szTitle, szMsg, svDir, nReserved );
```

## Parameters

**Table 1 • SdAskDestPath Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Choose Folder"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the text to display in the dialog. The text is considered a static control. Use the %P place holder in your message string to insert the product name (if any) that has been specified by a previous call to <a href="#">SdProductName</a> . To display the default instructions for this dialog, pass a null string ("").
<b>svDir</b>	Specifies the name of the directory to be selected by default. Returns the name of the directory selected by the end user.
<b>nReserved</b>	Pass zero in this parameter. No other value is allowed.

## Return Values

**Table 2 • SdAskDestPath Return Values**

Return Value	Description
<b>NEXT (1)</b>	Indicates that the end user clicked the Next button.
<b>BACK (12)</b>	Indicates that the end user clicked the Back button.

## Additional Information

- To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.
- If the end user specifies an incomplete, invalid, or write-protected path in the second dialog, an error message is displayed. If you want the end user to be able to select folders that are not writable, call the [AskPath](#) function instead.
- Setups that run in silent mode should create the new folder if it does not exist before calling SdAskDestPath. This ensures that the confirmation dialog is not displayed. Without this step, two response files are required to handle the two possible conditions.
- In earlier versions of InstallShield Professional, when the end user selected in the Choose Folder dialog a folder that did not exist, a confirmation message box was displayed asking whether the folder should be created. This message box proved to be confusing to many end users, so it has been removed from InstallShield.

## SdAskDestPath Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdAskDestPath function.
 *
 * SdAskDestPath is called to prompt the user for a path.
 * Then the selected path is assigned to the system variable
 * INSTALLDIR, which is displayed in a message box.
 *
 \*-----*/

#define TITLE_TEXT "SdAskDestPath Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

function OnBegin()
    STRING svDir;
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Set default folder name for call to SdAskDestPath.
    svDir = "C:\\ISEXampl\\Target";

    // Display the SdAskDestPath dialog. Pass a null string
    // in the second parameter to display the default message.
    if (SdAskDestPath (TITLE_TEXT, "", svDir, 0) = NEXT) then
        INSTALLDIR = svDir;
    endif;

    // Display the new target directory.
    sprintfBox (INFORMATION, "SdAskDestPath", "Successful.\n\nThe Target " +
        "directory is: " + INSTALLDIR);

end;

```

## SdAskDestPath2



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdAskDestPath** function creates a dialog that allows the end user to select an alternate destination path. When you click the Change button in that dialog, the [SelectDir](#) function is called to open a second dialog that enables the end user either to select an existing folder or to enter a new folder name.

## Syntax

```
SdAskDestPath2 ( szTitle, szMsg, svDir );
```

## Parameters

**Table 3 • SdAskDestPath2 Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Choose Destination Location"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the text to display in the dialog. The text is considered a static control. Use the %P place holder in your message string to insert the product name (if any) that has been specified by a previous call to <a href="#">SdProductName</a> . To display the default instructions for this dialog, pass a null string ("").
<b>svDir</b>	Specifies the name of the directory to be selected by default. Returns the name of the directory selected by the end user.

## Return Values

**Table 4 • SdAskDestPath2 Return Values**

Return Value	Description
<b>NEXT</b>	Indicates that the user selected the Next button.
<b>BACK</b>	Indicates that the user selected the Back button.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the dialog could not be displayed.

## Additional Information

- To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.
- If the end user specifies an incomplete, invalid, or write-protected path in the second dialog, an error message is displayed. If you want the end user to be able to select folders that are not writable, call the [AskPath](#) function instead.
- Installations that run in silent mode should create the new folder if it does not exist before calling SdAskDestPath2. This ensures that the confirmation dialog is not displayed. Without this step, two response files are required to handle the two possible conditions.

## SdAskDestPath2 Example



**Project •** This information applies to the following project types:



- *InstallScript*
- *InstallScript MSI*

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdAskDestPath2 function.
 *
 * SdProductName is called to set the product name so that it
 * can be displayed in place of the %P placeholder in the
 * SdAskDestPath2 dialog. Then, SdAskDestPath2 is called to
 * prompt the user for a path. Finally, the selected path
 * is assigned to the system variable TARGETDIR, which is
 * displayed in a message box.
 *
\*-----*/

#define TITLE_TEXT "SdAskDestPath2 Example"

    STRING svDir;

#include "ifx.h"

function OnBegin()
begin

    // Set product name.
    SdProductName ("Example Product 5.2");

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Set default folder name for call to SdAskDestPath2.
    svDir = "C:\\ISEXamp1\\Target";

    // Display the SdAskDestPath2 dialog. Pass a null string
    // in the second parameter to display the default message.
    if (SdAskDestPath2 (TITLE_TEXT, "", svDir ) = NEXT) then
        TARGETDIR = svDir;
    endif;

    // Display the new target directory.
    sprintfBox (INFORMATION, "SdAskDestPath2", "Successful.\n\nThe Target " +
        "directory is: " + TARGETDIR);

end;

```

## SdAskOptions



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdAskOptions** function creates a dialog that offers installation options. You can use check boxes or option buttons as the selection mechanism. The information shown beside the button is retrieved from a group of options. The default number of options is four. You can add or subtract the number of options as necessary in the group.

## Syntax

```
SdAskOptions ( szTitle, szMsg1, szMsg2, szId, szFeatures, nExclusiveFlag );
```

## Parameters

**Table 5 • SdAskOptions Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Select Components"), pass a null string ("") in this parameter.
<b>szMsg1</b>	Specifies a message to display in the dialog. This static field has an ID of 801. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>szMsg2</b>	Specifies a second message to display in the dialog. This static field has an ID of 802.
<b>szId</b>	Specifies an alternate numeric dialog ID. Use only numeric IDs expressed in string form (for example, ID 13001 as "13001"). You can copy the SdAskOptions dialog resource, make limited changes to it, give it a unique numeric ID, and call that dialog by passing its ID as a string in szId. (See above.) To create the standard four-option SdAskOptions dialog, pass a null string ("") in this parameter.
<b>szFeatures</b>	<p>Specifies the name of the feature that contains the subfeatures to be displayed. The subfeatures are preceded by check boxes or option buttons. To display all top-level features, pass a null string ("") in this parameter.</p> <p>SdAskOptions searches for the requested features in the file media library or script-created feature set specified by the system variable <a href="#">MEDIA</a>.</p>
<b>nExclusiveFlag</b>	<p>Specifies the type of button you want to display in the dialog. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>EXCLUSIVE</b>—Specifies option buttons.</li> <li>• <b>NONEXCLUSIVE</b>—Specifies check boxes.</li> </ul>

## Return Values

**Table 6 • SdAskOptions Return Values**

Return Value	Description
<b>NEXT (1)</b>	Indicates that the end user clicked the Next button.
<b>BACK (12)</b>	Indicates that the end user clicked the Back button.

## Additional Information

- To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

- If your setup includes required, visible components, do not call SdAskOptions to obtain installation options. Instead, call FeatureDialog, SdFeatureDialog, SdFeatureDialogAdv, SdFeatureMult, or SdAskOptionsList in nonexclusive mode.
- If your setup does not use a setup type dialog, you must call FeatureSetupTypeSet to specify a setup type that has been defined in the IDE's Setup Types view before calling SdAskOptions.
- SdAskOptions operates on the current media, which is specified by the system variable MEDIA. During setup initialization, the installation assigns to MEDIA a media name that is associated with your file media library (**Data1.cab**).

**Task****To display script-created features:**

1. Save the current value of MEDIA in a string variable, for example, szSaveMEDIAValue.
2. Assign to MEDIA the name of the script-created component set.
3. Call SdAskOptions to get end-user selections.
4. Assign to MEDIA the value that you saved in step 1. You must do this before calling FeatureTransferData.

You can create more than one dialog of the SdAskOptions type by copying the SdAskOptions dialog resource (located in \_isres.dll) using a resource editor, making limited changes to the copy, and giving it a unique ID. You should save the copy to \_isuser.dll. When you call SdAskOptions and pass the ID of the customized copy of the dialog in the parameter szId, the customized copy is displayed. Limit your changes to editing existing static text fields and adding static text fields. Adding controls that require handling is not recommended because it requires changing the SdAskOptions source script.

## SdAskOptions Example



**Project** • This information applies to the following project types:

- InstallScript
- InstallScript MSI

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdAskOptions function
 *
 * This script displays a dialog that offers installation
 * options.
 *
 * Note: To run this example script, create a project (or
 *       insert into a project) with several features and/or
 *       subfeatures.
 *
 *-----*/

// Specify your feature name here. These are the names you gave to your
// features in the IDE. A NULL ("" ) string specifies base features.
#define FEATURE ""
```

```

#define SDASKOPTSTITLE      "Component Selection"
#define SDASKOPTSMMSG1      "Select components to install."
#define SDASKOPTSMMSG2      "Your selections will be used to effect file transfer."
#define APPBASE_PATH        "Your Company Name\\Your Product Name"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "ifx.h"

function OnFirstUIBefore()
begin

    // Set a default destination path.
    INSTALLDIR = PROGRAMFILES ^ APPBASE_PATH;

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Get installation options.
    SdAskOptions (SDASKOPTSTITLE, SDASKOPTSMMSG1, SDASKOPTSMMSG2,
        "", FEATURE, NONEXCLUSIVE);

end;

```

## SdAskOptionsList



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdAskOptionsList** function creates a dialog that displays a list of features for a custom installation.

### Syntax

```
SdAskOptionsList ( szTitle, szMsg, szFeatures, nStyle );
```

## Parameters

**Table 7 • SdAskOptionsList Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Select Components"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message to display in the dialog. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>szFeatures</b>	<p>Specifies the name of the feature that contains the subfeatures to be displayed. The subfeatures are preceded by check boxes or option buttons. To display all top-level features, pass a null string ("") in this parameter.</p> <p>SdAskOptionsList searches for the requested feature(s) in the file media library or script-created feature set specified by the system variable <a href="#">MEDIA</a>.</p>
<b>nStyle</b>	<p>Specifies whether the end user's selection is limited. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>EXCLUSIVE</b>—Allows the end user to select only one item from the list. Do not use EXCLUSIVE mode if any of szFeatures' subfeatures are required features.</li> <li>• <b>NONEXCLUSIVE</b>—Allows the end user to select more than one item from the list, including multiple non-contiguous selections. Also displays two buttons—Select All and Clear All—which allow selection of all options or clearing of all selections.</li> </ul>

## Return Values

**Table 8 • SdAskOptionsList Return Values**

Return Value	Description
<b>NEXT (1)</b>	Indicates that the end user clicked the Next button.
<b>BACK (12)</b>	Indicates that the end user clicked the Back button.

## Additional Information

- To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.
- If your setup does not use a setup type dialog, you *must* call FeatureSetupTypeSet to specify a setup type that has been defined in the IDE's Setup Types view before calling SdAskOptionsList.
- SdAskOptionsList operates on the current media, which is specified by the system variable MEDIA. During setup initialization, the installation assigns to MEDIA a media name that is associated with your file media library (**Data1.cab**).

**Task****To display script-created features:**

1. Save the current value of MEDIA in a string variable, for example, szSaveMEDIAValue.
2. Assign to MEDIA the name of the script-created component set.
3. Call SdAskOptionsList to get end-user selections.
4. Assign to MEDIA the value that you saved in step 1. You must do this before calling FeatureTransferData.

## SdAskOptionsList Example



**Project** • This information applies to the following project types:

- InstallScript
- InstallScript MSI

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdAskOptionsList function with the
 * NONEXCLUSIVE and EXCLUSIVE options.
 *
 \*-----*/

// Define strings. In a real setup you would define these in your string
// tables and precede each constant with @ to use them in your script.
#define COMP_SELECT_TITLE          "Select Components"
#define COMP_SELECT_MSG            "Select components to install."
#define MY_FEATURE_NAME "DefaultFeature"

#include "ifx.h"

function OnFirstUIBefore()
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Let user select from top-level components, nonexclusively.
    SdAskOptionsList(COMP_SELECT_TITLE, COMP_SELECT_MSG + " NONEXCLUSIVE", "", NONEXCLUSIVE);

    // Let user select from subcomponents of MY_FEATURE_NAME, exclusively.
    SdAskOptionsList(COMP_SELECT_TITLE, COMP_SELECT_MSG + " EXCLUSIVE", MY_FEATURE_NAME, EXCLUSIVE);

end;
```

# SdBitmap



---

**Project** • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

The **SdBitmap** function displays a bitmap on a dialog. The maximum allowable size of the bitmap is 440 pixels wide by 275 pixels high. You can also display a message in the SdBitmap dialog, but only if you use a resource editor to modify the SdBitmap dialog resource so that the control that displays the message is made visible.

## Syntax

```
SdBitmap ( szTitle, szMsg, szBitmap );
```



## Parameters

**Table 9 • SdBitmap Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Welcome"), pass a null string ("") in this parameter.
<b>szMsg</b>	Pass a null string ("") in this parameter unless you use a resource editor to modify the SdBitmap dialog to display a message. See the Additional Information section, below.
<b>szBitmap</b>	<p>Specifies the file name of the bitmap to display and, optionally, a set of bitmap attributes. If bitmap attributes are included, the string passed in this parameter should be formatted as follows:</p> <pre>"bitmap file name;transparent flag;3-D flag;Background color"</pre> <ul style="list-style-type: none"> <li>• <b>bitmap file name</b>—Specifies the name of the bitmap file. If the file name is unqualified (that is, if it does not include a drive designation and path), InstallShield searches for the bitmap in <a href="#">SUPPORTDIR</a>.</li> <li>• <b>transparent flag</b>—Indicates whether to display the bitmap transparently. When this flag is 1 (true), all portions of the bitmap that are magenta (RGB Value: 255,0,255) will be displayed transparently. The default for this parameter is 0 (non-transparent).</li> <li>• <b>3-D flag</b>—Indicates whether to add a 3-D border around the edges of the static field that contains the bitmap. The default for this parameter is 0 (no 3D border).</li> <li>• <b>background color</b>—Indicates the color to use for the background of the static text field. Note that this color will be visible only if the bitmap is smaller than the static text field in which it appears or if the transparent flag is set to 1 and the bitmap has transparent areas. The background color must be expressed as an RGB value, that is, as three numeric values separated by commas.</li> </ul> <p>The following example displays the bitmap from the file MyBitmap.bmp, which is located in the SUPPORTDIR folder. The bitmap is displayed on a black background and has a three-dimensional border. Any parts of the bitmap that are magenta are displayed in the background color of black.</p> <pre>"MyBitmap.bmp;1;1;0,0,0"</pre>

## Return Values

**Table 10 • SdBitmap Return Values**

Return Value	Description
<b>NEXT (1)</b>	Indicates that the end user clicked the Next button.

**Table 10 • SdBitmap Return Values (cont.)**

Return Value	Description
<b>BACK (12)</b>	Indicates that the end user clicked the Back button.

## Additional Information

You can use a resource editor to modify the SdBitmap dialog resource so that a message string passed as the parameter `szMsg` is displayed in the SdBitmap dialog.

The SdBitmap dialog resource is contained in `_isres.dll`. The resource contains a static text control that receives the string passed as the parameter `szMsg`. However, by default this static text control is out of view in the SdBitmap dialog (below the dialog). SdBitmap also uses a static text control to display the bitmap image. You can resize the bitmap image static text control and move the message static text control into view in the dialog. The message in `szMsg` will then be visible when SdBitmap is called.

Be aware that changing the size of the bitmap image static text control may affect the display of your bitmap image. The bitmap image must be small enough to avoid being clipped when SdBitmap centers it in the bitmap image static text control.

This function does not support transparent bitmaps. If you use a transparent bitmap with this function, the transparent portions are displayed normally.

Metafiles are not supported by SdBitmap.

## SdBitmap Example



**Project •** This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the SdBitmap function.
*
* Note: Before running this script, set the defined constant
*       BITMAP_FILE so that it references a bitmap file
*       included in the Support Files/Billboards view.
*
\*-----*/

// The bitmap to display.
#define BITMAP_FILE SUPPORTDIR ^ "MyBitmap.bmp"

// The title to use for the SdBitmap dialog.
#define TITLE_TEXT "SdBitmap Example"

#include "Ifx.h"

function OnBegin()
```

```

begin
    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Display the specified bitmap in a dialog. Pass a
    // null string in the second parameter because the dialog
    // has not been customized to display a message,
    SdBitmap (TITLE_TEXT, "", BITMAP_FILE);

end;

```

## SdConfirmNewDir



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdConfirmNewDir** function creates a dialog that displays a folder name and prompts for confirmation. If the end user clicks the Yes button, this function creates the new folder automatically.

### Syntax

```
SdConfirmNewDir ( szTitle, szDir, nReserved );
```

## Parameters

**Table 11** • SdConfirmNewDir Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Confirm New Folder"), pass a null string ("") in this parameter.
<b>szDir</b>	Specifies the name of the directory to confirm. (Obtain this information by calling SdAskDestPath.)
<b>nReserved</b>	Pass zero in this parameter. No other value is allowed.

## Return Values

**Table 12** • SdConfirmNewDir Return Values

Return Value	Description
<b>YES (1)</b>	Indicates that the Yes button was clicked. The directory has been confirmed and will be created.
<b>NO (0)</b>	Indicates that the No button was clicked. The specified directory will not be created.
<b>&lt; 0</b>	Indicates that the Yes button was clicked but the function was unable to create the new directory.

## Additional Information

- To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.
- The dialog that is displayed by this function cannot be displayed with a skin; it appears the same regardless of whether you have specified a skin.

## SdConfirmNewDir Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the SdConfirmNewDir function.
*
```

```

* This example script first calls SdAskDestPath to get the
* destination folder from the user. If the folder does not
* exist, SdConfirmNewDir is then called to ask if the user
* wants to create the folder.
*
* Note: This script creates directories on the local hard disk.
*
\*-----*/

#define DEFAULT_TARGET_FOLDER "C:\\NONEXIST\\DIR";
#define TITLE_TEXT "SdConfirmNewDir Example"

#include "ifx.h"

function OnBegin()
    NUMBER nResult;
    STRING szMsg, svDir;
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

start:
    // Set up parameters for call to SdAskDestPath.
    szMsg = "Select destination folder:";
    svDir = DEFAULT_TARGET_FOLDER

    // Retrieve destination folder from user.
    nResult = SdAskDestPath (TITLE_TEXT, szMsg, svDir, 0);

    // Check if the selected folder exists.
    if (ExistsDir (svDir) = EXISTS) then
        // Inform user that the specified folder already exists.
        szMsg = "folder '%s' already exists.\n\nIn order for this example to " +
            "run properly, please specify a nonexisting folder.";
        SprintfBox (INFORMATION, TITLE_TEXT, szMsg, svDir);

        // Start over.
        goto start;
    else
        // The specified folder does not exist. Request user
        // confirmation to create it.
        nResult = SdConfirmNewDir (TITLE_TEXT, svDir, 0);

        if (nResult = NO) then
            // The user did not want it created select.
            MessageBox ("Selected folder was not created.", INFORMATION);

            // Start over.
            goto start;
        elseif (nResult = YES) then
            // The user wants to create the folder.
            SprintfBox (INFORMATION, TITLE_TEXT, "%s created.", svDir);
        elseif (nResult < 0) then
            // Report the error; then terminate.
            MessageBox ("SdConfirmNewDir failed.", SEVERE);
            abort;
        endif;
    endif;
endif;

```

```
end;
```

## SdConfirmRegistration



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdConfirmRegistration** function creates a message box that displays the User Name, Company Name, and Serial Number. If a null string ("") is entered in any field in the dialog, the displayed field will be empty.

### Syntax

```
SdConfirmRegistration ( szTitle, szName, szCompany, szSerial, nReserved );
```

### Parameters

**Table 13** • SdConfirmRegistration Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Registration Confirmation"), pass a null string ("") in this parameter.
<b>szName</b>	Specifies the end user's name.
<b>szCompany</b>	Specifies the company name.
<b>szSerial</b>	Specifies the serial number. If this parameter contains a null string (""), the serial number field is not displayed in the dialog.
<b>nReserved</b>	Pass zero in this parameter. No other value is allowed.

### Return Values

**Table 14** • SdConfirmRegistration Return Values

Return Value	Description
<b>YES (1)</b>	Indicates that the Yes button was clicked.
<b>NO (0)</b>	Indicates that the No button was clicked.

### Additional Information

- To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

- The dialog that is displayed by this function cannot be displayed with a skin; it appears the same regardless of whether you have specified a skin.
- To obtain the serial number and end user's name and company, call **SdRegisterUserEx**. To obtain only the end user's name and company, call **SdRegisterUser**.

## SdConfirmRegistration Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdRegisterUser and SdConfirmRegistration
 * functions.
 *
 * SdRegisterUser is called to prompt for the user's name
 * and company name. These entries are then confirmed when
 * SdConfirmRegistration is called.
 *
 \*-----*/

#define REG_TITLE      "SdRegisterUser Example"
#define REG_MSG        "Please register your product now."
#define CONFIRM_TITLE  "SdConfirmRegistration Example"

#include "Ifx.h"

function OnBegin()
    STRING  svName, svCompany;
    NUMBER  nResult;
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    repeat
        // Get the user's name and company name.
        SdRegisterUser (REG_TITLE, REG_MSG, svName, svCompany);

        // Confirm that the information is correct. Pass a null string in
        // parameter four since SdRegisterUser does not get a serial number.
        nResult = SdConfirmRegistration (CONFIRM_TITLE, svName, svCompany, "", 0);
    until nResult = YES;

end;
```

# SdCustomerInformation



---

**Project** • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

The **SdCustomerInformation** function displays a dialog that enables the end user to specify the user name and company name for the product being installed. The dialog may also include radio buttons that let the end user specify whether the product should be installed for all users or only the current user.

You can specify default values for these fields by specifying the appropriate parameters. If you specify a null string (""), the function uses the appropriate script variable.

The Next button becomes enabled only when data exists in both edit fields. The end user cannot leave any field blank.

## Syntax

```
SdCustomerInformation ( szTitle, svName, svCompany, bvAllUsers );
```



## Parameters

**Table 15 • SdCustomerInformation Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Customer Information"), pass a null string ("") in this parameter.
<b>svName</b>	<p>Specifies the default value for the Name edit field when the function is called.</p> <p>If a null string ("") is specified, the default value is the current value of the <a href="#">IFX_PRODUCT_REGISTEREDOWNER</a> variable. For an InstallScript project, this variable is read from the registry in a first-time installation or the corresponding text substitution in maintenance mode. For an InstallScript MSI project, the default value of the variable is always read from the Windows Installer property <b>USERNAME</b>.</p> <p>The function returns the value that the end user specified in this parameter. In an InstallScript installation, the function also sets the value of <i>IFX_PRODUCT_REGISTEREDOWNER</i> to the value that the end user specified. In an InstallScript MSI installation, the function automatically updates the Windows Installer property <b>USERNAME</b>.</p>
<b>svCompany</b>	<p>Specifies the default value for the Company edit field when the function is called.</p> <p>If a null string ("") is specified, the default value is the current value of the <a href="#">IFX_PRODUCT_REGISTEREDCOMPANY</a> variable. For an InstallScript project, this variable is read from the registry in a first-time installation or the corresponding text substitution in maintenance mode. For an InstallScript MSI project, the default value of the variable is always read from the Windows Installer property <b>COMPANYNAME</b>.</p> <p>The function returns the value that the end user specified in this parameter. In an InstallScript installation, the function also sets the value of <i>IFX_PRODUCT_REGISTEREDCOMPANY</i> to the value that the end user specified. In an InstallScript MSI installation, the function automatically updates the Windows Installer property <b>COMPANYNAME</b>.</p>

**Table 15 • SdCustomerInformation Parameters (cont.)**

Parameter	Description
<b>bvAllUsers</b>	<p>Returns which option the end user selected. After the function returns, bvAllUsers is set to one of the following values:</p> <ul style="list-style-type: none"> <li>• <b>TRUE</b>—The <b>Anyone who uses this computer [all users]</b> option is selected. If the end user selects this option in an InstallScript installation, the function sets the system variable <b>ALLUSERS</b> to a non-zero value. If the end user selects this option in an InstallScript MSI installation, the function sets the <b>ALLUSERS</b> property to a value of 2.</li> <li>• <b>FALSE</b>—The <b>Only for me [user name]</b> option is selected. If the end user selects this option in an InstallScript installation, the function sets the ALLUSERS variable to FALSE.</li> </ul> <p>The default option is not based on the current value of the bvAllUsers parameter, but on the <b>ALLUSERS</b> Windows Installer property in an InstallScript MSI installation, or on the ALLUSERS system variable in an InstallScript installation:</p> <ul style="list-style-type: none"> <li>• If the <b>ALLUSERS</b> property is 2 or the ALLUSERS system variable is non-zero, the all-users option is selected by default.</li> <li>• If the <b>ALLUSERS</b> property is 1 or the ALLUSERS system variable is FALSE, the per-user option is selected by default.</li> </ul> <p>One or both of the radio buttons can be disabled or hidden by updating script variables as follows:</p> <ul style="list-style-type: none"> <li>• <b>DISABLE_PERUSERBTN</b>—Indicates that the per-user option should be disabled (or hidden if HIDE_DISABLED_BTNS is TRUE) in cases where it would normally be enabled. The default value of this variable is FALSE. Note that the per-user option is always hidden on Windows 9x platforms, regardless of the value of this variable.</li> <li>• <b>DISABLE_ALLUSERBTN</b>—Indicates that the all-users option should be disabled (or hidden) in cases where it would normally be enabled. The default value of this variable is FALSE. Note that the all-users option is always hidden if the installation is being run without administrator or power-user privileges, regardless of the value of this variable.</li> <li>• <b>HIDE_DISABLED_BTNS</b>—Indicates that both options should be hidden instead of being disabled. The default value of this variable is TRUE. Note that when this variable is set to TRUE, both options are hidden if either option is determined to be disabled.</li> </ul>

## Return Values

**Table 16 •** SdCustomerInformation Return Values

Return Value	Description
<b>NEXT (1)</b>	Indicates that the Next button was clicked.
<b>BACK (12)</b>	Indicates that the Back button was clicked.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdCustomerInformation Example



**Project •** This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdCustomerInformation function.
 * SdCustomerInformation prompts the end user to enter a user
 * name and company name, and to specify whether the installation is
 * for anyone who uses the target system or for the current user only.
 *
 \*-----*/

#include "ifx.h"

function OnFirstUIBefore( )
    // ...other variable declarations...
    STRING  svName, svCompany, szMsg;
    NUMBER  nvUser, nReturn;
begin

    // ...show other dialogs...

    // get the end user's name and company name
    SdCustomerInformation("", svName, svCompany, nvUser);

    if (nvUser = 0) then
        szMsg = "per-user installation";
    else
        szMsg = "all-users installation";
    endif;

    MessageBox("You entered:\n\n" +
```

```

        "Name: " + svName + "\n" +
        "Company: " + svCompany + "\n" +
        "Type: " + szMsg,
        INFORMATION);

// ...other dialogs...

end;

```

## SdCustomerInformationEx



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdCustomerInformationEx** function displays a dialog that enables the end user to specify the user name, company name, and serial number for the product being installed. The dialog may also include radio buttons that let the end user specify whether the product should be installed for all users or only the current user.

You can specify default values for these fields by specifying the appropriate parameters. If you specify a null string (""), the function uses the appropriate script variable.

The Next button becomes enabled only when data exists in all three edit fields. The end user cannot leave any field blank.



**Note** • The **SdCustomerInformationEx** function does not verify the serial number. To learn how to add code that verifies the serial number, see the sample serial number validation project. This sample project is in one of the Samples subfolders within the InstallShield Program Files folder. The default installation location is **C:\Program Files\InstallShield\2016\Samples\InstallScript\Serial Number Validation Sample Project**.

### Syntax

```
SdCustomerInformationEx ( szTitle, svName, svCompany, svSerial, bvAllUsers );
```

## Parameters

**Table 17 • SdCustomerInformationEx Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Customer Information"), pass a null string ("") in this parameter.
<b>svName</b>	<p>Specifies the default value for the Name edit field when the function is called.</p> <p>If a null string ("") is specified, the default value is the current value of the <a href="#">IFX_PRODUCT_REGISTEREDOWNER</a> variable. For an InstallScript project, this variable is read from the registry in a first-time installation or the corresponding text substitution in maintenance mode. For an InstallScript MSI project, the default value of the variable is always read from the Windows Installer property <b>USERNAME</b>.</p> <p>The function returns the value that the end user specified in this parameter. In an InstallScript installation, the function also sets the value of <i>IFX_PRODUCT_REGISTEREDOWNER</i> to the value that the end user specified. In an InstallScript MSI installation, the function automatically updates the Windows Installer property <b>USERNAME</b>.</p>
<b>svCompany</b>	<p>Specifies the default value for the Company edit field when the function is called.</p> <p>If a null string ("") is specified, the default value is the current value of the <a href="#">IFX_PRODUCT_REGISTEREDCOMPANY</a> variable. For an InstallScript project, this variable is read from the registry in a first-time installation or the corresponding text substitution in maintenance mode. For an InstallScript MSI project, the default value of the variable is always read from the Windows Installer property <b>COMPANYNAME</b>.</p> <p>The function returns the value that the end user specified in this parameter. In an InstallScript installation, the function also sets the value of <i>IFX_PRODUCT_REGISTEREDCOMPANY</i> to the value that the end user specified. In an InstallScript MSI installation, the function automatically updates the Windows Installer property <b>COMPANYNAME</b>.</p>
<b>svSerial</b>	<p>Specifies the default value for the Serial Number edit field when the function is called.</p> <p>If a null string ("") is specified, the default value is the current value of the <a href="#">IFX_PRODUCT_REGISTEREDSERIALNUM</a> variable. This variable is read from the registry in a first-time installation or the corresponding text substitution in maintenance mode.</p> <p>The function returns the value that the end user specified in this parameter. The function also sets the value of <i>IFX_PRODUCT_REGISTEREDSERIALNUM</i> to the value that the end user specified.</p>

**Table 17 •** SdCustomerInformationEx Parameters (cont.)

Parameter	Description
<b>bvAllUsers</b>	<p>Returns which option the end user selected. After the function returns, bvAllUsers is set to one of the following values:</p> <ul style="list-style-type: none"> <li> <b>TRUE</b>—The <b>Anyone who uses this computer [all users]</b> option is selected.           <p>If the end user selects this option in an InstallScript installation, the function sets the system variable <b>ALLUSERS</b> to a non-zero value.</p> <p>If the end user selects this option in an InstallScript MSI installation, the function sets the <b>ALLUSERS</b> property to a value of 2.</p> </li> <li> <b>FALSE</b>—The <b>Only for me [user name]</b> option is selected.           <p>If the end user selects this option in an InstallScript installation, the function sets the ALLUSERS variable to FALSE.</p> </li> </ul> <p>The default option is not based on the current value of the bvAllUsers parameter, but on the <b>ALLUSERS</b> Windows Installer property in an InstallScript MSI installation, or on the ALLUSERS system variable in an InstallScript installation:</p> <ul style="list-style-type: none"> <li>If the <b>ALLUSERS</b> property is 2 or the ALLUSERS system variable is non-zero, the all-users option is selected by default.</li> <li>If the <b>ALLUSERS</b> property is 1 or the ALLUSERS system variable is FALSE, the per-user option is selected by default.</li> </ul> <p>One or both of the radio buttons can be disabled or hidden by updating script variables as follows:</p> <ul style="list-style-type: none"> <li> <b>DISABLE_PERUSERBTN</b>—Indicates that the per-user option should be disabled (or hidden if HIDE_DISABLED_BTNS is TRUE) in cases where it would normally be enabled. The default value of this variable is FALSE. Note that the per-user option is always hidden on Windows 9x platforms, regardless of the value of this variable.           </li> <li> <b>DISABLE_ALLUSERBTN</b>—Indicates that the all-users option should be disabled (or hidden) in cases where it would normally be enabled. The default value of this variable is FALSE. Note that the all-users option is always hidden if the installation is being run without administrator or power-user privileges, regardless of the value of this variable.           </li> <li> <b>HIDE_DISABLED_BTNS</b>—Indicates that both options should be hidden instead of being disabled. The default value of this variable is TRUE. Note that when this variable is set to TRUE, both options are hidden if either option is determined to be disabled.           </li> </ul>

## Return Values

**Table 18 •** SdCustomerInformationEx Return Values

Return Value	Description
<b>NEXT (1)</b>	Indicates that the Next button was clicked.
<b>BACK (12)</b>	Indicates that the Back button was clicked.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdCustomerInformationEx Example



**Project •** This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdCustomerInformationEx function.
 * SdCustomerInformationEx prompts the end user to enter a user
 * name, company name, and serial number, and to specify whether
 * the installation is for anyone who uses the target system
 * or for the current user only.
 *
\*-----*/

#include "ifx.h"

function OnFirstUIBefore( )
    // ...other variable declarations...
    STRING  svName, svCompany, svSerial, szMsg;
    NUMBER  nvUser, nReturn;
begin

    // ...show other dialogs...

    // get the end user's name and company name
    SdCustomerInformationEx("", svName, svCompany, svSerial, nvUser);

    if (nvUser = 0) then
        szMsg = "per-user installation";
    else
        szMsg = "all-users installation";
    endif;

```

```

        MessageBox("You entered:\n\n" +
            "Name: " + svName + "\n" +
            "Company: " + svCompany + "\n" +
            "Serial number: " + svSerial + "\n" +
            "Type: " + szMsg,
            INFORMATION);

// ...other dialogs...

end;

```

## SdDiskSpace2



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdDiskSpace2** function displays a dialog that shows either of the following:

- A list view of volumes, required space, available space, and the difference between available space and required space.
- A warning message indicating that the target system does not have enough available space for the installation to take place. The dialog also displays a list view of volumes, required space, available space, and the difference between available space and required space.

The **SdDiskSpace2** function supersedes the **SdDiskSpaceRequirements** and **SdOutOfDiskSpace** functions.

### Syntax

```
SdDiskSpace2 (szTitle, szMsg, bUseOutOfSpaceDialog);
```



## Parameters

**Table 19 • SdDiskSpace2 Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Disk Space Requirements" or "Out of Disk Space"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message to display in the dialog. This text is considered a static control. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>bUseOutOfSpaceDialog</b>	<p>Indicates whether to show the dialog that warns that the installation is out of space or the dialog that shows the disk space requirements. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>TRUE</b>—Show the dialog that displays a warning message indicating that the target system does not have enough available space for the installation to take place. The dialog also displays a list view of volumes, required space, available space, and the difference between available space and required space.</li> <li>• <b>FALSE</b>—Show the dialog that displays a list view of volumes, required space, available space, and the difference between available space and required space.</li> </ul>

## Return Values

**Table 20 • SdDiskSpace2 Return Values**

Return Value	Description
<b>NEXT (1)</b>	Indicates that the end user clicked the OK button.

## Additional Information



**Project •** In InstallScript MSI installations, the OnOutOfDiskSpace event handler responds to the Out Of Disk Space event. The default implementation of OnOutOfDiskSpace displays the **SdDiskSpace2** dialog, and then aborts the installation.

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdDiskSpace2 Example



**Project •** This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdDiskSpace2 function.
 * SdDiskSpace2 displays a list view of volumes,
 * required space, available space, and the difference between
 * available and required space.
 *
 \*-----*/

#include "ifx.h"

function OnFirstUIBefore( )
begin

// ... show other dialogs ...

    // display the disk space requirements
    SdDiskSpace2 ("",
        "Review the available and required disk space " +
        "to determine where to install the application.",
        FALSE);

// ... other dialogs ...

end;

```

## SdDiskSpaceRequirements



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdDiskSpace2** function displays a list view of volumes, required space, available space, and the difference between available space and required space.

The **SdDiskSpace2** function supersedes the **SdDiskSpaceRequirements** function.

### Syntax

```
SdDiskSpaceRequirements (szTitle, szMsg);
```

## Parameters

**Table 21** • SdDiskSpaceRequirements Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Disk Space Requirements"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message to display in the dialog. This text is considered a static control. To display the default instructions for this dialog, pass a null string ("") in this parameter.

## Return Values

**Table 22** • SdDiskSpaceRequirements Return Values

Return Value	Description
<b>NEXT (1)</b>	Indicates that the end user clicked the OK button.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

# SdDisplayTopics



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdDisplayTopics** function creates a dialog that displays information based on topic data. The dialog provides a heading and then topics of titles and descriptions. You can use this dialog to display Help topics, examples, and so on.

## Syntax

```
SdDisplayTopics ( szTitle, szMsg, listTopics, listDetails, nReserved );
```

## Parameters

**Table 23 • SdDisplayTopics Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Custom Installation Help"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message to display in the dialog. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>listTopics</b>	Specifies the string list that contains the topics to display.
<b>listDetails</b>	Specifies the string list that contains a description of each topic.
<b>nReserved</b>	Pass 0 (zero) in this parameter. No other value is allowed.

## Return Values

**Table 24 • SdDisplayTopics Return Values**

Return Value	Description
<b>NEXT (1)</b>	Indicates that the Next button was clicked.
<b>BACK (12)</b>	Indicates that the Back button was clicked.

## Additional Information

- To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.
- In an InstallScript MSI installation, when the end user clicks Next the USERNAME and COMPANYNAME properties are set using the information contained in the svName and svCompany fields, respectively.
- You can modify the font style of description text to distinguish it from title (topic) text. The message and topic titles are always displayed in bold type.
- The message static field must have an ID of 801. The topic identifiers must have numbers in the range of 802 - 849. Description fields have an ID range of 851 - 899.
- The spacing of the static description fields is fixed by the size of the dialog. You cannot dynamically change the spacing in the listDetails list. If the number of topics and descriptions is less than the number of static fields, nothing appears in the white space, but the size of the dialog is not changed.

## SdDisplayTopics Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdDisplayTopics function.
 *
 * This example script creates two lists: one for topic titles,
 * one for topic descriptions. It then calls SdDisplayTopics to
 * display the topics and descriptions.
 *
\*-----*/

#define TITLE_TEXT "SdDisplayTopics Example"
#define MSG_TEXT "Custom setup options let you choose which parts of YourApp to install."

#define TOPIC1 "YourApp Program:"
#define TOPIC2 "YourApp Help:"
#define TOPIC3 "YourApp Examples:"

#define DESC1 "Includes all the files to run and write YourApp."
#define DESC2 "A computer-based tutorial demonstrating how to create programs using YourApp."
#define DESC3 "Several examples applications created using YourApp."

#include "Ifx.h"

function OnBegin()
    LIST listDescriptions, listTopics;
begin

    // Create a list for topics.
    listTopics = ListCreate (STRINGLIST);

    // Create a list for topic descriptions.
    listDescriptions = ListCreate (STRINGLIST);

    if (listTopics = LIST_NULL) || (listDescriptions = LIST_NULL) then
        // Report the error; then terminate.
        MessageBox("Unable to create lists.", INFORMATION);
        abort;
    endif;

    // Build the list of topics.
    ListAddString (listTopics, TOPIC1, AFTER);
    ListAddString (listTopics, TOPIC2, AFTER);
    ListAddString (listTopics, TOPIC3, AFTER);

    // Build the list of topic descriptions.
    ListAddString (listDescriptions, DESC1, AFTER);
    ListAddString (listDescriptions, DESC2, AFTER);
    ListAddString (listDescriptions, DESC3, AFTER);

    // Display the topics and descriptions.
    SdDisplayTopics (TITLE_TEXT, MSG_TEXT, listTopics, listDescriptions, 0);

    // Remove the lists from memory.
    ListDestroy (listTopics);

```

```
ListDestroy (listDescriptions);  
  
end;
```

## SdExceptions



---

**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdExceptions** function displays a message box that informs the end user that a shared, locked (in use), or read-only file has been encountered, and it offers appropriate options.

### Syntax

```
SdExceptions (nExceptionType, szFilename);
```

## Parameters

**Table 25 • SdExceptions Parameters**

Parameter	Description
<b>nExceptionType</b>	Specifies which type of file problem has been encountered. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>● <b>SHARED</b>—A shared file's reference count has been reduced to zero.</li> <li>● <b>READONLY</b>—A read-only file has been encountered.</li> <li>● <b>LOCKED</b>—A locked file has been encountered.</li> </ul>
<b>szFilename</b>	Specifies the name of the file that caused the problem.

## Return Values

**Table 26 • SdExceptions Return Values**

Return Value	Description
<b>ERR_RETRY (4)</b>	Indicates that the Retry button was selected.
<b>ERR_IGNORE (5)</b>	Indicates that the Ignore button was selected.
<b>ERR_YES (6)</b>	Indicates that the Yes button was selected.
<b>ERR_NO (7)</b>	Indicates that the No button was selected.
<b>ERR_PERFORM_AFTER_REBOOT (100)</b>	Indicates that the Reboot button was selected.
<b>&lt; 0</b>	Indicates that the dialog could not be displayed.

## Additional Information

The dialog that is displayed by this function cannot be displayed with a skin; it appears the same regardless of whether you have specified a skin.

## SdExceptions Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
*
* InstallShield Example Script
*
```

```

* The SdExceptions function displays a dialog informing the
* end user that a shared, locked (in use), or read-only file has
* been encountered and offering appropriate options.
*
* The SdExcpetions function is used in the default code
* for the following Miscellaneous event handlers:
*
*     OnFileLocked
*     OnFileReadOnly
*     OnRemovingSharedFile
*
* The sample script below uses the OnFileReadOnly event. To get
* the SdExceptions prompt, the setup must attempt to overwrite or
* uninstall a read-only file.
*
\*-----*/

#include "Ifx.h"

//-----
// OnFileReadOnly
//
// The OnFileReadOnly event is called when a read-only file needs to be
// installed or uninstalled.
//
// szFile will contain the full path of the file that is read-only when the
// event is called.
//
// The event should return one of the following values:
//
// ERR_YES - Indicates that the file should be installed or uninstalled,
//
// ERR_NO - Indicates that the file should not be installed or uninstalled.
//-----
function OnFileReadOnly(szFile)
begin
    // TODO: Enable this code if you want to return ERR_YES and simply
    // install or uninstall the read-only file w/o confirmation.
    // return ERR_YES;

    return SdExceptions(READONLY, szFile);
end;

```

## SdFeatureDialog



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdFeatureDialog** function creates a dialog that displays a list of features in the setup that the user can install and the amount of disk space that each feature occupies. This function is identical to SdFeatureDialogAdv.



The end user can change the destination folder by clicking the Browse button and can check the available disk space on other drives by clicking the Disk Space button.

## Syntax

```
SdFeatureDialog ( szTitle, szMsg, svDir, szFeatures );
```

## Parameters

**Table 27 • SdFeatureDialog Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Select Features"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message to display in the dialog. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>svDir</b>	<p>Specifies the name of the folder to be selected by default and returns the name of the folder selected by the end user. The destination folder specified by svDir is not assigned automatically to INSTALLDIR, TARGETDIR, or any other system variable. To apply the value of svDir to the installation, you must assign it to INSTALLDIR (in an InstallScript MSI installation), TARGETDIR (in an InstallScript installation), or to a script-defined variable, if one is in use.</p> <p>If the default folder specified by svDir does not already exist on the end user's system, it is not created unless the end user clicks the Browse button and follows the steps to create it from the Choose Folder dialog. Therefore, whenever you specify a default folder, you must call <a href="#">ExistsDir</a> when FeatureDialog returns in order to determine whether that folder exists. If it does not exist, call <a href="#">CreateDir</a> to create it on the end user's system.</p>
<b>szFeatures</b>	<p>Specifies the name of the feature whose subfeatures are to be displayed. To display all top-level features, pass a null string ("") in this parameter. To learn how to refer to top-level features and subfeatures, see <a href="#">Specifying Features and Subfeatures in Function Calls</a>.</p> <p>SdFeatureDialog searches for the requested features in the script-created feature set specified by the system variable <a href="#">MEDIA</a>; see the Additional Information section.</p>

## Return Values

**Table 28 • SdFeatureDialog Return Values**

Return Value	Description
<b>NEXT (1)</b>	Indicates that the end user clicked the Next button.
<b>BACK (12)</b>	Indicates that the end user clicked the Back button.

## Additional Information

- To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.
- A feature's size is displayed as 0 until it is selected. Once it has been selected, its actual size is displayed.

- The required disk space that is displayed by the dialog includes the size of the files that must be installed to enable maintenance and uninstallation. If all application components are deselected, the size of these files is still displayed.
- If your installation does not use a setup type dialog, you must call **FeatureSetupTypeSet** to specify a setup type that has been defined in the Setup Types view before calling **SdFeatureDialog**.
- **SdFeatureDialog** operates on the current media, which is specified by the system variable MEDIA. During setup initialization, the installation assigns to MEDIA a media name that is associated with your file media library (**Data1.cab**).



### Task

#### To display script-created features:

1. Save the current value of MEDIA in a string variable, for example, szSaveMEDIAValue.
  2. Assign to MEDIA the name of the script-created component set.
  3. Call **SdFeatureDialog** to get end-user selections.
  4. Assign to MEDIA the value that you saved in step 1. You must do this before calling **FeatureTransferData**.
- If necessary, feature names are truncated to allow the display of the largest possible feature size. The space required to display the size depends on the maximum feature size (2 GB), the feature size options currently in use, and the font used to display feature information in the dialog. Feature size options are set with the **DialogSetInfo** function.

When the space required to display the maximum possible size has been determined, all feature names are truncated automatically, if necessary, to fit the remaining space. The name of a feature that requires less space to display its size (or that is not selected) may still be truncated under this method. To maximize performance and ensure that feature names appear complete, make feature names or display names smaller than the space available in the dialog.

- The Available Disk Space dialog cannot be displayed with a skin; it appears the same regardless of whether you have specified a skin.
- The Disk Space button has an ID of 101. This button automatically displays the available disk space dialog. You can remove this button/option if you prefer. The Directory static field requires an ID of 851. The list box ID has a multiple selection style.
- In installations that were created with early versions of InstallShield Professional, when the end user selected in the Choose Folder dialog a folder that did not exist, a confirmation message box was displayed asking whether the folder should be created. This message box proved to be confusing to many end users, so it has been removed from InstallShield.

## SdFeatureDialog Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdFeatureDialog function.
 *
 * This example script displays a dialog that displays a list
 * of features in the setup that the end user can install and the
 * amount of space that each feature occupies.
 *
 * Comments: To run this example script, create a project (or
 *           insert into a project) with several features
 *           and/or subfeatures with components containing
 *           files.
 *
\*-----*/

#include "Ifx.h"

function OnBegin()
    STRING szTitle, szMsg, svDir;
begin
    svDir    = TARGETDIR;
    szTitle = "Select Features";
    szMsg    = "Select the features you want to install on your computer.";

    // Display all top-level features available.
    SdFeatureDialog (szTitle, szMsg, svDir, "");

end;

```

## SdFeatureDialog2



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdFeatureDialog2** function creates a dialog that displays the following:

- A list of features that the end user can install.
- The space required for the selected features and the space that is available in the destination location. A feature's size is displayed as the number 0 until it is selected. When the feature is selected, its actual size is displayed.
- A description of the selected feature—the value of the feature's Description setting.

If a particular feature has subfeatures, the Change button becomes active when the end user clicks the feature. Clicking the Change button launches the Select Subfeatures dialog, where the end user can make further selections.

### Syntax

```
SdFeatureDialog2 ( szTitle, szMsg, szDir, szFeatures );
```

## Parameters

**Table 29 • SdFeatureDialog2 Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title (“Select Features”), pass a null string (“”) in this parameter.
<b>szMsg</b>	Specifies the message to display in the dialog. To display the default instructions for this dialog, pass a null string (“”) in this parameter.
<b>szDir</b>	<p>Specifies the name of the folder to be selected by default and returns the name of the folder selected by the end user. The destination folder specified by szDir is not assigned automatically to INSTALLDIR, TARGETDIR, or any other system variable. To apply the value of szDir to the installation, you must assign it to INSTALLDIR (in an InstallScript MSI installation), TARGETDIR (in an InstallScript installation), or to a script-defined variable, if one is in use.</p> <p>If the default folder specified by szDir does not already exist on the end user’s system, it is not created unless the end user clicks the Browse button and follows the steps to create it from the Choose Folder dialog. Therefore, whenever you specify a default folder, you must call <a href="#">ExistsDir</a> when <b>FeatureDialog</b> returns in order to determine whether that folder exists. If it does not exist, call <a href="#">CreateDir</a> to create it on the end user’s system.</p>
<b>szFeatures</b>	<p>Specifies the name of the feature whose subfeatures are displayed. To display all top-level features, pass a null string (“”) in this parameter. To learn how to refer to top-level features and subfeatures, see <a href="#">Specifying Features and Subfeatures in Function Calls</a>.</p> <p>SdFeatureDialog2 searches for the requested features in the script-created feature set specified by the system variable <a href="#">MEDIA</a>; see the <a href="#">Additional Information</a> section.</p>

## Return Values

**Table 30 • SdFeatureDialog2 Return Values**

Return Value	Description
<b>NEXT (1)</b>	Indicates that the end user clicked the Next button.
<b>BACK (12)</b>	Indicates that the end user clicked the Back button.

## Additional Information

- To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.
- Default selection settings are cleared when the end user selects a feature or subfeature displayed in the dialog. If the end user clears a feature selection, all of its subfeatures are cleared. Conversely, if the end user clears all of a feature’s subfeature selections, the feature’s selection is cleared.

When a feature is not selected by default, its subfeatures are not selected by default. If all subfeatures of a feature are not selected by default, the parent feature should not be selected by default. For information on default feature and subfeature selection settings, see [FeatureAddItem](#).

- A feature's size is displayed as the number 0 until it is selected. Once it has been selected, its actual size is displayed.
- The required disk space displayed by the dialog includes the size of the files that must be installed to enable maintenance setups and uninstallation. Even if all application components are deselected, the size of these files is still displayed.
- If your installation does not use a setup type dialog, you must call [FeatureSetupTypeSet](#) to specify a setup type that has been defined in the Setup Types view before calling **SdFeatureDialog2**.
- **SdFeatureDialog2** operates on the current media, which is specified by the system variable MEDIA. During setup initialization, the installation assigns to MEDIA a media name that is associated with your file media library (**Data1.cab**).



#### Task

#### To display script-created features:

1. Save the current value of MEDIA in a string variable, for example, szSaveMEDIAValue.
  2. Assign to MEDIA the name of the script-created component set.
  3. Call **SdFeatureDialog2** to get end-user selections.
  4. Assign to MEDIA the value that you saved in step 1. You must do this before calling [FeatureTransferData](#).
- If necessary, feature names are truncated to allow the display of the largest possible feature size. The space required to display the size depends on the maximum feature size (2 GB), the feature size options currently in use, and the font used to display feature information in the dialog. Feature size options are set with the [DialogSetInfo](#) function.

When the space required to display the maximum possible size has been determined, all feature names are truncated automatically, if necessary, to fit the remaining space. The name of a feature that requires less space to display its size (or that is not selected) may still be truncated under this method. To maximize performance and ensure that feature names appear complete, make feature names or display names smaller than the space available in the dialog.

- The Select Subfeatures dialog cannot be displayed with a skin; it appears the same regardless of whether you have specified a skin.

## SdFeatureDialog2 Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
 *
 * InstallShield Example Script
```

```

*
* Demonstrates the SdFeatureDialog2 function.
*
* This example script displays a dialog that displays a list
* of features in the setup that the end user can install and the
* amount of space that each feature occupies.
*
* Comments: To run this example script, create a project (or
*           insert into a project) with several features
*           and/or subfeatures with components containing
*           files.
*
\*-----*/

#include "Ifx.h"

function OnBegin()
    STRING szTitle, szMsg, svDir;
begin
    svDir   = TARGETDIR;
    szTitle = "Select Features";
    szMsg   = "Select the features you want to install on your computer.";

    // Display all top-level features available.
    SdFeatureDialog2 (szTitle, szMsg, svDir, "");

end;

```

## SdFeatureDialogAdv



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdFeatureDialogAdv** function creates a dialog that displays a list of features in the setup that the end user can install and the amount of disk space that each feature occupies. This function is identical to SdFeatureDialog.

The end user can change the destination folder by clicking the Browse button and can check the available disk space on other drives by clicking the Disk Space button.

### Syntax

```
SdFeatureDialogAdv ( szTitle, szMsg, svDir, szFeatures );
```

## Parameters

**Table 31** • SdFeatureDialogAdv Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Select Features"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message to display in the dialog. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>svDir</b>	<p>Specifies the name of the folder to be selected by default and returns the name of the folder selected by the end user. The destination folder specified by svDir is not assigned automatically to INSTALLDIR, TARGETDIR, or any other system variable. To apply the value of svDir to the setup, you must assign it to INSTALLDIR (in an InstallScript MSI installation), TARGETDIR (in an InstallScript installation), or to a script-defined variable, if one is in use.</p> <p>If the default folder specified by svDir does not already exist on the end user's system, it is not created unless the end user clicks the Browse button and follows the steps to create it from the Choose Folder dialog. Therefore, whenever you specify a default folder, you must call <a href="#">ExistsDir</a> when FeatureDialog returns in order to determine whether that folder exists. If it does not exist, call <a href="#">CreateDir</a> to create it on the end user's system.</p>
<b>szFeatures</b>	<p>Specifies the name of the feature whose subfeatures are to be displayed. To display all top-level features, pass a null string ("") in this parameter. To learn how to refer to top-level features and subfeatures, see <a href="#">Specifying Features and Subfeatures in Function Calls</a>.</p> <p>SdCFeatureDialogAdv searches for the requested features in the script-created feature set specified by the system variable <a href="#">MEDIA</a>; see the <a href="#">Additional Information</a> section.</p>

## Return Values

**Table 32** • SdFeatureDialogAdv Return Values

Return Value	Description
<b>NEXT (1)</b>	Indicates that the end user clicked the Next button.
<b>BACK (12)</b>	Indicates that the end user clicked the Back button.

## Additional Information

- To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.
- A feature's size is displayed as 0 until it is selected. Once it has been selected, its actual size is displayed.



- The required disk space displayed by the dialog includes the size of the files that must be installed to enable maintenance setups and uninstallation. Even if all application components are deselected, the size of these files is still displayed.
- If your setup does not use a setup type dialog, you *must* call `FeatureSetupTypeSet` to specify a setup type that has been defined in the IDE's Setup Types view before calling `SdFeatureDialogAdv`.
- `SdFeatureDialogAdv` operates on the current media, which is specified by the system variable `MEDIA`. During setup initialization, the installation assigns to `MEDIA` a media name that is associated with your file media library (**Data1.cab**).



### Task

#### To display script-created features:

1. Save the current value of `MEDIA` in a string variable, for example, `szSaveMEDIAValue`.
  2. Assign to `MEDIA` the name of the script-created component set.
  3. Call `SdFeatureDialogAdv` to get end-user selections.
  4. Assign to `MEDIA` the value that you saved in step 1. You must do this before calling `FeatureTransferData`.
- If necessary, feature names are truncated to allow the display of the largest possible feature size. The space required to display the size depends on the maximum feature size (2 GB), the feature size options currently in use, and the font used to display feature information in the dialog. Feature size options are set with the `DialogSetInfo` function.

When the space required to display the maximum possible size has been determined, all feature names are truncated automatically, if necessary, to fit the remaining space. The name of a feature that requires less space to display its size (or that is not selected) may still be truncated under this method. To maximize performance and ensure that feature names appear complete, make feature names or display names smaller than the space available in the dialog.

- The Available Disk Space dialog cannot be displayed with a skin; it appears the same regardless of whether you have specified a skin.
- The Disk Space... button has an ID of 101. This button automatically displays the available disk space dialog. You can remove this button/option if you prefer. The Directory static field requires an ID of 851. The list box ID has a multiple selection style.
- In earlier versions of InstallShield Professional, when the end user selected in the Choose Folder dialog a folder that did not exist, a confirmation message box was displayed asking whether the folder should be created. This message box proved to be confusing to many end users, so it has been removed from InstallShield.

## SdFeatureDialogAdv Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
*
```

```

* InstallShield Example Script
*
* Demonstrates the SdFeatureDialogAdv function.
*
* This example script displays a dialog that displays a list
* of features in the setup that the end user can install and the
* amount of space that each feature occupies.
*
* Comments: To run this example script, create a project (or
*           insert into a project) with several features
*           and/or subfeatures with components containing
*           files.
*
\*-----*/

#include "Ifx.h"

function OnBegin()
    STRING szTitle, szMsg, svDir;
begin

    svDir    = TARGETDIR;
    szTitle = "Select Features";
    szMsg    = "Select the features you want to install on your computer.";

    // Display all top-level features available.
    SdFeatureDialogAdv (szTitle, szMsg, svDir, "");

end;

```

## SdFeatureMult



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdFeatureMult** function creates a dialog that displays the following:

- A list of features and subfeatures that the end user can select for installation. The dialog has two feature lists. If the feature selected in the first list has subfeatures, the subfeatures are displayed in the second list.
- The disk space required for the selected features and the space available in the destination location. A feature's size is displayed as 0 until it is selected. When the end user selects a feature, its actual size is displayed.
- The description of a feature or subfeature. The end user can view the description by clicking on the feature or subfeature.

### Syntax

```
SdFeatureMult ( szTitle, szMsg, svDir, szFeatures );
```

## Parameters

**Table 33 • SdFeatureMult Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Select Features"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message to display in the dialog. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>svDir</b>	<p>Specifies the name of the folder to be selected by default and returns the name of the folder selected by the end user. The destination folder specified by svDir is not assigned automatically to INSTALLDIR, TARGETDIR, or any other system variable. To apply the value of svDir to the setup, you must assign it to INSTALLDIR (in an InstallScript MSI installation), TARGETDIR (in an InstallScript installation), or to a script-defined variable, if one is in use.</p> <p>If the default folder specified by svDir does not already exist on the end user's system, it is not created unless the end user clicks the Browse button and follows the steps to create it from the Choose Folder dialog. Therefore, whenever you specify a default folder, you must call <a href="#">ExistsDir</a> when FeatureDialog returns in order to determine whether that folder exists. If it does not exist, call <a href="#">CreateDir</a> to create it on the end user's system.</p>
<b>szFeatures</b>	<p>Specifies the name of the feature whose subfeatures are displayed. To display all top-level features, pass a null string ("") in this parameter. To learn how to refer to top-level features and subfeatures, see <a href="#">Specifying Features and Subfeatures in Function Calls</a>.</p> <p>SdFeatureMult searches for the requested features in the script-created feature set specified by the system variable <a href="#">MEDIA</a>; see the <a href="#">Additional Information</a> section.</p>

## Return Values

**Table 34 • SdFeatureMult Return Values**

Return Value	Description
<b>NEXT (1)</b>	Indicates that the end user clicked the Next button.
<b>BACK (12)</b>	Indicates that the end user clicked the Back button.

## Additional Information

- To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

- Default selection settings are cleared when the end user selects a feature or subfeature displayed in the dialog. If the end user clears a feature selection, all of its subfeatures are cleared. Conversely, if the end user clears all of a feature's subfeature selections, the feature's selection is cleared.

When a feature is not selected by default, its subfeatures are not selected by default. If all subfeatures of a feature are not selected by default, the parent feature should not be selected by default. See [FeatureAddItem](#) for information on default feature and subfeature selection settings.

- A feature's size is displayed as 0 until it is selected. Once it has been selected, its actual size is displayed.
- The required disk space displayed by the dialog includes the size of the files that must be installed to enable maintenance setups and uninstallation. Even if all application components are deselected, the size of these files is still displayed.
- If your setup does not use a setup type dialog, you *must* call `FeatureSetupTypeSet` to specify a setup type that has been defined in the IDE's Setup Types view before calling `SdFeatureMult`.
- `SdFeatureMult` operates on the current media, which is specified by the system variable `MEDIA`. During setup initialization, the installation assigns to `MEDIA` a media name that is associated with your file media library (**Data1.cab**).



#### Task

#### To display script-created features:

1. Save the current value of `MEDIA` in a string variable, for example, `szSaveMEDIAValue`.
  2. Assign to `MEDIA` the name of the script-created component set.
  3. Call `SdFeatureMult` to get end-user selections.
  4. Assign to `MEDIA` the value that you saved in step 1. You must do this before calling `FeatureTransferData`.
- If necessary, feature names are truncated to allow the display of the largest possible feature size. The space required to display the size depends on the maximum feature size (2 GB), the feature size options currently in use, and the font used to display feature information in the dialog. Feature size options are set with the `DialogSetInfo` function.

When the space required to display the maximum possible size has been determined, all feature names are truncated automatically, if necessary, to fit the remaining space. The name of a feature that requires less space to display its size (or that is not selected) may still be truncated under this method. To maximize performance and ensure that feature names appear complete, make feature names or display names smaller than the space available in the dialog.

- The Select Subfeatures dialog cannot be displayed with a skin; it appears the same regardless of whether you have specified a skin.

## SdFeatureMult Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdFeatureMult function.
 *
 * This example script displays a dialog that lets the user
 * select features and subfeatures, view feature descriptions,
 * and see the amount of space required for the selected features
 * and the space available on the destination folder.
 *
 * Comments: To run this example script, create a project (or
 *           insert into a project) with several features
 *           and/or subfeatures with components containing
 *           files.
 *
\*-----*/

#include "Ifx.h"

function OnBegin()
    STRING szTitle, szMsg, svDir;
begin

    svDir  = TARGETDIR;
    szTitle = "Select Features";
    szMsg  = "Select the features you want to install on your computer.";

    // Display all top-level features available.
    SdFeatureMult (szTitle, szMsg, svDir, "");

end;

```

## SdFeatureTree



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdFeatureTree** function displays a dialog that contains the following:

- A tree control in which end users can select the features they want on their system and clear the features they do not want on their system.
- A description of the selected feature—the text in the feature's Description property.
- The drive space required to perform the file operations selected in the tree control, and the space available on the drive of the path specified by szDir. The calculation of required drive space takes into account the cluster size on the szDir drive.

### Syntax

```
SdFeatureTree ( szTitle, szMsg, szDir, szFeatures, nLevel );
```

## Parameters

**Table 35 • SdFeatureTree Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Select Features"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message to display in the dialog. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>szDir</b>	Specifies the path that is used in calculating required and available drive space. The script variable TARGETDIR should always be specified for this parameter.
<b>szFeatures</b>	Specifies the name of the feature whose subfeatures are displayed. To display all top-level features, pass a null string ("") in this parameter. To learn how to refer to top-level features and subfeatures, see Specifying Features and Subfeatures in Function Calls.
<b>nLevel</b>	Specifies how many levels of features and subfeatures are open in the tree control when the dialog is first displayed. (For example, an nLevel of 2 causes third- and lower-level subfeatures to be closed in the tree control when the dialog is first displayed.)

## Return Values

**Table 36 • SdFeatureTree Return Values**

Return Value	Description
<b>NEXT (1)</b>	Indicates that the end user clicked the Next button.
<b>BACK (12)</b>	Indicates that the end user clicked the Back button.

## Additional Information

- To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.
- The required disk space displayed by the dialog includes the size of the files that must be installed to enable maintenance installations and uninstallation. Even if all application features are deselected, the size of these files is still displayed.
- **SdFeatureTree** operates on the current media, which is specified by the system variable MEDIA. During setup initialization, the installation assigns to MEDIA a media name that is associated with your file media library (**Data1.cab**).

**Task****To display script-created features:**

1. Save the current value of MEDIA in a string variable, for example, szSaveMEDIAValue.
2. Assign to MEDIA the name of the script-created component set.
3. Call **SdFeatureTree** to get end-user selections.
4. Assign to MEDIA the value that you saved in step 1. You must do this before calling **FeatureTransferData**.

## SdFeatureTree Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdFeatureTree function.
 *
 * This example script displays a dialog that lets the user
 * select features and subfeatures, view feature descriptions,
 * and see the amount of space required for the selected features
 * and the space available on the destination folder.
 *
 * Comments: To run this example script, create a project (or
 *           insert into a project) with several features
 *           and/or subfeatures with components containing
 *           files.
 *
 \*-----*/

#include "Ifx.h"

function OnBegin()
    STRING szTitle, szMsg, svDir;
begin

    svDir    = TARGETDIR;
    szTitle  = "Select Features";
    szMsg     = "Select the features you want to install on your computer.";

    // Display all top-level features available, with third- and
    // lower-level subfeatures to be closed in the tree control
    // when the dialog is first displayed.
    SdFeatureTree (szTitle, szMsg, svDir, "", 2);

end;
```

# SdFilesInUse



**Project** • The InstallScript MSI project type support the **SdFilesInUse** function.

The **SdFilesInUse** dialog can also be called manually in InstallScript projects, in InstallScript MSI projects, and Basic MSI projects that have InstallScript custom actions; however, the installation must provide the list of applications that are locking files through the *nvlistApps* parameter, and it must handle the return value appropriately.

The **SdFilesInUse** function displays a dialog that includes a list box containing a list of the applications that are open and are locking files.

Typically, the OnFilesInUse event handler displays this dialog in an InstallScript MSI installation as a response to an INSTALLMESSAGE\_FILESINUSE message sent by the Windows Installer. When this occurs, the Windows Installer provides to the installation the list of applications that are locking files. The list of applications are passed through the *szMessage* parameter to the OnFilesInUse event. The event passes this information to the **SdFilesInUse** function through the *szMessage* parameter. The event then passes the return value from the function as the return value of the event, which causes the Windows Installer to act appropriately.


## Syntax

```
SdFilesInUse ( byval string szTitle, byval string szMsg, byval string szFilesInUse, byref LIST  
              nvlistApps );
```



## Parameters

**Table 37 • SdFilesInUse Parameters**


Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title (Files in Use), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message to display in the dialog. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>szFilesInUse</b>	<p>When <b>SdFilesInUse</b> is called by the OnFilesInUse event handler in an InstallScript MSI installation, this parameter specifies the list of applications that the Windows Installer found to be locking files. The function parses this string in order to display the list of applications in the dialog's list box.</p> <p>If you manually call <b>SdFilesInUse</b>, pass an empty string ("") in this parameter. Note that if nvlistApps is a valid string list, this parameter is ignored.</p>
<b>nvlistApps</b>	<p>When <b>SdFilesInUse</b> is called by the OnFilesInUse event handler in an InstallScript MSI installation, this parameter is specified as an uninitialized list variable—that is, a variable with a value of 0.</p> <p>If you manually call <b>SdFilesInUse</b>, specify a string list of applications that are locking files and that should be displayed in the dialog. Each string in the list indicates one application. (Use the <b>ListCreate</b> function and associated list functions to create and initialize the string list.)</p>
 <p><b>Note •</b> You must provide a variable for this parameter; you cannot specify a literal value.</p>	

## Return Values

**Table 38 • SdFilesInUse Return Values**

Return Value	Description
<b>IDRETRY (4)</b>	<p>Indicates that the end user clicked the Retry button.</p> <p>If you manually call <b>SdFilesInUse</b>, you should handle this return value by rechecking locked files and redisplaying the dialog as needed. When <b>SdFilesInUse</b> is called by the OnFilesInUse event handler, this value is returned by the event handler, and this is handled by the Windows Installer.</p>
<b>IDIGNORE (5)</b>	Indicates that the end user clicked the Ignore button.

**Table 38 • SdFilesInUse Return Values (cont.)**

Return Value	Description
<b>IDCANCEL</b>	Indicates that the end user clicked the Exit button.
	 <p><b>Note</b> • Note that unlike other script dialogs, this dialog does not call the <i>OnCanceling</i> event handler when the user clicks the Exit button. Therefore, if you are calling this function manually, you must handle this return value manually in order to display the Cancel Setup dialog.</p>

## SdFilesInUse Example



**Project** • The InstallScript MSI project type support the **SdFilesInUse** function.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdFilesInUse function. SdFilesInUse displays
 * a list box dialog that lists applications that are locking
 * files.
 *
 * Keep displaying SdFilesInUse if Notepad.exe is running
 * and user clicks Retry.
 *
 \*-----*/

#include "ifx.h"

function OnFirstUIBefore( )
    // ...other variables...
    LIST listID;
    NUMBER nReturn;
begin

    // ...display other dialogs...

    // display file-locked dialog only if Notepad.exe running
    if (Is(FILE_LOCKED, WINDIR ^ "Notepad.exe")) then
        // create a string list
        listID = ListCreate(STRINGLIST);

        // if an error occurred, report it; then terminate.
        if (listID = LIST_NULL) then
            MessageBox("Unable to create list.", SEVERE);
            abort;
        endif;

        ListAddString(listID, "Notepad.exe", AFTER);
    
```

```

AskAgain:

// display the Files in Use dialog.
nReturn = SdFilesInUse("", "The following applications are currently locking files.", "", listID);

// if user clicks RETRY, show locked-file dialog again;
// otherwise, let the file-transfer process handle it, which might require a reboot
if (nReturn = IDRETRY) then
    if (Is(FILE_LOCKED, WINDIR ^ "Notepad.exe")) then
        goto AskAgain;
    endif;
endif;

// remove the list from memory
ListDestroy(listID);
endif;

end;

```

## SdFinish



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdFinish** function displays a dialog to inform the end user that the installation is complete and to provide information or options. The SdFinish dialog displays up to two messages and two check box selection options. For example, you could offer the end user the option of either viewing a README file or launching the application.

To insert the product name into the messages and check box descriptions, use the place holder %P in the strings passed in szMsg1, szMsg2, szOpt1, and szOpt2.



**Note** • *SdFinish* has no option to terminate the setup and reboot the end user's computer. When *SdFinish* returns, the setup continues to execute. To provide the end user with the option to reboot, call *SdFinishReboot* instead.

### Syntax

```
SdFinish ( szTitle, szMsg1, szMsg2, szOpt1, szOpt2, bvOpt1, bvOpt2 );
```

## Parameters

**Table 39** • SdFinish Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Setup Complete"), pass a null string ("") in this parameter.
<b>szMsg1</b>	Specifies the message to display at the top of the dialog. To display the default instructions, which inform the user that the installation is complete, pass a null string ("") in this parameter.
<b>szMsg2</b>	Specifies the message to display at the bottom of the dialog. To display the default instructions ("Click Finish to complete Setup"), pass a null string ("") in this parameter.
<b>szOpt1</b>	Specifies the text to display beside the first check box. Pass a null string ("") in this parameter to hide the check box.
<b>szOpt2</b>	Specifies the text to display beside the second check box. Pass a null string ("") in this parameter to hide the check box.
<b>bvOpt1</b>	Returns the selection state (TRUE or FALSE) of the first check box.
<b>bvOpt2</b>	Returns the selection state (TRUE or FALSE) of the second check box.

## Return Values

**Table 40** • SdFinish Return Values

Return Value	Description
<b>NEXT (1)</b>	Indicates that the Finish button was clicked.



**Note** • Because SdFinish announces the end of the installation, the Back button is disabled.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdFinish Example



**Project** • This information applies to the following project types:

- InstallScript

- *InstallScript MSI*

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the function SdFinish.
 *
 * Note: Before running this script, set the preprocessor
 *       constants so that they reference the fully-qualified
 *       names of the Windows Notepad executable and a valid
 *       text file in the Support Files/Billboards view.
 *
\*-----*/

//Assign the name of a text file to READMEFILE
#define NOTEPAD WINDIR ^ "Notepad.exe"
#define READMEFILE SUPPORTDIR ^ "ReadMe.txt"

#include "Ifx.h"

function OnBegin()
    STRING    szProductName, szTitle;
    STRING    szMsg1, szMsg2, szOpt1, szOpt2;
    BOOL      bvOpt1, bvOpt2;
    NUMBER    nReturn;
begin

    // Set the product name to substitute for the %P place holder.
    szProductName = "My Application";
    SdProductName (szProductName);

    // Setup parameters that will be passed to SdFinish.
    szTitle = "SdFinish Example";
    szMsg1 = "%P Setup is almost complete.\n" +
            "Choose the options you want below.";
    szMsg2 = "Click Finish to complete %P Setup.";
    szOpt1 = "I would like to view the README file.";
    szOpt2 = "I would like to launch %P.";

    // Display the SdFinish dialog.
    SdFinish (szTitle, szMsg1, szMsg2, szOpt1, szOpt2, bvOpt1, bvOpt2);

    if (bvOpt1) then
        // Display the read me file.
        LaunchAppAndWait (NOTEPAD, READMEFILE, WAIT);
    endif;

    if (bvOpt2) then
        // Because this example does not actually install an
        // application, a message box is displayed in place of
        // the call to LaunchApp that would normally be made here,
        // for example:
        // LaunchApp (TARGETDIR ^ "MyApp.exe", "");

        SprintfBox (INFORMATION, szTitle, "Launch %s here.", szProductName);
    endif;

end;

```

# SdFinishEx



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdFinishEx** function calls either SdFinish or SdFinishReboot to display a dialog that informs the end user that the installation is complete and provides information or options. If the system variable BATCH\_INSTALL is equal to FALSE (indicating that no locked files were encountered during the setup), SdFinishEx calls SdFinish to display the dialog. If BATCH\_INSTALL is equal to a non-zero value, SdFinishEx calls SdFinishReboot to display the dialog.

To insert the product name into the messages and check box descriptions, use the place holder %P in the strings passed in szMsg1, szMsg2, szOpt1, and szOpt2.

## Syntax

```
SdFinishEx (szTitle, szMsg1, szMsg2, szOpt1, szOpt2, bvOpt1, bvOpt2);
```

## Parameters

The parameters are the same as those of SdFinish. If BATCH\_INSTALL is equal to TRUE, these parameters are ignored and SdFinishReboot("", "", SYS\_BOOTMACHINE, "", 0) is called.

## Return Values

**Table 41** • SdFinishEx Return Values

Return Value	Description
<b>0</b>	Indicates that SdFinish was called.
<b>NEXT (1)</b>	Indicates that SdFinishReboot was called and the user chose not to reboot the computer.
<b>&lt; 0</b>	Indicates that SdFinishReboot was called and the user chose to reboot the computer, but the reboot failed.

## Additional Information

To display non-default text in the SdFinishReboot dialog, do not call SdFinishEx. Instead, use code like the following:

```
if (!BATCH_INSTALL) then
    SdFinish( szTitle, szMsg1, szMsg2,
              szOption1, szOption2, bOpt1, bOpt2 );
else
    SdFinishReboot( szRebootTitle, szRebootMsg1,
                   SYS_BOOTMACHINE, szRebootMsg2, 0 );
endif;
```

## SdFinishEx Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the function SdFinishEx.
 *
 * The SdFinishEx function calls either SdFinish or SdFinishReboot
 * to display a dialog informing the end user that the
 * installation is complete and giving the user information or
 * options. If the system variable BATCH_INSTALL is equal to
 * FALSE (0), indicating that no locked files were encountered
 * during the setup, SdFinishEx calls SdFinish to display the
 * dialog. If BATCH_INSTALL is equal to a non-zero value,
 * SdFinishEx calls SdFinishReboot to display the dialog.
 *
 \*-----*/

#include "Ifx.h"

function OnBegin()
    STRING szTitle, szMsg1, szMsg2, szOption1, szOption2;
    NUMBER bOpt1, bOpt2;
begin

    bOpt1 = FALSE;
    bOpt2 = FALSE;
    szMsg1 = SdLoadString(IFX_SDFINISH_MSG1);
    SdFinishEx(szTitle, szMsg1, szMsg2, szOption1, szOption2, bOpt1, bOpt2);
end;
```

## SdFinishReboot



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdFinishReboot** function at the end of your installation announces that the installation is complete and gives the end user the option to restart the system. Restarting the system allows changes to **Autoexec.bat**, **Config.sys**, and some .ini files to take effect.

The **SdFinishReboot** dialog displays up to two messages in static text fields. Set the value of these fields with the parameters szMsg1 and szMsg2. To insert the product name into the messages, use the place holder %P in the strings passed in szMsg1 and szMsg2.

## Syntax

```
SdFinishReboot ( szTitle, szMsg1, nDefOption, szMsg2, nReserved );
```

## Parameters

**Table 42 • SdFinishReboot Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Setup Complete"), pass a null string ("") in this parameter.
<b>szMsg1</b>	Specifies the text to display at the top of the dialog, informing the user about the end of the installation. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>nDefOption</b>	Specifies a default radio button option selection. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>SYS_BOOTMACHINE</b>—Reboot the computer when the setup is finished.</li> <li>• <b>0</b>—Do not reboot the computer.</li> </ul>
<b>szMsg2</b>	Specifies the text to display at the bottom of the dialog, giving the user information about what to do. To display the default instructions, pass a null string ("").
<b>nReserved</b>	Pass zero in this parameter. No other value is allowed.

## Return Values

**Table 43 • SdFinishReboot Return Values**

Return Value	Description
<b>WILL_REBOOT</b>	Indicates that the user chose to reboot the system.
<b>NEXT (1)</b>	Indicates that the user chose not to reboot the system or restart Windows.
<b>&lt; 0</b>	Indicates that the user chose to either reboot the system or restart Windows, but the reboot or restart failed.



**Note** • Because **SdFinishReboot** announces the end of the installation, the Back button is disabled.

## Additional Information

- To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.
- The installation makes every attempt *not* to reboot the computer when other instances of the installation are running. Because of this, you must ensure that all other instances of the installation are shut down before



allowing **SdFinishReboot** to restart Windows or the system. Additionally, your message to the end user should request that if they plan to restart the system later, they should ensure that all other applications are shut down first.

- The installation automatically ensures that locked .dll and .exe files are updated the next time that the system starts.

## SdFinishReboot Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdFinishReboot dialog.
 *
 * Warning: If you select the reboot option, your computer will
 *          reboot. Be sure you do not have any unsaved files
 *          open when you select this option.
 *
 \*-----*/

#include "Ifx.h"

function OnBegin()
    STRING szTitle, szMsg1, szMsg2;
    NUMBER nDefOption, nReserved;
begin

    // Set up variables to pass as parameters to SdFinishReboot.
    szTitle   = "SdFinishReboot Example";
    szMsg1    = "Setup has completed installing %P.";

    // nDefOption - Specifies a default radio button option selection.
    // SYS_BOOTMACHINE - Reboot the computer when the setup is finished.
    // 0 - Do not reboot the computer.
    nDefOption = 0;
    szMsg2     = "Click Finish to exit %P setup.";
    nReserved = 0;

    // Display the SdFinishReboot dialog.
    if (SdFinishReboot (szTitle, szMsg1, nOption, szMsg2, nReserved) < 0) then
    // Indicates that the user chose to either reboot the system
    // or restart Windows, but the reboot or restart failed.
        MessageBox ("SdFinishReboot failed.", SEVERE);
    endif;

end;
```

# SdFinishUpdate



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The [SdFinishUpdateEx](#) function supersedes the **SdFinishUpdate** function. **SdFinishUpdate** calls the following:

```
SdFinishUpdateEx( szTitle, szMsg1, szMsg2, "", "", bDefOption );
```

The **SdFinishUpdate** function at the end of your installation displays a dialog that indicates that the installation is complete. The dialog includes the option to check for application updates.



**Note** • **SdFinishUpdate** does not check for updates; to check for updates, add FlexNet Connect API calls in your *InstallScript* code. For more information, see the *FlexNet Connect SDK documentation*.

## Syntax

```
SdFinishUpdate ( szTitle, szMsg1, szMsg2, bDefOption );
```

# SdFinishUpdateEx



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdFinishUpdateEx** function at the end of your installation displays a dialog that indicates that the installation is complete. The dialog includes the option to check for application updates.



**Note** • **SdFinishUpdateEx** does not check for updates; to check for updates, add FlexNet Connect API calls in your *InstallScript* code. For more information, see the *FlexNet Connect SDK documentation*.

## Syntax

```
SdFinishUpdateEx ( szTitle, szMsg1, szMsg2, szOpt1, szOpt2, bDefOption );
```

## Parameters

**Table 44 • SdFinishUpdateEx Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Setup Complete"), pass a null string ("") in this parameter.
<b>szMsg1</b>	Specifies the text to display at the top of the dialog, informing the user about the end of the installation. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>szMsg2</b>	Specifies the text to display at the bottom of the dialog, providing information to the end user about what to do next. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>szOpt1</b>	Specifies the text to display beside the first radio button. To display the default instructions ("Yes, check for program updates. (Recommended)\nPlease ensure that you're connected to the Internet before you proceed."), pass a null string ("") in this parameter.
<b>szOpt2</b>	Specifies the text to display beside the second radio button. To display the default instructions ("No, skip this step."), pass a null string ("") in this parameter.
<b>bDefOption</b>	Specifies the option button that is selected by default. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li><b>TRUE</b>—Specifies the first option button as the default selection.</li> <li><b>FALSE</b>—Specifies the second option button as the default selection.</li> </ul>

## Return Values

**Table 45 • SdFinishUpdateEx Return Values**

Return Value	Description
<b>TRUE</b>	Indicates that the <b>Check for program updates</b> option button is selected.
<b>FALSE</b>	Indicates that the <b>Skip this step</b> option is selected.



**Note** • Because *SdFinishUpdateEx* announces the end of the installation, the Back button is disabled.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

# SdFinishUpdateReboot



---

**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdFinishUpdateReboot** function at the end of your installation displays a dialog that indicates that the installation is complete. The dialog gives the end user the option to restart the system and to check for application updates. Restarting the system enables changes to **Autoexec.bat**, **Config.sys**, and some .ini files to take effect.



---

**Note** • **SdFinishUpdateReboot** does not check for updates; to check for updates, add FlexNet Connect API calls in your InstallScript code. For more information, see the FlexNet Connect SDK documentation.

## Syntax

```
SdFinishUpdateReboot ( szTitle, szMsg1, nDefOption, szMsg2, nChkUpdate, nReserved );
```

## Parameters

**Table 46 • SdFinishUpdateReboot Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Setup Complete"), pass a null string ("") in this parameter.
<b>szMsg1</b>	Specifies the text to display at the top of the dialog, informing the end user about the end of the installation. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>nDefOption</b>	Specifies a default radio button selection. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>SYS_BOOTMACHINE</b>—Reboot the computer when the setup is finished.</li> <li>• <b>0</b>—Do not reboot the computer.</li> </ul>
<b>szMsg2</b>	Specifies the text to display at the bottom of the dialog, giving the end user information about what to do. To display the default instructions, pass a null string ("").
<b>nvChkUpdate</b>	Returns the state of the FlexNet Connect check box: <ul style="list-style-type: none"> <li>• <b>0</b>—Check box is not selected. Do not check for application updates.</li> <li>• <b>Other than 0</b>—Check box is selected. Check for application updates.</li> </ul>
<b>nReserved</b>	Pass zero in this parameter. No other value is allowed.

## Return Values

**Table 47 • SdFinishUpdateReboot Return Values**

Return Value	Description
<b>WILL_REBOOT</b>	Indicates that the end user chose to reboot the system.
<b>NEXT (1)</b>	Indicates that the end user chose not to reboot the system or restart Windows.
<b>&lt; 0</b>	Indicates that the end user chose to either reboot the system or restart Windows, but the reboot or restart failed.



**Note** • Because *SdFinishUpdateReboot* announces the end of the installation, the Back button is disabled.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdFinishUpdateReboot Example



**Project** • This information applies to the following project types:

- InstallScript
- InstallScript MSI

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdFinishUpdateReboot dialog.
 *
 * This dialog is used at the end of an installation to announce
 * that the installation is complete and gives the user the option
 * to restart the system and to check for application updates.
 *
 \*-----*/

// Include Ifx.h for built-in InstallScript function prototypes
#include "ifx.h"

export prototype ExFn_SdFinishUpdateReboot(HWND);

function ExFn_SdFinishUpdateReboot(hMSI)
    STRING szTitle, szMsg1, szMsg2;
    NUMBER nDefOption, nChkUpdate, nReserved;
begin

    szTitle = "SdFinishUpdateReboot Example";
    szMsg1 = "";
    szMsg2 = "";

    // nDefOption - Specifies the option button selected by default.
    // 1 - Yes, check for program updates
    // 0 - No, skip this step
    nDefOption = 1;

    // nChkUpdate - Specifies the default state of the FlexNet Connect check box.
    // nChkUpdate - Returns the state of the FlexNet Connect check box.
    // 0 - Check box is not selected. Do not check for application updates.
    // Other than 0 - Check box is selected. Check for application updates.
    nChkUpdate = 1;
    nReserved = 0;
    SdFinishUpdateReboot ( szTitle, szMsg1, nDefOption, szMsg2, nChkUpdate, nReserved );

end;
```

## SdGeneralInit

The **SdGeneralInit** function provides standard dialog initialization, including setting the enable or disable state of the Next, Back, and Cancel buttons. This function also replaces all %P, %VS, and %VI instances with IFX\_PRODUCT\_DISPLAY\_NAME, IFX\_PRODUCT\_DISPLAY\_VERSION, and IFX\_INSTALLED\_DISPLAY\_VERSION on static controls with control IDs 700 through 724, and 202.

## Syntax

```
SdGeneralInit (szDialog, hwndDlg, nUnused, szUnused);
```

## Parameters

**Table 48 • SdGeneralInit Parameters**

Parameter	Description
<b>szDialog</b>	Name of the dialog created with <a href="#">EzDefineDialog</a> or <a href="#">DefineDialog</a> .
<b>hwndDlg</b>	Handle to the current dialog. This is obtained by calling <a href="#">CmdGetHwndDlg</a> .
<b>nUnused</b>	Unused parameter; pass 0.
<b>szUnused</b>	Unused parameter; pass an empty string ("").

## Return Values

None.

## SdGeneralInit Example

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdGeneralInit, which is used to initialize
 * a dialog that is created with the EzDefineDialog function.
 *
 * This script opens a simple custom dialog that displays
 * a bitmap. The dialog can be closed with any of three
 * buttons: Back, Next, or Cancel.
 *
 * The "custom" dialog used in this script is actually the
 * InstallShield Sd dialog that is displayed by the built-in
 * function SdBitmap. Because this dialog is stored in
 * the file _isres.dll, which is already compressed in
 * the installation, it can be used in a script as a custom
 * dialog.
 *
 * In order to use this custom dialog, the script first defines
 * it by calling EzDefineDialog. It then displays the dialog
 * and gets dialog events by calling WaitOnDialog. When an
 * event ends dialog processing, EndDialog is called to close
 * the dialog. Then the dialog is released from memory by
 * a call to ReleaseDialog.
 *
 *-----*/

// Dialog and control IDs.
#define RES_DIALOG_ID      12027  // ID of dialog itself
#define RES_PBUT_NEXT     1      // ID of Next button
#define RES_PBUT_CANCEL   9      // ID of Cancel button
```

```

#define RES_PBUT_BACK          12    // ID of Back button

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_EzDefineDialog(HWND);

function ExFn_EzDefineDialog(hMSI)
    STRING  szDialogName, szDLLName, szDialog;
    NUMBER  nDialog, nResult, nCmdValue;
    BOOL     bDone;
    HWND     hInstance, hwndParent, hwndDlg;
begin

    // Specify a name to identify the custom dialog in this installation.
    szDialogName = "CustomDialog";

    // Define the dialog. Pass a null string in the second parameter
    // to get the dialog from _isuser.dll or _isres.dll. Pass a null
    // string in the third parameter because the dialog is identified
    // by its ID in the fourth parameter.
    nResult = EzDefineDialog (szDialogName, "", "", RES_DIALOG_ID);

    if (nResult < 0) then
        // Report an error; then terminate.
        MessageBox ("Error in defining dialog", SEVERE);
        abort;
    endif;

    // Initialize the indicator used to control the while loop.
    bDone = FALSE;

    // Loop until done.
    repeat

        // Display the dialog and return the next dialog event.
        nCmdValue = WaitOnDialog(szDialogName);

        // Respond to the event.
        switch (nCmdValue)
            case DLG_CLOSE:
                // The user clicked the window's Close button.
                Do (EXIT);
            case DLG_ERR:
                MessageBox ("Unable to display dialog. Setup canceled.", SEVERE);
                abort;
            case DLG_INIT:
                // Initialize the back, next, and cancel button enable/disable states
                // for this dialog and replace %P, %VS, %VI with
                // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
                // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724 and 202.
                hwndDlg = CmdGetHwndDlg(szDialogName);
                SdGeneralInit(szDialogName, hwndDlg, 0, "");
            case RES_PBUT_CANCEL:
                // The user clicked the Cancel button.
                Do (EXIT);
            case RES_PBUT_NEXT:
                bDone = TRUE;
            case RES_PBUT_BACK:

```



```

        bDone = TRUE;
        if (SdIsStdButton( nCmdValue ) && SdDoStdButton( nCmdValue )) then
            bDone = TRUE;
        endif;
    endswitch;

until bDone;

// Close the dialog.
EndDialog (szDialogName);

// Free the dialog from memory.
ReleaseDialog (szDialogName);

end;

```

## SdInit

The **SdInit** function prepares an installation for Sd dialog function calls by loading required resource strings, restoring the installation's window if it is minimized, and specifying Windows 95-style check boxes in Sd dialogs.



**Note** • This function is called automatically by each Sd function. It is unnecessary to call SdInit explicitly unless your script calls DialogSetInfo before calling any of the Sd dialog functions. In that case, your script must call SdInit before it calls DialogSetInfo—otherwise the call to DialogSetInfo has no effect.

### Syntax

```
SdInit ( );
```

### Parameters

This function takes no parameters.

### Return Values

**Table 49** • SdInit Return Values

Return Value	Description
0	Indicates that the setup was initialized for Sd dialog function calls.
1	Indicates that the setup has already been initialized for Sd dialog function calls.

## SdInit Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdInit function.
 *

```

```

* Note that SdInit is called automatically by each Sd function.
* It is unnecessary to call SdInit explicitly unless your script
* calls DialogSetInfo before calling any of the Sd dialog functions.
* In that case, your script must call SdInit before it calls
* DialogSetInfo—otherwise the call to DialogSetInfo has no effect.
*
\*-----*/

#include "Ifx.h"

function OnBegin()
    STRING szInfoString;
begin

    // Initialize setup for calls to Sd dialog functions.
    SdInit ();

    // Set check box style for feature selection.
    DialogSetInfo ( DLG_INFO_CHECKSELECTION, szInfoString, CHECKBOX );

    // Get feature selections.
    SdFeatureDialog2 ( "", "", TARGETDIR, "" );

end;

```

## SdLicense



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The [SdLicenseEx](#) function supersedes the **SdLicense** function.

The **SdLicense** function displays a dialog that contains a license agreement in a multi-line edit field. The license agreement is stored in a text file identified in the parameter `szLicenseFile`.


The end user can scroll up and down to read the agreement. The end user must choose either the Yes, No, or—if enabled—Back button. Because this is usually the first dialog that you would display, you might want to disable the Back button. If the user selects Yes, the installation continues. If the user selects No, the installation displays the ExitSetup dialog.

### Syntax

```
SdLicense ( szTitle, szMsg, szQuestion, szLicenseFile );
```

## Parameters

**Table 50 • SdLicense Parameters**

Parameter	Description
<b>szTitle</b>	Specify the title of the dialog. To display the default title ("Software License Agreement"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specify the message to display in the static text field above the multi-line edit field. To display the default instructions, pass a null string ("") in this parameter.
<b>szQuestion</b>	Specify the text to display in the static text field below the multi-line edit field. You would likely place a question here, to which the user should respond by selecting either Yes and No. To display the default instructions, pass a null string ("") in this parameter.
<b>szLicenseFile</b>	Specify the name of the ANSI text file that contains the license agreement. This file must be added to the appropriate language folder in the Support Files/Billboards view. You can also specify szLicenseFile by entering the fully qualified name, in quotation marks, or a UNC path.
	 <p><b>Note</b> • This file must be a text file (.txt).</p>

## Return Values

**Table 51 • SdLicense Return Values**

Return Value	Description
<b>YES (1)</b>	The end user selected the Yes button.
<b>BACK (12)</b>	Indicates that the user selected the Back button.



**Note** • This function cannot return NO because if the end user clicks the No button, the ExitSetup dialog is displayed.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdLicense Example



**Project** • This information applies to the following project types:

- InstallScript

- *InstallScript MSI*

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdLicense function.
 *
 * This script calls SdLicense to display a license agreement
 * and prompt the user to accept or reject that agreement.
 *
 * Note: The license agreement is read from a text file that
 *       is stored at the location specified by the constant
 *       LICENSE_PATH. Before running this script, set that
 *       constant so that it references an existing text file
 *       in the Support Files/Billboards view.
 *
\*-----*/

#define LICENSE_PATH SUPPORTDIR ^ "License.txt"
#define TITLE "SdLicense Example"

#include "Ifx.h"

function OnBegin()
    STRING szMsg, szQuestion;
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Set up the variables to pass as parameters to SdLicense.
    szMsg      = "Please read the following license agreement. Use " +
                "the scroll bar to view\nthe rest of this agreement.";
    szQuestion = "Select Yes to accept the agreement.\n" +
                "Select No to cancel the setup.";

    // Display the SdLicense dialog.
    if (SdLicense (TITLE, szMsg, szQuestion, LICENSE_PATH) = YES) then
        MessageBox ("Continue with the installation.", INFORMATION);
    endif;

end;

```

## SdLicense2



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The [SdLicense2Ex](#) function supersedes the **SdLicense2** function.

The **SdLicense2** function displays a dialog that contains a license agreement in a multi-line edit field. The license agreement is stored in a text file identified in the parameter szLicenseFile.


The end user can scroll up and down to read the agreement. The end user must select the upper option button for the Next button to become enabled, which allows the installation to continue. Because this is usually the first dialog that you would display, you might want to disable the Back button.

## Syntax

```
SdLicense2 ( szTitle, szOpt1, szOpt2, szLicenseFile, bLicenseAccepted );
```

## Parameters

**Table 52 • SdLicense2 Parameters**

Parameter	Description
<b>szTitle</b>	Specify the title of the dialog. To display the default title ("License Agreement"), pass a null string ("") in this parameter.
<b>szOpt1</b>	Specify the text to display in the static text field next to the upper option button. To display the default text ("I accept the terms of the license agreement"), pass a null string ("") in this parameter.
<b>szOpt2</b>	Specify the text to display in the static text field next to the lower option button. To display the default text ("I do not accept the terms of the license agreement"), pass a null string ("") in this parameter.
<b>szLicenseFile</b>	Specify the name of the ANSI text file that contains the license agreement. This file must be added to the appropriate language folder in the Support Files/Billboards view. You can also specify szLicenseFile by entering the fully qualified name, in quotation marks, or a UNC path.   <b>Note</b> • This file must be a text file (.txt).
<b>bLicenseAccepted</b>	Specify the option button that is selected by default. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>TRUE</b>—Specifies the upper option button as the default selection.</li> <li>• <b>FALSE</b>—Specifies the lower option button as the default selection.</li> </ul>

## Return Values

**Table 53 • SdLicense2 Return Values**

Return Value	Description
<b>NEXT</b>	The end user selected the upper option button and the Next button.
<b>BACK</b>	The end user selected the Back button.
<b>&lt; ISERR_SUCCESS</b>	The dialog could not be displayed.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdLicense2 Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdLicense2 function.
 *
 * This script calls SdLicense2 to display a license agreement
 * and prompt the user to accept or reject that agreement.
 *
 * Note: The license agreement is read from a text file that
 *       is stored at the location specified by the constant
 *       LICENSE_PATH. Before running this script, set that
 *       constant so that it references an existing text file
 *       on the target system.
 *
\*-----*/

#define LICENSE_PATH "License.txt"
#define TITLE "SdLicense2 Example"

#include "ifx.h"

function OnBegin()
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Display the SdLicense2 dialog.
    if (SdLicense2 (TITLE, "", "", LICENSE_PATH, FALSE) = NEXT) then
        MessageBox ("Continue with the installation.", INFORMATION);
    endif;

end;
```

## SdLicense2Ex



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdLicense2Ex** function displays a dialog that contains a license agreement in a multi-line edit field. The license agreement is stored in a text file (.txt) or a rich text format file (.rtf) identified in the parameter `szLicenseFile`.

The end user can scroll up and down to read the agreement. The end user must select the upper option button for the Next button to become enabled, which allows the installation to continue. Because this is usually the first dialog that you would display, you might want to disable the Back button.


## Syntax

```
SdLicense2Ex (byval string szTitle, byval string szOpt1, byval string szOpt2, byval string  
szLicenseFile, byval bool bLicenseAccepted, byval bool bRtf);
```



## Parameters

**Table 54 • SdLicense2Ex Parameters**

Parameter	Description
<b>szTitle</b>	Specify the title of the dialog. To display the default title ("License Agreement"), pass a null string ("") in this parameter.
<b>szOpt1</b>	Specify the text to display in the static text field next to the upper option button. To display the default text ("I accept the terms of the license agreement"), pass a null string ("") in this parameter.
<b>szOpt2</b>	Specify the text to display in the static text field next to the lower option button. To display the default text ("I do not accept the terms of the license agreement"), pass a null string ("") in this parameter.
<b>szLicenseFile</b>	<p>Specify the name of the ANSI text file or .rtf file that contains the license agreement. This file must be added to the appropriate language folder in the Support Files/Billboards view. You can also specify szLicenseFile by entering the fully qualified name, in quotation marks, or a UNC path.</p>  <p><b>Note</b> • If you are using an .rtf file, the file size limit is 16 MB.</p>
<b>bLicenseAccepted</b>	<p>Specify the option button that is selected by default. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>TRUE</b>—Specifies the upper option button as the default selection.</li> <li>• <b>FALSE</b>—Specifies the lower option button as the default selection.</li> </ul>
<b>bRtf</b>	<p>Indicate whether to use the dialog with the rich text edit control. Available options are:</p> <ul style="list-style-type: none"> <li>• <b>TRUE</b>—Use the rich text edit control. If you select this option, the file that you specify for szLicenseFile must be an .rtf file.</li> <li>• <b>FALSE</b>—Use the standard edit control. If you select this option, the file that you specify for szLicenseFile must be a .txt file.</li> </ul>

## Return Values

**Table 55 • SdLicense2Ex Return Values**

Return Value	Description
<b>NEXT</b>	The end user selected the upper option button and the Next button.
<b>BACK</b>	The end user selected the Back button.
<b>&lt; ISERR_SUCCESS</b>	The dialog could not be displayed.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

# SdLicense2Rtf



---

**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The [SdLicense2Ex](#) function supersedes the **SdLicense2Rtf** function.

The **SdLicense2Rtf** function displays a dialog that contains a license agreement in a multi-line edit field. The license agreement is stored in a text file or rich text format file (.rtf) file identified in the parameter szLicenseFile.


The user can scroll up and down to read the agreement, then must select the upper option button for the Next button to become enabled, which allows the setup to continue. Because this is usually the first dialog you would display, you might want to disable the Back button.

## Syntax

```
SdLicense2Rtf ( szTitle, szOpt1, szOpt2, szLicenseFile, bLicenseAccepted );
```

## Parameters

**Table 56 • SdLicense2Rtf Parameters**

Parameter	Description
<b>szTitle</b>	Specify the title of the dialog. To display the default title ("License Agreement"), pass a null string ("") in this parameter.
<b>szOpt1</b>	Specify the text to display in the static text field next to the upper option button. To display the default text ("I accept the terms of the license agreement"), pass a null string ("") in this parameter.
<b>szOpt2</b>	Specify the text to display in the static text field next to the lower option button. To display the default text ("I do not accept the terms of the license agreement"), pass a null string ("") in this parameter.
<b>szLicenseFile</b>	Specify the name of the .rtf file or ANSI text file that contains the license agreement. This file must be added to the appropriate language folder in the Support Files/ Billboards view. You can also specify szLicenseFile by entering the fully qualified name, in quotation marks, or a UNC path.  
<b>Note</b> • If you are using an .rtf file, the file size limit is 16 MB.	
<b>bLicenseAccepted</b>	Specifies the option button that is selected by default. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>TRUE</b>—Specifies the upper option button as the default selection.</li> <li>• <b>FALSE</b>—Specifies the lower option button as the default selection.</li> </ul>

## Return Values

**Table 57 • SdLicense2Rtf Return Values**

Return Value	Description
<b>NEXT</b>	The end user selected the upper option button and the Next button.
<b>BACK</b>	The end user selected the Back button.
<b>&lt; ISERR_SUCCESS</b>	The dialog could not be displayed.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdLicense2Rtf Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdLicense2Rtf function.
 *
 * This script calls SdLicense2Rtf to display a license agreement
 * and prompt the user to accept or reject that agreement.
 *
 * Note: The license agreement is read from an RTF file that
 *       is stored at the location specified by the constant
 *       LICENSE_PATH. Before running this script, set that
 *       constant so that it references an existing RTF file
 *       on the target system.
 *
\*-----*/

#define LICENSE_PATH "License.rtf"
#define TITLE "SdLicense2Rtf Example"

#include "ifx.h"

function OnBegin()
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Display the SdLicense2Rtf dialog.
    if (SdLicense2Rtf (TITLE, "", "", LICENSE_PATH, FALSE) = NEXT) then
        MessageBox ("Continue with the installation.", INFORMATION);
    endif;

end;
```

## SdLicenseEx



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdLicenseEx** function displays a dialog that contains a license agreement in a multi-line edit field. The license agreement is stored in a text file (.txt) or a rich text format file (.rtf).


The end user can scroll up and down to read the agreement. The end user must choose either the Yes, No, or—if enabled—Back button. Because this is usually the first dialog that you would display, you might want to disable the Back button. If the user selects Yes, the installation continues. If the user selects No, the installation displays the ExitSetup dialog.

## Syntax

```
SdLicenseEx (byval string szTitle, byval string szMsg, byval string szQuestion, byval string  
            szLicenseFile, byval bool bRtf);
```

## Parameters

**Table 58 • SdLicenseEx Parameters**

Parameter	Description
<b>szTitle</b>	Specify the title of the dialog. To display the default title ("Software License Agreement"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specify the message to display in the static text field above the multi-line edit field. To display the default instructions, pass a null string ("") in this parameter.
<b>szQuestion</b>	Specify the text to display in the static text field below the multi-line edit field. You would likely place a question here, to which the user should respond by selecting either Yes and No. To display the default instructions, pass a null string ("") in this parameter.
<b>szLicenseFile</b>	Specify the name of the ANSI text file or .rtf file that contains the license agreement. This file must be added to the appropriate language folder in the Support Files/Billboards view. You can also specify szLicenseFile by entering the fully qualified name, in quotation marks, or a UNC path.   <b>Note</b> • If you are using an .rtf file, the file size limit is 16 MB.
<b>bRtf</b>	Indicate whether to use the dialog with the rich text edit control. Available options are: <ul style="list-style-type: none"> <li>• <b>TRUE</b>—Use the rich text edit control. If you select this option, the file that you specify for szLicenseFile must be an .rtf file.</li> <li>• <b>FALSE</b>—Use the standard edit control. If you select this option, the file that you specify for szLicenseFile must be a .txt file.</li> </ul>

## Return Values

**Table 59 • SdLicenseEx Return Values**

Return Value	Description
<b>YES (1)</b>	The end user selected the Yes button.
<b>BACK (12)</b>	The end user selected the Back button.



**Note** • This function cannot return NO because if the end user clicks the No button, the ExitSetup dialog is displayed.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

# SdLicenseRtf



---

**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The [SdLicenseEx](#) function supersedes the **SdLicenseRtf** function.

The **SdLicenseRtf** function displays a dialog that contains a license agreement in a multi-line edit field. The license agreement is stored in a text file or rich text format file (.rtf) file identified in the parameter `szLicenseFile`.

The user can scroll up and down to read the agreement, then must choose either the Yes, No, or—if enabled—the Back button. Because this is usually the first dialog you would display, you might want to disable the Back button. If the user selects Yes, the installation continues. If the user selects No, the installation displays the ExitSetup dialog.


## Syntax

```
SdLicenseRtf ( szTitle, szMsg, szQuestion, szLicenseFile );
```

## Parameters

**Table 60 • SdLicenseRtf Parameters**

Parameter	Description
<b>szTitle</b>	Specify the title of the dialog. To display the default title ("Software License Agreement"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specify the message to display in the static text field above the multi-line edit field. To display the default instructions, pass a null string ("") in this parameter.
<b>szQuestion</b>	Specify the text to display in the static text field below the multi-line edit field. You would likely place a question here, to which the user should respond by selecting either Yes and No. To display the default instructions, pass a null string ("") in this parameter.
<b>szLicenseFile</b>	Specify the name of the .rtf file or ANSI text file that contains the license agreement. This file must be added to the appropriate language folder in the Support Files/Billboards view. You can also specify szLicenseFile by entering the fully qualified name, in quotation marks, or a UNC path.



**Note** • If you are using an .rtf file, the file size limit is 16 MB.

## Return Values

**Table 61 • SdLicenseRtf Return Values**

Return Value	Description
<b>YES (1)</b>	The end user selected the Yes button.
<b>BACK (12)</b>	The end user selected the Back button.



**Note** • This function cannot return NO because if the end user clicks the No button, the ExitSetup dialog is displayed.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdLicenseRtf Example



**Project** • This information applies to the following project types:

- InstallScript



- *InstallScript MSI*

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdLicenseRtf function.
 *
 * This script calls SdLicenseRtf to display a license agreement
 * and prompt the user to accept or reject that agreement.
 *
 * Note: The license agreement is read from an RTF file that
 *       is stored at the location specified by the constant
 *       LICENSE_PATH. Before running this script, set that
 *       constant so that it references an existing RTF file
 *       in the Support Files/Billboards view.
 *
\*-----*/

#define LICENSE_PATH SUPPORTDIR ^ "License.rtf"
#define TITLE "SdLicenseRtf Example"

#include "Ifx.h"

function OnBegin()
    STRING szMsg, szQuestion;
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Set up the variables to pass as parameters to SdLicense.
    szMsg      = "Please read the following license agreement. Use " +
                "the scroll bar to view\nthe rest of this agreement.";
    szQuestion = "Select Yes to accept the agreement.\n" +
                "Select No to cancel the setup.";

    // Display the SdLicenseRtf dialog.
    if (SdLicenseRtf (TITLE, szMsg, szQuestion, LICENSE_PATH) = YES) then
        MessageBox ("Continue with the installation.", INFORMATION);
    endif;

end;

```

## SdLoadString

The **SdLoadString** function returns the string value associated with the specified resource ID. The InstallScript engine first looks for the resource in **\_isuser.dll**, if that file exists. If the resource is not found, the InstallScript engine looks in **\_isres.dll**.

### Syntax

```
SdLoadString (nID);
```

## Parameters

**Table 62** • SdLoadString Parameters

Parameter	Description
<b>nID</b>	<p>Specifies the identifier of a string resource in <b>_isuser.dll</b> or <b>_isres.dll</b>. Some valid values of nID can be found in the <i>InstallShield Program Files Folder\Script\Ifx\Include</i> subfolder's <b>Ifx.h</b> file. The default OnMaintUIBefore event handler function code uses the following identifiers with their corresponding string values:</p> <ul style="list-style-type: none"> <li>• <b>IFX_MAINTUI_MSG</b>—Do you want to completely remove the selected application and all of its components?</li> <li>• <b>IFX_ONMAINTUI_CAPTION</b>—Confirm Uninstall</li> </ul>

## Return Values

**Table 63** • SdLoadString Return Values

Return Value	Description
<b>non-null string value</b>	The string value successfully retrieved by <b>SdLoadString</b> .
<b>null string value ("")</b>	Indicates that <b>SdLoadString</b> failed.

## SdLoadString Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the SdLoadString function.
*
* This script calls SdLoadString to get the string resource
* that is stored in _isres.dll with the ID IFX_MAINTUI_MSG.
* That string resource is then displayed in a message box.
*
\*-----*/

#include "Ifx.h"

function OnBegin()
    STRING svResource;
begin

    // Get the string resource.
    svResource = SdLoadString (IFX_MAINTUI_MSG);
  
```

```

    if (svResource = "") then
        // Report the error.
        MessageBox ("SdLoadString failed.", WARNING);
    else
        // Display the string resource.
        sprintfBox ( INFORMATION , "SdLoadString" ,
            "Value of string resource %ld:\n%s" , IFX_MAINTUI_MSG,
            svResource );
    endif;
end;

```

## SdLogonUserBrowse



**Project** • This information applies to the following project types:

- InstallScript
- InstallScript MSI

The **SdLogonUserBrowse** function displays a dialog that allows the end user to select a domain or server and a user name. This dialog is displayed when an end user clicks the Browse button on the **SdLogonUserInformation** dialog.

### Syntax

```
SdLogonUserBrowse();
```

## SdLogonUserCreateUser



**Project** • This information applies to the following project types:

- InstallScript
- InstallScript MSI

The **SdLogonUserCreateUser** function displays a dialog that allows the end user to enter new user information after the end user clicks the New User Information button on the **SdLogonUserInformation** dialog.

### Syntax

```
SdLogonUserCreateUser();
```

## SdLogonUserInformation



**Project** • This information applies to the following project types:

- InstallScript

- *InstallScript MSI*

The **SdLogonUserInformation** function displays a dialog that prompts the end user for existing user account information or new user information if an account is to be created during the installation.

If the end user clicks the Browse button next to the User box, the **SdLogonUserBrowse** dialog is displayed.

## Syntax

```
SdLogonUserInformation (byval string szTitle, byval string szMsg, byref string szAssociatedAccountName,
    byref string szAssociatePassword);
```

## Parameters

**Table 64** • SdLogonUserInformation Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title, pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message in the dialog. To display the default title, pass a null string ("") in this parameter.
<b>szAssociatedAccountName</b>	Specifies the BYREF value that is the account name. Example: INSTALLSHIELD\John Smith
<b>szAssociatePassword</b>	Specifies the BYREF value that is the password.

## Return Values

**Table 65** • SdLogonUserInformation Return Values

Return Value	Description
<b>NEXT</b>	Indicates that the user selected the Next button
<b>BACK</b>	Indicates that the user selected the Back button.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the dialog could not be displayed.

# SdLogonUserListGroup



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdLogonUserListGroup** function displays a dialog that allows the end user to select a group from a specified server or domain and populate the Group field in the SdLogonUserCreateUser dialog.

## Syntax

```
SdLogonUserListGroup();
```

# SdLogonUserListServers



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdLogonUserListServers** function displays a dialog that allows the end user to browse for the domain or server with which the user account is associated.

## Syntax

```
SdLogonUserListServers();
```

# SdLogonUserListUsers



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdLogonUserListUsers** function displays a dialog that allows the end user to browse and select an existing user for a specified domain or server.

## Syntax

```
SdLogonUserListUsers();
```

# SdMakeName

The **SdMakeName** function creates a section name for a custom dialog. This section name is used in writing to and reading from an .iss file, which is used when running an InstallScript-based installation project's Setup.exe in silent mode.

## Syntax

```
SdMakeName ( svSection, szDlg, szUnused, nvDlgName );
```

## Parameters

**Table 66 • SdMakeName Parameters**

Parameter	Description
<b>svSection</b>	Specifies the section name (for example, "MyDlg-0"). InstallShield places a value into this variable using the variables szDlg and nvDlgName. This value is used by SilentReadData and SilentWriteData.
<b>szDlg</b>	Specifies the name of the custom dialog (for example, "MyDlg") for which to create a section name.
<b>szUnused</b>	This parameter is not used; pass a null string ("") in this parameter.
<b>nvDlgName</b>	<p>Specifies the counter that records the number of times SdMakeName is called for the dialog named in szDlg. InstallShield automatically increments this counter.</p> <p>For sections to be properly named, you must use a unique variable name in this parameter for each different custom dialog. A simple way to do this is to use the dialog name in szDlg to name the variable. For example, when szDlg is "MyDlgOne," name the variable in this parameter nvMyDlgOne, and when szDlg is "MyDlgTwo," name the variable nvMyDlgTwo.</p>

## Return Values

None.

## SdMakeName Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the functions SdMakeName, SilentReadData, and
* SilentWriteData.
*
* This example script shows how to handle a custom dialog
* in a silent installation. The resource .dll for the example
* custom dialog shown below should be stored in a compressed
* form under the Support Files/Billboards view of InstallShield.
*
* The example dialog was built from the custom dialog
* template provided with InstallShield.
*
* Dialog control IDs and other information are included in
* the Resource.h file (not shown). This file, which is included
* in the first line of the example, must be inserted in the
```

```

* InstallScript view of InstallShield.
*
* The example creates a text file called Cominit.txt. If the
* installation runs in silent mode, SilentReadData is called
* and the custom dialog control selections are read from
* the .iss file. The selections are then saved in the file
* Cominit.txt as a means of demonstrating that they were
* successfully read from the .iss file. If the installation
* runs in normal mode, the custom dialog is displayed
* and the selections are recorded in the .iss file and
* displayed in message boxes. The initial .ISS file text is
* shown after the example script.
*
\*-----*/

#include "Resource.h"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_SdMakeName(HWND);

function ExFn_SdMakeName(hMSI)
  BOOL bDone;
  STRING svSection, svComPort, svPulse, svTone, svDial9, svVal;
  NUMBER nvCommDialog, nCmdValue, nPulseState, nToneState;
  NUMBER nDial9State, nResult, nvHandle;
  LIST listID;
  HWND hwndDlg;
begin

  // Open the text file COMINIT.TXT so custom dialog selections
  // from the .ISS file can be stored in it.
  OpenFileMode (FILE_MODE_APPEND);
  OpenFile (nvHandle, "c:\\rul", "cominit.txt");

  // If operating in silent mode, then read from the .ISS file.
  if (MODE=SILENTMODE) then
    SdMakeName (svSection, "COMM_DIALOG", "", nvCommDialog);

    SilentReadData (svSection, "Result", DATA_NUMBER, svVal, nResult);

    if (nResult = 1) then

      // Read the data from the .ISS file. For purposes of
      // writing the results to a text file, read the
      // data as strings.
      SilentReadData (svSection, "nPulseState", DATA_STRING,
        svPulse, nResult);

      SilentReadData (svSection, "nToneState", DATA_STRING,
        svTone, nResult);

      SilentReadData (svSection, "nDial9State", DATA_STRING,
        svDial9, nResult);

      // Store the custom dialog selections in
      // the text file COMINIT.TXT.
      svVal = "Pulse box is: " ^ svPulse;

```

```

        WriteLine(nvHandle, svVal);

        svVal = "Tone box is: " ^ svTone;
        WriteLine(nvHandle, svVal);

        svVal = "Dial9 box is: " ^ svDial9;
        WriteLine(nvHandle, svVal);
    endif;

    // If not in silent mode, then call and handle the custom dialog
    // as you normally would.
    else
        listID = ListCreate (STRINGLIST);
        ListAddString (listID, "COMM1:", AFTER);
        ListAddString (listID, "COMM2:", AFTER);
        ListAddString (listID, "COMM3:", AFTER);
        ListAddString (listID, "COMM4:", AFTER);

        EzDefineDialog ("MYCOMDIALOG", SUPPORTDIR^"RESOURCE.DLL",
                        "COMM_DIALOG", 0);

        bDone = FALSE;

        while (bDone=FALSE)
            nCmdValue = WaitOnDialog ("MYCOMDIALOG");

            switch (nCmdValue)
                case DLG_INIT:
                    // Initialize the back, next, and cancel button enable/disable states
                    // for this dialog and replace %P, %VS, %VI with
                    // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
                    // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724
                    // and 202.
                    hwndDlg = CmdGetHwndDlg("MYCOMDIALOG");
                    SdGeneralInit("MYCOMDIALOG", hwndDlg, 0, "");
                    CtrlSetState ("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);
                    CtrlSetList ("MYCOMDIALOG", ID_COMPORT, listID);
                    CtrlSetState ("MYCOMDIALOG", ID_DIAL9, BUTTON_CHECKED);
                case OK:
                    CtrlGetCurSel ("MYCOMDIALOG", ID_COMPORT, svComPort);
                    nPulseState = CtrlGetState ("MYCOMDIALOG", ID_PULSE);
                    nToneState = CtrlGetState ("MYCOMDIALOG", ID_TONE);
                    nDial9State = CtrlGetState ("MYCOMDIALOG", ID_DIAL9);
                    nResult = NEXT;
                    bDone = TRUE;
                case BACK:
                    nResult = BACK;
                    bDone = TRUE;
                case RES_PBTN_CANCEL:
                    // The user clicked the Cancel button.
                    Do (EXIT);
                case DLG_CLOSE:
                    // The user clicked the window's close button.
                    Do (EXIT);
                case ID_PULSE:
                    nPulseState = CtrlGetState ("MYCOMDIALOG", ID_PULSE);

                    if (nPulseState = BUTTON_CHECKED) then
                        CtrlSetState ("MYCOMDIALOG", ID_TONE, BUTTON_UNCHECKED);

```



```

        CtrlSetState ("MYCOMDIALOG", ID_PULSE, BUTTON_CHECKED);
    else
        CtrlSetState ("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);
        CtrlSetState ("MYCOMDIALOG", ID_PULSE, BUTTON_UNCHECKED);
    endif;
case ID_TONE:
    nToneState = CtrlGetState ("MYCOMDIALOG", ID_TONE);

    if (nPulseState = BUTTON_CHECKED) then
        CtrlSetState ("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);
        CtrlSetState ("MYCOMDIALOG", ID_PULSE, BUTTON_UNCHECKED);
    else
        CtrlSetState ("MYCOMDIALOG", ID_TONE, BUTTON_UNCHECKED);
        CtrlSetState ("MYCOMDIALOG", ID_PULSE, BUTTON_CHECKED);
    endif;
case DLG_ERR:
    MessageBox ("Unable to display dialog. Setup canceled.", SEVERE);
    abort;
endswitch;

endwhile;

EndDialog ("MYCOMDIALOG");
ReleaseDialog ("MYCOMDIALOG");

SdMakeName (svSection, "COMM_DIALOG", "", nvCommDialog);

SilentWriteData (svSection, "nPulseState", DATA_NUMBER,
    svPulse, nPulseState);

SilentWriteData (svSection, "nToneState", DATA_NUMBER,
    svTone, nToneState);
SilentWriteData (svSection, "nDial9State", DATA_NUMBER,
    svDial9, nDial9State);

if (nPulseState = BUTTON_CHECKED) then
    MessageBox ("The Pulse button was checked.", INFORMATION);
else
    MessageBox ("The Pulse button was unchecked.", INFORMATION);
endif;

if (nToneState = BUTTON_CHECKED) then
    MessageBox ("The Tone button was checked.", INFORMATION);
else
    MessageBox ("The Tone button was unchecked.", INFORMATION);
endif;

if (nDial9State = BUTTON_CHECKED) then
    MessageBox ("The Dial9 button was checked.", INFORMATION);
else
    MessageBox ("The Dial9 button was unchecked.", INFORMATION);
endif;

endif;

// Close the text file COMINIT.TXT
CloseFile (nvHandle);

end;

```

/\*The following is the initial .iss file text for the above example, where <PRODUCT\_GUID> represents your project's GUID, including the surrounding braces. Note that -1001 is the numeric value of BUTTON\_CHECKED and -1002 is the numeric value of BUTTON\_UNCHECKED.

```
[InstallShield Silent]
Version=v9.00
File=Response File
[Application]
Name=MyDialog
Version=4.0
Company=My Software Company
[<PRODUCT_GUID>-DlgOrder]
Dlg0=<PRODUCT_GUID>-COMM_DIALOG-0
Count=1
[<PRODUCT_GUID>-COMM_DIALOG-0]
nPulseState=-1001
nToneState=-1002
nDlg9State=-1001
Result=1
*/
```

## SdOptionsButtons



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdOptionsButtons** function displays a dialog that contains from one to four push button controls that have bitmap images. A short text description is displayed next to each control.



**Important** • Do not change the control identifiers of the push button controls on the **SdOptionsButtons** dialog, or set one of the push button controls as the default control. If you change the control identifiers or set one of these controls as the default control of the dialog, the **SdOptionsButtons** dialog will not work as expected.



**Note** • Although **SdOptionsButtons** can be used as a setup type dialog, the **SdSetupTypeEx** dialog is the recommended dialog for allowing the end user to select a setup type because it does not require any customization. If you call **SdOptionsButtons** to get the end user's setup type selection, you must then call **FeatureSetupTypeSet** to establish the selected setup type for your setup.

### Syntax


```
SdOptionsButtons (szTitle, szMsg, listButton, listDescription);
```

## Parameters

**Table 67 • SdOptionsButtons Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the dialog title. To display the default title ("Select Features"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message to display on the dialog. To display the default instructions for this dialog ("Please select the features that you want to install."), pass a null string ("") in this parameter.
<b>listButton</b>	<p>Specifies a string list that contains from one to four elements. Each element is a formatted string that specifies an image to be displayed on a button. One button is displayed for each string element in the list. The string list elements should have the following format:</p> <p><code>@@ResourceID;ScaleFactor</code></p> <p><i>ResourceID</i> indicates the resource identifier number that should be used to look up either a .png image (stored as resource type PNG) or a bitmap image (stored as resource type RT_BITMAP). <i>ScaleFactor</i> indicates the DPI scale percentage for which the image is intended.</p> <p>For example, the scale factor can be 100% (96 DPI), 125% (120 DPI), 150% (144 DPI), or 200% (192 DPI). If the scale factor that is specified for an image is 200, the image will be shrunk down for display on target systems that are running less than 200% DPI scaling. On a 200% target system, it will be displayed at 1:1. If the scale factor that is specified for the image is 100, the image will be scaled up for display on target systems that are running 200% DPI scaling.</p>

**Table 67 • SdOptionsButtons Parameters (cont.)**

Parameter	Description
<b>listButton (cont.)</b>	<p>The previous format for identifying bitmap images (.bmp) still works, but it does not have support for scaling, or for .png images:</p> <pre>@BitmapResourceID;TransparentFlag;3-DFlag;&lt;unused&gt;;TransparentColorKey</pre> <p><i>BitmapResourceID</i> indicates the resource identifier number for the bitmap image. <i>TransparentFlag</i> is 1 (true) or 0 (false), indicating whether the color that is specified in the <i>TransparentColorKey</i> field will be transparent when the bitmap is displayed. <i>TransparentColorKey</i> specifies an RGB value that is a transparent color for the bitmap.</p> <p>InstallShield provides four default bitmaps in <b>_isres.dll</b> that can be used by this function. These bitmaps have IDs of 12001 through 12004 and correspond to the Typical, Portable, Compact, and Custom setup types in the example script. If you are using this dialog for another purpose or if you want to use setup types other than the types provided in <b>_isres.dll</b>, you must to add your own custom buttons to the <b>_isuser.dll</b> dialog template and then include the customized <b>_isuser.dll</b> with your installation.</p> <div>  <p><b>Tip •</b> To determine the bitmap IDs of bitmap controls created in the Dialog Editor, you need to build your project and open the .rc file to locate the bitmap IDs. The .rc file is located in your project folder (by default, C:\InstallShield 2016 Projects\Your Project Name). Adding additional bitmap controls may change the bitmap IDs. Because of this, you need to rebuild your project and open the .rc file again to find the updated bitmap IDs.</p> </div>
<b>listDescription</b>	Specifies a string list that contains from one to four string elements, each corresponding to a string in the parameter listButton. Each string is a text description to be displayed with its corresponding button.

## Return Values

**Table 68** • SdOptionsButtons Return Values

Return Value	Description
<b>NEXT (1)</b>	Indicates that the Next button was clicked.
<b>BACK (12)</b>	Indicates that the Back button was clicked.
<b>101</b>	Indicates that the button corresponding to the first string element in listButton was selected.
<b>102</b>	Indicates that the button corresponding to the second string element in listButton was selected.
<b>103</b>	Indicates that the button corresponding to the third string element in listButton was selected.
<b>104</b>	Indicates that the button corresponding to the fourth string element in listButton was selected.



**Note** • To prevent the user from exiting the dialog without clicking a particular button, call the **Disable** function to disable the Next button before you call **SdOptionsButtons**.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdOptionsButtons Example



**Project** • This information applies to the following project types:

- InstallScript
- InstallScript MSI

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the SdOptionsButtons function.
*
* This script allows the user to select a setup type. First,
* two lists are created, one for the setup type button icons,
* another for the setup type descriptions. Then the buttons
* and descriptions are added to the lists. Next, the dialog
* is presented. When the user clicks a setup type button, the
* dialog closes and the user's selection is displayed in a
```

```

* message box.
*
\*-----*/

#include "Ifx.h"

function OnBegin()
    STRING  szTitle, szMsg;
    LIST    listButton, listDesc;
    NUMBER  nResult;
begin

    // Disable the Back and Next buttons in setup dialogs.
    Disable (BACKBUTTON);
    Disable (NEXTBUTTON);

    // Create the lists for buttons and descriptions.
    listButton = ListCreate (STRINGLIST);
    listDesc = ListCreate (STRINGLIST);

    if (listButton = LIST_NULL) || (listDesc = LIST_NULL) then
        // Report the error; then terminate.
        MessageBox ("Unable to create lists.", INFORMATION);
        abort;
    endif;

    // Add the bitmap buttons to listButton.
    ListAddString (listButton, "@12001;1;255,0,255", AFTER);
    ListAddString (listButton, "@12002;1;255,0,255", AFTER);
    ListAddString (listButton, "@12003;1;255,0,255", AFTER);
    ListAddString (listButton, "@12004;1;255,0,255", AFTER);

    // Add the descriptions to listDesc.
    ListAddString (listDesc, "Typical\n" +
        "Recommended for most computers.", AFTER);
    ListAddString (listDesc, "Portable\n" +
        "The application will be set up with " +
        "options that are useful for portable " +
        "computers.", AFTER);
    ListAddString (listDesc, "Compact\n" +
        "To save disk space, none of the " +
        "optional features will be " +
        "installed.", AFTER);
    ListAddString (listDesc, "Custom\n" +
        "For advanced users and system " +
        "administrators only. You can " +
        "customize all available Setup " +
        "options.", AFTER);

    // Display the dialog.
    nResult = SdOptionsButtons (szTitle, szMsg, listButton, listDesc);

    // Display a message showing which button was selected.
    switch (nResult)
        case 101:
            MessageBox ("Typical installation selected.", INFORMATION);
        case 102:
            MessageBox ("Portable installation selected.", INFORMATION);
        case 103:

```

```

        MessageBox ("Compact installation selected.", INFORMATION);
    case 104:
        MessageBox("Custom installation selected.", INFORMATION);
    default:
        MessageBox ("SdOptionsButtons:\n\n An error occurred.", SEVERE);
endswitch;

Enable(NEXTBUTTON);

// Destroy the lists.
ListDestroy (listButton);
ListDestroy (listDesc);

end;

```

## SdOutOfDiskSpace



**Project** • This information applies to InstallScript MSI projects.

The **SdOutOfDiskSpace** function displays a dialog warning the end user that the target system does not have enough available space for the application installation to take place. It also displays a list view of volumes, required space, available space, and the difference between available space and required space.

In an InstallScript MSI installation, this function is triggered when an `INSTALLMESSAGE_OUTOFDISKSPACE` message is detected. This message comes from the MSI engine.

The **SdDiskSpace2** function supersedes the **SdOutOfDiskSpace** function.

### Syntax

```
SdOutOfDiskSpace ( szTitle, szMsg );
```

## Parameters

**Table 69** • SdOutOfDiskSpace Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Out of Disk Space"), pass a null string ("" ) in this parameter.
<b>szMsg</b>	Specifies the message to display in the dialog. This text is considered a static control. To display the default instructions for this dialog, pass a null string ("" ) in this parameter.

## Return Values

**Table 70** • SdOutOfDiskSpace Return Values

Return Value	Description
<b>NEXT (1)</b>	Indicates that the end user clicked the OK button.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

# SdPatchWelcome



**Project** • This information applies to InstallScript MSI projects.

The **SdPatchWelcome** function creates a dialog that displays a welcome message to the end user during a patch installation.

## Syntax

```
SdPatchWelcome ( szTitle, szMsg );
```



## Parameters

**Table 71** • SdPatchWelcome Parameters

Parameter	Description
<b>szTitle</b>	Specifies the text to display in the title of the dialog. To display the default title ("Welcome"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message to display in the Welcome dialog. To include in this message the product name set by a previous call to <a href="#">SdProductName</a> , insert the place holder %P anywhere in the message string. When the message is displayed, %P is replaced by the product name. To display the default welcome message, pass a null string ("") in this parameter.

## Return Values

**Table 72** • SdPatchWelcome Return Values

Return Value	Description
<b>NEXT (1)</b>	Indicates that the Next button was clicked.

## Additional Information

- To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.
- If you call **SdPatchWelcome** but SD\_NDLG\_PATCHWELCOME—its dialog resource (12059)—cannot be found in the **\_isres.dll** file of the installation, **SdWelcome** is called automatically instead of **SdPatchWelcome** to display a dialog that is similar to the **SdPatchWelcome** dialog. In this case, the values for szTitle and szMsg are passed to the **SdWelcome** function, or custom strings are used if szTitle and szMsg are blank. This occurs only if you are patching a product whose installation was created with InstallShield Developer 7, or the dialog template has been removed manually.

Note that when this occurs, the dialog uses the same template as **SdWelcome**; thus, any changes to the **SdWelcome** dialog template affect the **SdPatchWelcome** dialog as well.

Note that in this scenario, silent mode is not supported, and it does not work correctly.

## SdPatchWelcome Example



**Project** • This information applies to InstallScript MSI projects.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdPatchWelcome function.
 *
 * This function displays a welcome message to the end user during
```

```
* a patch installation. This function is called in the default
* script for the OnPatchUIBefore event, which can be found in the
* Miscellaneous event category for InstallScript MSI projects.
*
\*-----*/

#include "Ifx.h"

function OnBegin()
    STRING szTitle, szMsg;
begin
    // Display SdPatchWelcome with default title and message
    szTitle = "";
    szMsg   = "";
    SdPatchWelcome(szTitle, szMsg);

end;
```

# SdProductName


The **SdProductName** function sets the value of the system variable IFX\_PRODUCT\_DISPLAY\_NAME, which makes your product name available to all instances of the %P place holder. The %P place holder is found in static text fields in some Sd dialogs. In addition, some Sd dialog functions, such as SdFinish, allow you to include %P in strings passed as parameters to functions.

## Syntax

```
SdProductName ( szProductName );
```

## Parameters

Table 73 • SdProductName Parameters

Parameter	Description
<b>szProductName</b>	Specifies the name of the product that is being installed. This name replaces the product name placeholder (%P) wherever it appears in appropriate static fields in Sd dialogs.
	 <b>Caution</b> • If you want to include an ampersand (&) in your product name, you must use two ampersands (&&) to display the name properly in end user dialogs. For example, to display “New & Improved Product”, you should enter the product name as New && Improved Product.

## Return Values

This function does not return a value.

## Additional Information

Rather than call SdProductName, for example,

```
SdProductName( "My Product Name" );
```

you can simply assign the desired value to IFX\_PRODUCT\_DISPLAY\_NAME, for example,

```
IFX_PRODUCT_DISPLAY_NAME = "My Product Name";
```

## SdProductName Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the functions SdProductName.
 *
 * The function SdProductName is called to set the product name to
 * substitute for the %P place holder, which is embedded in the
 * message strings passed to Sd dialogs.
 *
 \*-----*/

#include "Ifx.h"

function OnBegin()
    STRING    szProductName, szTitle;
begin

    // Set the product name to substitute for the %P place holder.
    szProductName = "My Application";
    SdProductName (szProductName);

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Display the SdWelcome dialog. The null string in parameter
    // two specifies the default message, which uses the %P place holder.
    szTitle = "SdProductName Example";
    SdWelcome (szTitle, "");

end;
```

## SdRegisterUser



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdRegisterUser** function displays a dialog that enables the end user to specify the user name and company name for the product being installed.

You can specify default values for these fields by specifying the appropriate parameters. If you specify a null string (""), the function uses the appropriate script variable.

The Next button becomes enabled only when data exists in both edit fields. The end user cannot leave any field blank.

## Syntax

```
SdRegisterUser ( szTitle, szMsg, svName, svCompany );
```

## Parameters

**Table 74 • SdRegisterUser Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("User Information"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message to display in the dialog. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>svName</b>	<p>Specifies the default value for the Name edit field when the function is called.</p> <p>If a null string ("") is specified, the default value is the current value of the <a href="#">IFX_PRODUCT_REGISTEREDOWNER</a> variable. For an InstallScript project, this variable is read from the registry in a first-time installation or the corresponding text substitution in maintenance mode. For an InstallScript MSI project, the default value of the variable is always read from the Windows Installer property <b>USERNAME</b>.</p> <p>The function returns the value that the end user specified in this parameter. In an InstallScript installation, the function also sets the value of <i>IFX_PRODUCT_REGISTEREDOWNER</i> to the value that the end user specified. In an InstallScript MSI installation, the function automatically updates the Windows Installer property <b>USERNAME</b>.</p>
<b>svCompany</b>	<p>Specifies the default value for the Company edit field when the function is called.</p> <p>If a null string ("") is specified, the default value is the current value of the <a href="#">IFX_PRODUCT_REGISTEREDCOMPANY</a> variable. For an InstallScript project, this variable is read from the registry in a first-time installation or the corresponding text substitution in maintenance mode. For an InstallScript MSI project, the default value of the variable is always read from the Windows Installer property <b>COMPANYNAME</b>.</p> <p>The function returns the value that the end user specified in this parameter. In an InstallScript installation, the function also sets the value of <i>IFX_PRODUCT_REGISTEREDCOMPANY</i> to the value that the end user specified. In an InstallScript MSI installation, the function automatically updates the Windows Installer property <b>COMPANYNAME</b>.</p>

## Return Values

**Table 75 • SdRegisterUser Return Values**

Return Value	Description
<b>NEXT (1)</b>	Indicates that the Next button was clicked.
<b>BACK (12)</b>	Indicates that the Back button was clicked.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdRegisterUser Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdRegisterUser and SdConfirmRegistration
 * functions.
 *
 * SdRegisterUser is called to prompt for the user's name
 * and company name. These entries are then confirmed when
 * SdConfirmRegistration is called.
 *
 \*-----*/

#define REG_TITLE      "SdRegisterUser Example"
#define REG_MSG        "Please register your product now."
#define CONFIRM_TITLE  "SdConfirmRegistration Example"

#include "Ifx.h"

function OnBegin()
    STRING  svName, svCompany;
    NUMBER  nResult;
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    repeat
        // Get the user's name and company name.
        SdRegisterUser (REG_TITLE, REG_MSG, svName, svCompany);

        // Confirm that the information is correct. Pass a null string in
        // parameter four since SdRegisterUser does not get a serial number.
        nResult = SdConfirmRegistration (CONFIRM_TITLE, svName, svCompany, "", 0);
    until nResult = YES;

end;
```

# SdRegisterUserEx



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdRegisterUserEx** function displays a dialog that enables the end user to specify the user name, company name, and serial number for the product being installed.

You can specify default values for these fields by specifying the appropriate parameters. If you specify a null string (""), the function uses the appropriate script variable.

The Next button becomes enabled only when data exists in all three edit fields. The end user cannot leave any field blank.



**Note** • The **SdRegisterUserEx** function does not verify the serial number. To learn how to add code that verifies the serial number, see the sample serial number validation project. This sample project is in one of the Samples subfolders within the InstallShield Program Files folder. The default installation location is **C:\Program Files\InstallShield\2016\Samples\InstallScript\Serial Number Validation Sample Project**.

## Syntax

```
SdRegisterUserEx ( szTitle, szMsg, svName, svCompany, svSerial );
```

## Parameters

**Table 76 • SdRegisterUserEx Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("User Information"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message to display in the dialog. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>svName</b>	<p>Specifies the default value for the Name edit field when the function is called.</p> <p>If a null string ("") is specified, the default value is the current value of the <a href="#">IFX_PRODUCT_REGISTEREDOWNER</a> variable. For an InstallScript project, this variable is read from the registry in a first-time installation or the corresponding text substitution in maintenance mode. For an InstallScript MSI project, the default value of the variable is always read from the Windows Installer property <b>USERNAME</b>.</p> <p>The function returns the value that the end user specified in this parameter. In an InstallScript installation, the function also sets the value of <i>IFX_PRODUCT_REGISTEREDOWNER</i> to the value that the end user specified. In an InstallScript MSI installation, the function automatically updates the Windows Installer property <b>USERNAME</b>.</p>
<b>svCompany</b>	<p>Specifies the default value for the Company edit field when the function is called.</p> <p>If a null string ("") is specified, the default value is the current value of the <a href="#">IFX_PRODUCT_REGISTEREDCOMPANY</a> variable. For an InstallScript project, this variable is read from the registry in a first-time installation or the corresponding text substitution in maintenance mode. For an InstallScript MSI project, the default value of the variable is always read from the Windows Installer property <b>COMPANYNAME</b>.</p> <p>The function returns the value that the end user specified in this parameter. In an InstallScript installation, the function also sets the value of <i>IFX_PRODUCT_REGISTEREDCOMPANY</i> to the value that the end user specified. In an InstallScript MSI installation, the function automatically updates the Windows Installer property <b>COMPANYNAME</b>.</p>
<b>svSerial</b>	<p>Specifies the default value for the Serial Number edit field when the function is called.</p> <p>If a null string ("") is specified, the default value is the current value of the <a href="#">IFX_PRODUCT_REGISTEREDSERIALNUM</a> variable. This variable is read from the registry in a first-time installation or the corresponding text substitution in maintenance mode.</p> <p>The function returns the value that the end user specified in this parameter. The function also sets the value of <i>IFX_PRODUCT_REGISTEREDSERIALNUM</i> to the value that the end user specified.</p>



## Return Values

**Table 77 • SdRegisterUserEx Return Values**

Return Value	Description
<b>NEXT (1)</b>	Indicates that the Next button was clicked.
<b>BACK (12)</b>	Indicates that the Back button was clicked.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdRegisterUserEx Example



**Project •** This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the function SdRegisterUserEx.
 *
 * SdRegisterUserEx is called to collect installation information
 * from the user. The information is stored in a list and
 * displayed in SdShowInfoList.
 *
 \*-----*/

#include "Ifx.h"

function OnBegin()
    STRING  szTitle, szMsg, svName, svCompany, svSerial;
    LIST    listData;
begin

    // Create the list.
    listData = ListCreate (STRINGLIST);

    // Set up parameters for call to SdRegisterUserEx.
    szTitle = "SdRegisterUserEx Example";
    szMsg   = "Please enter your name, company, and serial number.";

    // Retrieve registration information.
    SdRegisterUserEx (szTitle, szMsg, svName, svCompany, svSerial);

    // Add the information to the list.
    ListAddString (listData, "User Information: ", AFTER);
```

```

ListAddString (listData, "          " + svName, AFTER);
ListAddString (listData, "          " + svCompany, AFTER);
ListAddString (listData, "          " + svSerial, AFTER);
ListAddString (listData, "", AFTER);

// Display user selections from list in SdShowInfoList.
szMsg  = "The user name, company name, and serial number " +
        "entered in SdRegisterUserEx.";
SdShowInfoList(szTitle, szMsg, listData);

end;

```

## SdRMFilesInUse



**Project** • The *InstallScript* MSI project type support the **SdFilesInUse** function.

The **SdRMFilesInUse** function displays a dialog that includes a list box containing a list of the applications that are open and are locking files. The dialog also includes two radio buttons that allow end users to specify whether the installation should attempt to use the Restart Manager to shut down the applications that are locking files or overwrite the locked files (which most likely results in the need for a reboot to complete the installation).

Typically, the OnRMFilesInUse event handler displays this dialog in an *InstallScript* MSI installation as a response to an `INSTALLMESSAGE_FILESINUSE` message sent by the Windows Installer. When this occurs, the Windows Installer provides to the installation the list of applications that are locking files. The list of applications are passed through the `szMessage` parameter to the OnRMFilesInUse event. The event passes this information to the **SdRMFilesInUse** function through the `szMessage` parameter. The event then passes the return value from the function as the return value of the event, which causes the Windows Installer to act appropriately.



**Project** • The **SdRMFilesInUse** dialog can also be called manually in *InstallScript* projects and in *InstallScript* MSI projects; however, the installation must provide the list of applications that are locking files through the `nvlistApps` parameter, and it must handle the return value appropriately. It must also interact with the Restart Manager to attempt to shut down applications that are locking files if the end user selects the corresponding option on the **SdRMFilesInUse** dialog.

### Syntax



```

SdRMFilesInUse ( byval string szTitle, byval string szMsg, byval string szFilesInUse, byref LIST
                nvlistApps, byref BOOL bvCloseRestart );

```


## Parameters

**Table 78 • SdRMFilesInUse Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title (Files in Use), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message to display in the dialog. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>szFilesInUse</b>	<p>When <b>SdRMFilesInUse</b> is called by the OnRMFilesInUse event handler in an InstallScript MSI installation, this parameter specifies the list of applications that the Windows Installer found to be locking files. The function parses this string in order to display the list of applications in the dialog's list box.</p> <p>If you manually call <b>SdFilesInUse</b>, pass an empty string ("") in this parameter. Note that if nvlistApps is a valid string list, this parameter is ignored.</p>
<b>nvlistApps</b>	<p>When <b>SdRMFilesInUse</b> is called by the OnRMFilesInUse event handler in an InstallScript MSI installation, this parameter is specified as an uninitialized list variable—that is, a variable with a value of 0.</p> <p>If you manually call <b>SdRMFilesInUse</b>, specify a string list of applications that are locking files and that should be displayed in the dialog. Each string in the list indicates one application. (Use the <b>ListCreate</b> function and associated list functions to create and initialize the string list.)</p> <p></p> <p><b>Note</b> • You must provide a variable for this parameter; you cannot specify a literal value.</p>
<b>bvCloseRestart</b>	<p>Indicates the default for the aforementioned radio buttons. TRUE indicates that the close-and-restart-applications radio button is the default. FALSE indicates that the ignore-and-reboot-later radio button should be the default.</p> <p></p> <p><b>Project</b> • In InstallScript MSI projects, the OnRMFilesInUse event sets the value of bvCloseRestart based on the current value of the Windows Installer property called MSIRESTARTMANAGERCONTROL Property.</p> <p>When the <b>SdRMFilesInUse</b> function returns, this value indicates which radio button the end user selected—TRUE for the close-and-restart-applications option or FALSE for the ignore-and-reboot-later option. Note that the function also returns the same information in the return value of the function.</p>

## Return Values

**Table 79** • SdRMFilesInUse Return Values

Return Value	Description
<b>IDOK</b>	Indicates that the end user selected the close-and-restart-applications option and then clicked the OK button.
<b>IDIGNORE (5)</b>	Indicates that the end user selected the ignore-and-reboot-later option and then clicked the OK button. Note that this value can be returned even though the dialog does not include an Ignore button.
<b>IDCANCEL</b>	Indicates that the end user clicked the Exit button.
	 <p><b>Note</b> • Note that unlike other script dialogs, this dialog does not call the OnCanceling event handler when the user clicks the Exit button. Therefore, if you are calling this function manually, you must handle this return value manually in order to display the Cancel Setup dialog.</p>

## SdSelectFolder



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdSelectFolder** function displays folders for selection. **SdSelectFolder** enables you to offer a default selection. The end user can also enter a new folder name. **SdSelectFolder** returns only the selected or entered folder name. It cannot create the folder.

### Syntax

```
SdSelectFolder ( szTitle, szMsg, svDefGroup );
```

## Parameters

**Table 80** • SdSelectFolder Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Select Program Folder"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message to display in the dialog. This text is considered a static control. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>svDefGroup</b>	Returns the name of the selected folder.

## Return Values

**Table 81** • SdSelectFolder Return Values

Return Value	Description
<b>NEXT (1)</b>	Indicates that the Next button was clicked.
<b>BACK (12)</b>	Indicates that the Back button was clicked.

## Additional Information

- To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.
- If **ProgDefGroupType** is called before calling **SdSelectFolder**, the program folders displayed by **SdSelectFolder** (Common or Personal) depend on the parameter that was passed to **ProgDefGroupType**.

## SdSelectFolder Example



**Project** • This information applies to the following project types:

- InstallScript*
- InstallScript MSI*

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the SdSelectFolder function.
*
* SdSelectFolder is called to prompt the user to select a
* folder.
*
* Note: Before running this script, verify that the constant
*       DEF_FOLDER references an existing folder on the

```

```

*      target computer.
*
\*-----*/

#define DEF_FOLDER "Startup"

#include "Ifx.h"

function OnBegin()
    STRING  szTitle, szMsg, svDefGroup;
begin

    // Set up parameters for call to SdSelectFolder.
    szTitle  = "SdSelectFolder Example";
    szMsg    = "";
    svDefGroup = DEF_FOLDER;

    // Get user's folder selection.
    SdSelectFolder (szTitle, szMsg, svDefGroup);

end;

```

## SdSetupCompleteError



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdSetupCompleteError** function displays a dialog to inform the end user that the installation was interrupted before it could be completed. It also informs the end user that the product has not been installed and that the target system was not modified.

### Syntax

```
SdSetupCompleteError ( szTitle, szMsg1, szMsg2 );
```

## Parameters

**Table 82** • SdSetupCompleteError Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("InstallShield Wizard Completed"), pass a null string ("") in this parameter.
<b>szMsg1</b>	Specifies the message to display at the top of the dialog. To display the default instructions, which inform the end user that the installation has been interrupted and the product has not been installed, pass a null string ("") in this parameter.
<b>szMsg2</b>	Specifies the message to display at the bottom of the dialog. To display the default instructions ("Click Finish to exit the wizard"), pass a null string ("") in this parameter.

## Return Values

**Table 83** • SdSetupCompleteError Return Values

Return Value	Description
<b>NEXT (1)</b>	Indicates that the Finish button was clicked.

## Additional Information

If you write your own procedural script, that is, a program/endprogram block, you can use the **SdSetupCompleteError** function to display a dialog if the end user cancels the installation. If you are using the default script event model, you do not need to call this function.

Because **SdSetupCompleteError** indicates that the installation was interrupted and cannot be completed, the Back button is disabled.

The **SdSetupCompleteError** dialog is a variation of the **SdFinish** dialog. In a silent response file (.iss), the **SdSetupCompleteError** dialog is referred to as **SdFinish**.

## SdSetupCompleteError Example



**Project** • This information applies to the following project types:

- InstallScript
- InstallScript MSI

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdSetupCompleteError function.
```

```

*
* SdSetupCompleteError function displays a dialog to inform
* the end user that the installation was interrupted before it
* could be completed. It also informs the end user that the
* application has not been installed and that the target system
* was not modified. This function is only available for MSI
* based projects.
*
\*-----*/

#include "Ifx.h"

function OnBegin()
    STRING    szTitle, szMsg1, szMsg2;
begin
    // Set up parameters for call to SdSetupCompleteError.
    szTitle = "SdSetupCompleteError Example";
    szMsg1  = "";
    szMsg2  = "";
    SdSetupCompleteError ( szTitle, szMsg1, szMsg2 );

end;

```

## SdSetupType



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdSetupType** function displays a dialog that enables the end user to select one of the three standard setup types: Typical, Compact, or Custom. These setup options are displayed with standard description text. If you want to add other setup types or change the displayed setup type names or descriptions, call **SdSetupTypeEx** instead.

The dialog also displays a default destination path. A browse button launches a dialog that allows the end user to change the destination path, either by entering a new folder name or by selecting an existing folder from a list. If the end user enters the name of a folder that does not exist, this function automatically creates the specified folder. The fully qualified path of the specified folder is returned in **svDir**.



**Caution** • If the end user returns to the **SdSetupType** dialog after using a feature dialog to select and deselect features associated with the selected setup type, those choices are lost. This occurs because the **SdSetupType** function automatically resets the default feature selections for the selected setup type each time it is called.

### Syntax

```
SdSetupType ( szTitle, szMsg, svDir, nReserved );
```



## Parameters

**Table 84** • SdSetupType Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Setup Type"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies a message to display in the dialog. This text is considered a static control. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>svDir</b>	Specifies a default folder name. Returns the name of the folder selected by the end user.
<b>nReserved</b>	Reserved for future use. Pass 0 (zero) in this parameter.

## Return Values

**Table 85** • SdSetupType Return Values

Return Value	Description
<b>TYPICAL (301)</b>	Indicates that the user selected Typical Setup.
<b>COMPACT (302)</b>	Indicates that the user selected Compact Setup.
<b>CUSTOM (303)</b>	Indicates that the user selected Custom Setup.
<b>BACK (12)</b>	Indicates that the Back button was clicked.

## Additional Information

- In an InstallScript project, if you do not display a setup type dialog, your script must do one of the following:
  - Select a setup type.
  - Call a feature selection dialog function such as SdFeatureTree.
  - Directly select features.
- To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdSetupType Example



**Project** • This information applies to the following project types:

- *InstallScript*

- *InstallScript MSI*

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the functions SdSetupType.
 *
 * This function displays a dialog that enables the end user
 * to select one of the three standard setup types: Typical,
 * Compact, or Custom.
 *
 \*-----*/

#include "Ifx.h"

function OnBegin()
    STRING    szTitle, szMsg, svDir;
    NUMBER    nReserved, nResult;
begin

    // Display the SdSetupType dialog.
    szTitle = "SdSetupType Example";
    szMsg   = "";
    // Default destination folder displayed in dialog.
    svDir   = "C:\\Example";
    nReserved = 0;
    nResult = SdSetupType (szTitle, szMsg, svDir, nReserved);

    // Set TARGETDIR to the user selected destination folder.
    TARGETDIR = svDir;

    // Retrieve user selected setup type.
    switch(nResult)

        case CUSTOM: MessageBox("Custom setup type selected", 0);

        case TYPICAL: MessageBox("Typical setup type selected", 0);

        case COMPACT: MessageBox("Compact setup type selected", 0);

    endswitch;

end;

```

## SdSetupType2



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdSetupType2** function displays a dialog that enables the end user to select one of the two standard setup types: Typical or Custom. These setup options are displayed with standard description text. If you want to add other setup types or change the displayed setup type names or descriptions, call **SdSetupTypeEx** instead.

The dialog also displays a default destination path. A browse button launches a dialog that allows the end user to change the destination path, either by entering a new folder name or by selecting an existing folder from a list. If the end user enters the name of a folder that does not exist, this function automatically creates the specified folder. The fully qualified path of the specified folder is returned in `svDir`.



**Caution** • If the end user returns to the `SdSetupType2` dialog after using a feature dialog to select and deselect features associated with the selected setup type, those choices are lost. This occurs because the `SdSetupType2` function automatically resets the default feature selections for the selected setup type each time it is called.

## Syntax

```
SdSetupType2 ( szTitle, szMsg, svDir, nReserved );
```

## Parameters

**Table 86** • SdSetupType2 Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Setup Type"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies a message to display in the dialog. This text is considered a static control. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>svDir</b>	Specifies a default folder name. Returns the name of the folder selected by the end user.
<b>nReserved</b>	Reserved for future use. Pass 0 (zero) in this parameter.

## Return Values

**Table 87** • SdSetupType2 Return Values

Return Value	Description
<b>COMPLETE (304)</b>	Indicates that the user selected the Complete setup type.
<b>TYPICAL (301)</b>	Indicates that the user selected the Typical setup type.
<b>CUSTOM (303)</b>	Indicates that the user selected the Custom setup type.
<b>BACK (12)</b>	Indicates that the user clicked the Back button.

## Additional Information

- In an InstallScript project, if you do not display a setup type dialog, your script must do one of the following:
  - Select a setup type.

- Call a feature selection dialog function such as SdFeatureTree.
- Directly select features.
- To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdSetupType2 Example



**Project** • This information applies to the following project types:

- InstallScript
- InstallScript MSI

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the function SdSetupType2.
 *
 * SdSetupType2 is called to collect installation information
 * from the user.
 *
 \*-----*/

#include "ifx.h"

function OnBegin()
    STRING    szTitle, szMsg, svDir;
    STRING    svSetupType;
    NUMBER    nResult;
begin
    // Disable Back button
    Disable(BACKBUTTON);

    // Set up parameters for call to SdSetupType2.
    szTitle = "SdSetupType2 Example";
    szMsg    = "Choose the type of installation by clicking one of the buttons.";

    // Retrieve setup type and directory information.
    svDir = TARGETDIR;
    nResult = SdSetupType2 (szTitle , szMsg, svDir, 0);
    TARGETDIR = svDir;

    // Create a string that describes the selected setup type.
    switch (nResult)

        case COMPLETE:
            svSetupType = "COMPLETE: Application will be installed " +
                "with all options.";

        case CUSTOM:
            svSetupType = "CUSTOM: You select the options that you " +
                "want installed.";

        default:
            MessageBox ("Invalid setup type selection!", SEVERE);
```

```

                                abort;
        endswitch;

        MessageBox("Setup Type: " + svSetupType, 0);

    end;

```

## SdSetupTypeEx



**Project** • This information applies to *InstallScript MSI* projects.

The **SdSetupTypeEx** function displays a dialog that enables the end user to select the setup type when you specify setup types beyond Complete and Custom. The dialog displays the names of the setup types as you specified them in the Setup Types view.

### Syntax

```
SdSetupTypeEx ( szTitle, szMsg, szReserved, svSetupType, nReserved );
```

## Parameters

**Table 88 • SdSetupTypeEx Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Setup Type"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message to be displayed in the dialog. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>szReserved</b>	Pass a null string ("") in this parameter. No other value is allowed.
<b>svSetupType</b>	Specifies a default setup type and returns the setup type selected by the end user. The string returned in this parameter will match the name of the setup type as you specified it in the IDE.
<b>nReserved</b>	Pass zero in this parameter. No other value is allowed.

## Return Values

**Table 89 • SdSetupTypeEx Return Values**

Return Value	Description
<b>0</b>	Indicates that SdSetupTypeEx was successful.
<b>BACK (12)</b>	Indicates that the Back button was clicked.

## Additional Information

- In an InstallScript project, if you do not display a setup type dialog, your script must do one of the following:
  - Select a setup type
  - Call a feature selection dialog function such as SdFeatureTree.
  - Directly select features.
- To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdSetupTypeEx Example



**Project •** This information applies to InstallScript MSI projects.

```
/*-----*\
*
* InstallShield Example Script
*
```

```

* Demonstrates the SdSetupTypeEx function.
*
* The SdSetupTypeEx function displays a dialog that enables
* the end user to select the setup type when you specify setup
* types beyond Complete and Custom. The dialog
* displays the names of the setup types you create in the IDE's
* Setup Types view.
*
* This sample sets the default setup type in the SdSetupTypeEx
* dialog to "Complete".
*
\*-----*/

#include "ifx.h"

function OnBegin()
    STRING szTitle, szMsg, szReserved, svSetupType;
    NUMBER nReserved;
begin
    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Set up the variables to pass as parameters to SdSetupTypeEx.
    szTitle= "SdSetupTypeEx Sample";
    szMsg= "";
    szReserved = "";

    // Specifies a default setup type (as specified in the Setup Type
    // view) and returns the setup type selected by the end user
    svSetupType = "Complete";
    nReserved = 0;

    // Display the SdSetupTypeEx dialog.
    SdSetupTypeEx(szTitle, szMsg, szReserved, svSetupType, nReserved);

    // Display a MessageBox with the setup type selected by the end user
    MessageBox("Setup Type selected: " + svSetupType, 0);

end;

```

## SdShowAnyDialog



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdShowAnyDialog** function displays a custom or modified dialog. This function is recommended for advanced users only.

### Syntax

```
SdShowAnyDialog ( szTitle, szID, nID, nReserved );
```

## Parameters

**Table 90** • SdShowAnyDialog Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Welcome"), pass a null string ("") in this parameter.
<b>szID</b>	Specifies the string identifier that identifies the dialog. If this parameter contains a null string (""), SdShowAnyDialog uses the value in nID.
<b>nID</b>	Specifies the numeric value that identifies the dialog. If you entered a value in szID, this parameter is ignored.
<b>nReserved</b>	Pass zero in this parameter. No other value is allowed.

## Return Values

**Table 91** • SdShowAnyDialog Return Values

Return Value	Description
<b>NEXT (1)</b>	Indicates that the Next button was clicked.
<b>BACK (12)</b>	Indicates that the Back button was clicked.

## Additional Information

- To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.
- In order to use the SdShowAnyDialog function, you must know the ID of either the modified dialog in \_isres.dll or the custom dialog in \_isuser.dll that you want to display.
- If the dialog has only static controls, you do not need to modify the SdShowAnyDialog script file. However, if your dialog has any other controls, you must modify the Sdsadlg.rul file—located in the Script\Isrt\Src folder of your InstallShield program files folder—in order to process the feedback from the user.

## SdShowAnyDialog Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
 *
 * InstallShield Example Script
 *
```



```

* Demonstrates the SdShowAnyDialog function.
*
* SdShowAnyDialog is called twice, first with the ID of the
* Welcome dialog, then with the ID of the Finish dialog
* box.
*
\*-----*/

#define TITLE "SdShowAnyDialog Example"

#include "Ifx.h"

function OnBegin()
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Set up the product name so that it can be displayed
    // in place of the %P placeholder in Sd dialogs.
    SdProductName ("ExampApp");

    // Display the SdWelcome dialog.
    SdShowAnyDialog (TITLE, "", SD_NDLG_WELCOME, 0);

    // Display the SdFinish dialog.
    SdShowAnyDialog (TITLE, "", SD_NDLG_FINISH, 0);

end;

```

## SdShowDlgEdit1



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdShowDlgEdit1** function creates a general dialog that displays a message and one single-line edit field. You can specify a title for the dialog.

### Syntax

```
SdShowDlgEdit1 ( szTitle, szMsg, szField1, svEdit1 );
```

## Parameters

**Table 92** • SdShowDlgEdit1 Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Edit Data"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message to display in the dialog. To include in this message the product name set by a previous call to <a href="#">SdProductName</a> , insert the place holder %P anywhere in the message string. When the message is displayed, %P is replaced by the product name.
<b>szField1</b>	Specifies a field name to be displayed to the left of the edit field. The default field name is "Field 1:". To display the default name, pass a null string ("") in this parameter. The maximum number of characters that can be displayed is approximately 10. The actual maximum depends on the combined width of each character in the field name. If the field name exceeds the available space, it will be truncated on the right when the dialog is displayed.
<b>svEdit1</b>	Specifies an initial value for the edit field; returns the value of the edit field when the dialog is closed.

## Return Values

**Table 93** • SdShowDlgEdit1 Return Values

Return Value	Description
<b>NEXT (1)</b>	Indicates that the Next button was clicked.
<b>BACK (12)</b>	Indicates that the Back button was clicked.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdShowDlgEdit1 Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*
 *
 * InstallShield Example Script
```

```

*
* Demonstrates the SdShowDlgEdit1 function.
*
* This example script calls SdShowDlgEdit1 to obtain the name
* of a folder, which is then displayed in a message box.
*
\*-----*/

#include "Ifx.h"

function OnBegin()
    STRING szTitle, szMsg, szField1, svEdit1;
begin
    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Set up parameters for call to SdShowDlgEdit1.
    szTitle = "SdShowDlgEdit1 Example";
    szMsg = "Please choose a folder for YourApp:";
    szField1 = "Target:";
    svEdit1 = "C:\\Example\\Target\\YourApp";

    // Get a target folder name from the user.
    if (SdShowDlgEdit1 (szTitle, szMsg, szField1, svEdit1) < 0) then
        // Report an error.
        MessageBox ("SdShowDlgEdit1 failed.", SEVERE);

    else
        // Display svEdit1 string variable.
        sprintfBox (INFORMATION, szTitle, "You selected %s", svEdit1);

    endif;

end;

```

## SdShowDlgEdit2



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdShowDlgEdit2** function creates a general dialog that displays a message and two single-line edit fields. You can specify a title for the dialog.

### Syntax

```
SdShowDlgEdit2 ( szTitle, szMsg, szField1, szField2, svEdit1, svEdit2 );
```

## Parameters

**Table 94 • SdShowDlgEdit2 Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. If you pass a null string ("") in this parameter, the default title ("Edit Data") is displayed.
<b>szMsg</b>	Specifies the message to display in the dialog. To include in this message the product name set by a previous call to <a href="#">SdProductName</a> , insert the place holder %P anywhere in the message string. When the message is displayed, %P is replaced by the product name.
<b>szField1</b>	Specifies a field name to be displayed to the left of the first edit field. The default field name is "Field 1:" ; to display the default name, pass a null string ("") in this parameter. The maximum number of characters that can be displayed is approximately 10. The actual maximum depends on the combined width of each character in the field name. If the field name exceeds the available space, it will be truncated on the right when the dialog is displayed.
<b>szField2</b>	Specifies a field name for the second edit field. "Field 2:" is the default.
<b>svEdit1</b>	Specifies an initial value for the first edit field; returns the value of the first edit field when the dialog is closed.
<b>svEdit2</b>	Specifies an initial value for the second edit field; returns the value of the second edit field when the dialog is closed.

## Return Values

**Table 95 • SdShowDlgEdit2 Return Values**

Return Value	Description
<b>NEXT (1)</b>	Indicates that the Next button was clicked.
<b>BACK (12)</b>	Indicates that the Back button was clicked.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdShowDlgEdit2 Example



**Project •** This information applies to the following project types:

- *InstallScript*

- *InstallScript MSI*

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdShowDlgEdit2 function.
 *
 * This script displays a dialog that prompts the user to
 * specify a source folder and a target folder. It then displays
 * the user's selections in a message box.
 *
 \*-----*/

#include "Ifx.h"

function OnBegin()
    STRING szTitle, szMsg, szField1, szField2, svEdit1, svEdit2;
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Set up parameters for call to SdShowDlgEdit2.
    szTitle = "SdShowDlgEdit2 Example";
    szMsg = "All files within the Source directory will be copied into " +
        "the Target directory.";
    szField1 = "Source:";
    szField2 = "Target:";
    svEdit1 = "C:\\Example\\Source";
    svEdit2 = "C:\\Example\\Target";

    // Get source and target folder names.
    if (SdShowDlgEdit2 (szTitle, szMsg, szField1, szField2,
        svEdit1, svEdit2) < 0) then
        // Report an error.
        MessageBox ("SdShowDlgEdit2 failed.", SEVERE);

    else
        // Display the user's selections.
        sprintfBox (INFORMATION, szTitle, "svEdit1: %s\nsvEdit2: %s",
            svEdit1, svEdit2);
    endif;

end;

```

## SdShowDlgEdit3



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdShowDlgEdit3** function creates a general dialog that displays a message and three single-line edit fields. You can specify a title for the dialog in szTitle.

## Syntax

```
SdShowDlgEdit3 ( szTitle, szMsg, szField1, szField2, szField3, svEdit1, svEdit2, svEdit3 );
```

## Parameters

**Table 96 • SdShowDlgEdit3 Parameters**

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. If you pass a null string (""), the default title ("Edit Data") is displayed.
<b>szMsg</b>	Specifies the message to display in the dialog. To include in this message the product name set by a previous call to <a href="#">SdProductName</a> , insert the place holder %P anywhere in the message string. When the message is displayed, %P is replaced by the product name.
<b>szField1</b>	Specifies a field name to be displayed to the left of the first edit field. The default field name is "Field 1:". To display the default name, pass a null string (""), in this parameter. The maximum number of characters that can be displayed is approximately 10. The actual maximum depends on the combined width of the characters in the field name. If the field name is too large for the available space, it will be truncated on the right when the dialog is displayed.
<b>szField2</b>	Specifies a field name for the second edit field. "Field 2:" is the default.
<b>szField3</b>	Specifies a field name for the third edit field. "Field 3:" is the default.
<b>svEdit1</b>	Specifies an initial value for the first edit field; returns the value of the first edit field when the dialog is closed.
<b>svEdit2</b>	Specifies an initial value for the second edit field; returns the value of the second edit field when the dialog is closed.
<b>svEdit3</b>	Specifies an initial value for the third edit field; returns the value of the third edit field when the dialog is closed.

## Return Values

**Table 97 • SdShowDlgEdit3 Return Values**

Return Value	Description
<b>NEXT (1)</b>	Indicates that the Next button was clicked.
<b>BACK (12)</b>	Indicates that the Back button was clicked.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdShowDlgEdit3 Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdShowDlgEdit3 function.
 *
 * This script calls SdShowDlgEdit3 to get three folder names
 * from the user. These are then displayed in a message box.
 *
 \*-----*/

#include "Ifx.h"

function OnBegin()
    STRING szTitle, szMsg, szField1, szField2, szField3, svEdit1, svEdit2;
    STRING svEdit3;
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Set up parameters for call to SdShowDlgEdit3.
    szTitle   = "SdShowDlgEdit3 Example";
    szMsg      = "Please choose up to two valid directories in Backup: or " +
                "Alternate: to back up the source directory.\n\n";
    szField1   = "Directory";
    szField2   = "Backup";
    szField3   = "Alternate";
    svEdit1    = "C:\\YourApp";
    svEdit2    = "C:\\Backup1";
    svEdit3    = "C:\\Backup2";

    // Get three folder names from the user.
    if (SdShowDlgEdit3 (szTitle, szMsg, szField1, szField2, szField3,
                        svEdit1, svEdit2, svEdit3) < 0) then
        // Report an error.
        MessageBox ("SdShowDlgEdit3 failed.", SEVERE);
    else
        // Display the folder names specified by the user.
        szMsg = "svEdit1: %s\n\svEdit2: %s\n\svEdit3: %s";
        sprintfBox (INFORMATION, szTitle, szMsg, svEdit1, svEdit2, svEdit3);
    endif;
end;
```



# SdShowFileMods



**Project** • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

The **SdShowFileMods** function creates a dialog that displays changes you want to make to a file. These choices are available:

- Make changes to the target file.
- Make changes to the alternate file, which is a copy of the target file, but incorporates changes.
- Do not make any changes. SdShowFileMods does not make changes to a file. You must write those changes into your script using the appropriate file functions.

## Syntax

```
SdShowFileMods ( szTitle, szMsg, szTargetFile, szAltFile, listChanges, nvSelection );
```

## Parameters

**Table 98** • SdShowFileMods Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Modifying File"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message to display in the dialog. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>szTargetFile</b>	Specifies the name of the file to modify. This will be displayed with the first radio button.
<b>szAltFile</b>	Specifies an alternate name to give the file if the end user decides to make the changes. This will be displayed with the second radio button. To use the name of file specified in szTargetFile with the extension .bak, pass a null string ("") in this parameter.
<b>listChanges</b>	Specifies the name of a string list that contains the list of changes to make to the file. This list is placed in a multi-line edit field that allows the end user to select the changes to be implemented.
<b>nvSelection</b>	Returns the ID of the button selected by the end user: <ul style="list-style-type: none"> <li>● <b>101</b>—"Let Setup modify the &lt;szTargetFile&gt; file."</li> <li>● <b>102</b>—"Save the required changes to &lt;szAltFile&gt; file."</li> <li>● <b>103</b>—"Do not make any changes."</li> </ul>

## Return Values

**Table 99** • SdShowFileMods Return Values

Return Value	Description
<b>NEXT (1)</b>	Indicates that the Next button was clicked.
<b>BACK (12)</b>	Indicates that the Back button was clicked.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdShowFileMods Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdShowFileMods function.
 *
 * This script displays a list of changes to make to a specified
 * file. The user may elect to make those changes, to save the
 * changes to a new file, or not to make or save the changes.
 *
 * Note: SdShowFileMods does not itself modify or create files.
 *       It simply obtains the user's choice.
 *
\*-----*/

#include "Ifx.h"

function OnBegin()
    STRING szTitle, szMsg, szTargetFile, szAltFile;
    NUMBER nvSelection;
    LIST listID;
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Set up variables for call to SdShowFileMods.
    szTitle = "SdShowFileMods Example";

    szMsg = "Choose what option to take";
    szTargetFile = "Example.txt";
    szAltFile = "Example.new";

    // Create a list to hold file modification details.
    listID = ListCreate (STRINGLIST);

    // Add file modification details to the list.
    ListAddString (listID, "PATH = C:\\Example", AFTER);
    ListAddString (listID, "FILES = 40", AFTER);

    // Present options and get the user's choice.
    SdShowFileMods (szTitle, szMsg, szTargetFile, szAltFile,
                    listID, nvSelection);

    // Handle the user's selection.
    switch(nvSelection)

        case 101:
            sprintfBox (INFORMATION, szTitle, "Setup modified the %s file.",
                        szTargetFile);

        case 102:
            sprintfBox (INFORMATION, szTitle, "The required changes were saved " +
                        "to the %s file.", szAltFile);

        case 103:

```

```
        SprintfBox (INFORMATION, szTitle, "No changes were made.");  
  
    endswitch;  
  
end;
```

## SdShowInfoList



---

**Project** • *This information applies to the following project types:*

- *InstallScript*
- *InstallScript MSI*

The **SdShowInfoList** function creates a dialog that displays a list of scrollable messages. SdShowInfoList can display lists of up to approximately 57,200 characters.

### Syntax

```
SdShowInfoList ( szTitle, szMsg, listID );
```

## Parameters

**Table 100** • SdShowInfoList Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Information"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message to display on one line above the information box. If the message is too long to fit on one line, it is truncated on the right. If the message includes a newline escape sequence (\n), the text following that escape sequence is not displayed. To display the default message ("Text"), pass a null string ("") in this parameter.
<b>listID</b>	Specifies the list of messages to display in the dialog. All messages that appear in the dialog are read only.

## Return Values

**Table 101** • SdShowInfoList Return Values

Return Value	Description
<b>NEXT (1)</b>	Indicates that the Next button was clicked.
<b>BACK (12)</b>	Indicates that the Back button was clicked.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdShowInfoList Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the SdShowInfoList function.
*
* This script calls GetSystemInfo to retrieve information about
* the user's system. The ListAdd function is used to add the
* information to a string list, which is displayed by
* calling SdShowInfoList function.
```

```

*
\*-----*/

#include "Ifx.h"

function OnBegin()
    STRING  szTitle, szMsg, svReturn, szInfo;
    NUMBER  nvReturn;
    LIST    listInfo;
begin

    // Create a list to hold system information.
    listInfo = ListCreate (STRINGLIST);

    // Check if the system has a CD-ROM drive.
    GetSystemInfo (CDROM, nvReturn, svReturn);

    if (nvReturn = TRUE) then
        szInfo = "Your machine has a CD-ROM Drive.";
    else
        szInfo = "Your machine does not have a CD-ROM drive.";
    endif;

    // Add the CD-ROM info to the list.
    ListAddString (listInfo, szInfo, AFTER);

    // Check the time on the system.
    GetSystemInfo (TIME, nvReturn, svReturn);
    Sprintf (szInfo, "The time now is %s.", svReturn);

    // Add the time to the list.
    ListAddString (listInfo, szInfo, AFTER);

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Set up title and message parameters for call to SdShowInfoList.
    szTitle = "SdShowInfoList Example";
    szMsg    = "Following is some information related to your system:";

    // Display the information.
    SdShowInfoList (szTitle, szMsg, listInfo);

end;

```

## SdShowMsg



**Project** • This information applies to the following project types:

- InstallScript
- InstallScript MSI

The **SdShowMsg** function provides a simple way to display an informative message that remains on screen while script processing continues.

The **SdShowMsg** function opens or closes a small modeless window that displays the message specified by `szMsg`. When `bShow` is `TRUE`, the window is opened, the message is displayed in the window, and processing continues with the next statement in the script. Note that the **SdShowMsg** window is positioned at the center of the setup window. When `bShow` is `FALSE`, `szMsg` is ignored and the **SdShowMsg** window is closed.



**Note** • When the **SdShowMsg** window is open, subsequent calls to **SdShowMsg** with `TRUE` in the second parameter are ignored. To change the message, you must first close the window by calling **SdShowMsg** with `FALSE` in the second parameter and then call **SdShowMsg** again with the new message in `szMsg` and `TRUE` in the second parameter.

## Syntax

```
SdShowMsg ( szMsg, bShow );
```

## Parameters

**Table 102** • SdShowMsg Parameters

Parameter	Description
<b>szMsg</b>	<p>Specifies the message to display in the window. To display the default message ("Setup is searching for installed features"), pass a null string ("") in this parameter. This parameter is ignored when <code>bShow</code> is <code>FALSE</code>.</p> <p><b>SdShowMsg</b> is designed to display a message on a single line. Do not embed newline characters (\n) in <code>szMsg</code>.</p> <div data-bbox="626 1073 667 1119" data-label="Image"> </div> <p><b>Note</b> • The <b>SdShowMsg</b> window is sized horizontally to display the value of <code>szMsg</code> on a single line. If the length of the message exceeds the maximum width of the window, the message is truncated to fit the window.</p>
<b>bShow</b>	<p>Specifies whether to open or close the window. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li><b>TRUE</b>—Opens the window if it is not already open.</li> <li><b>FALSE</b>—Closes the window if it is open.</li> </ul>

## Return Values

This function always returns 0.

## Additional Information

- The dialog that is displayed by the **SdShowMsg** function cannot be displayed with a skin; it appears the same regardless of whether you have specified a skin.
- The Dialog Editor does not support dialogs, such as **SdShowMsg**, that do not have a title bar. Therefore, this dialog is not displayed in the Dialogs view as one of the dialogs that you can edit. To customize this dialog, use the **SdShowMsg** call.

## SdShowMsg Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SdShowMsg dialog.
 *
 * SdShowMsg is called to display a message for three seconds.
 * It's then called again to remove the message from the screen.
 * A final call to SdShowMsg displays another message.
 *
 \*-----*/

#include "Ifx.h"

function OnBegin()
    STRING szMsg;
begin

    // Display a message on the screen.
    szMsg = "This message will appear for three seconds.";
    SdShowMsg (szMsg, TRUE);

    // Delay for 3 seconds.
    Delay (3);

    // Remove message from screen.
    SdShowMsg (szMsg, FALSE);

    // Display another message on the screen.
    szMsg = "This is another message which will " +
        "appear for a mere three seconds.";
    SdShowMsg (szMsg, TRUE);
    Delay(3);

    // Remove message from screen.
    SdShowMsg (szMsg, FALSE);

end;

// Source file:Is5fn157.rul
```

## SdStartCopy



**Project** • This information applies to the following project types:

- *InstallScript*



- *InstallScript MSI*

The **SdStartCopy** function creates a multi-line edit field displaying the settings and selections made during the installation. The end user can click the Back button on the dialog to return to previous dialogs in order to change settings as required. Call SdStartCopy after retrieving the selections from the user, but before beginning the file transfer process.

Use a string list to collect the information obtained during the installation. You then pass the string list to SdStartCopy in the parameter listData. SdStartCopy displays the list and allows the user to verify that the information is correct before continuing with the file transfer process.

## Syntax

SdStartCopy (szTitle, szMsg, listData);

## Parameters

**Table 103** • SdStartCopy Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Start Copying Files"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message to display in the static text field above the multi-line edit field. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>listData</b>	Specifies a string list of information retrieved from the end user. SdStartCopy automatically places each element into the multi-line edit field. If listData has not been initialized by a call to <a href="#">ListCreate</a> , the multi-line edit field is hidden and only the static text field is visible.

## Return Values

**Table 104** • SdStartCopy Return Values

Return Value	Description
<b>NEXT (1)</b>	Indicates that the user selected the Next button.
<b>BACK (12)</b>	Indicates that the user selected the Back button.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdStartCopy Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the function SdStartCopy, with the use of
 * SdRegisterUserEx and SdSetupType.
 *
 * SdRegisterUserEx and SdSetupType are called to collect
 * installation information from the user. The information is
 * stored in a list and displayed by SdStartCopy.
 *
 \*-----*/

#include "Ifx.h"

function OnBegin()
    STRING  szTitle, szMsg, svName, svCompany, svSerial, svDir, svSetupType;
    LIST    listData;
    NUMBER  nResult;
begin

start:
    // Create the list.
    listData = ListCreate (STRINGLIST);

    // Set up message parameter for call to SdRegisterUserEx.
    szMsg = "Please enter your name, company, and serial no.";

    // Retrieve registration information.
    SdRegisterUserEx ("Registration", szMsg, svName, svCompany, svSerial);

    // Add the information to the list.
    ListAddString (listData, "User Information: ", AFTER);
    ListAddString (listData, "          " + svName, AFTER);
    ListAddString (listData, "          " + svCompany, AFTER);
    ListAddString (listData, "          " + svSerial, AFTER);
    ListAddString (listData, "", AFTER);

SetupTypeLabel:
    // Set up parameters for call to SdSetupType.
    szMsg = "Choose the type of installation by clicking one of the buttons.";
    svDir = TARGETDIR;

    // Retrieve setup type and directory information.
    nResult = SdSetupType("Select Setup Type", szMsg, svDir, 0);

    // Create a string that describes the selected setup type.
    switch (nResult)
        case TYPICAL:
```

```

        svSetupType = "TYPICAL: Application will be installed " +
            "with the most common options.";
    case COMPACT:
        svSetupType = "COMPACT: Application will be installed " +
            "with the minimum required options.";
    case CUSTOM:
        svSetupType = "CUSTOM: You select the options that you " +
            "want installed.";
    case BACK:
        goto start;
    default:
        MessageBox ("Invalid setup type selection!", SEVERE);
        abort;
endswitch;

// Add the setup type information to the list.
ListAddString(listData, "Setup Type:", AFTER);
ListAddString(listData, "          " + svSetupType, AFTER);
ListAddString(listData, "", AFTER);
ListAddString(listData, "Destination Directory:", AFTER);
ListAddString(listData, "          " + svDir, AFTER);

// Set up title and message parameters for call to SdStartCopy.
szTitle = "SdStartCopy Example";
szMsg  = "Setup has enough information to begin the file-transfer\n" +
        "operation.  If you want to review or change any of the\n" +
        "settings, click Back.  If you are satisfied with the\n" +
        "settings, click Next to begin copying files.";

// Call SdStartCopy to display user selections.
nResult = SdStartCopy (szTitle, szMsg, listData);

// Handle the user's exit from the SdStartCopy dialog.
switch(nResult)
    case NEXT:
        MessageBox ("SdStartCopy successful.", INFORMATION);
    case BACK:
        goto SetupTypeLabel;
    default:
        MessageBox ("SdStartCopy failed.", SEVERE);
endswitch;

end;

```

## SdStartCopy2



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdStartCopy2** function informs the user that the file transfer process is about to begin. The user can click the Back button to return to previous dialogs in order to change settings as required. Call SdStartCopy2 after retrieving the selections from the user, but before beginning the file transfer process.

## Syntax

```
SdStartCopy2 ( szTitle, szMsg );
```

## Parameters

**Table 105** • SdStartCopy2 Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Ready to Install the Program"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message to display in the static text field. To display the default instructions for this dialog, pass a null string ("") in this parameter.

## Return Values

**Table 106** • SdStartCopy2 Return Values

Return Value	Description
<b>NEXT</b>	Indicates that the user selected the Install button.
<b>BACK</b>	Indicates that the user selected the Back button.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the dialog could not be displayed.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdStartCopy2 Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the functions SdProductName, SdWelcome,
 * SdStartCopy2, and SdFinish.
 *
 * First, SdProductName is called to set the product name to
 * substitute for the %P place holder, which is embedded in the
 * message strings passed to SdWelcome and SdFinish.
 *
```

```

* Next, SdWelcome is called to display a welcome message and
* then SdStartCopy2 is called. Finally, a call to SdFinish
* completes the setup process by giving the user final options.
*
* Note: Before running this script, set the preprocessor
*       constants so that they reference the fully-qualified
*       names of the Windows Notepad executable and a valid
*       text file on the target system.
*
\*-----*/

//Assign the name of a text file to READMEFILE

#define NOTEPAD "C:\\Windows\\Notepad.exe"
#define READMEFILE ""

    STRING    szProductName, szTitle, szMsg, szFeatures;
    STRING    szMsg1, szMsg2, szOpt1, szOpt2;
    BOOL      bvOpt1, bvOpt2;
    NUMBER    nReturn;

#include "ifx.h"

function OnBegin()
begin

    // Set the product name to substitute for the %P place holder.
    szProductName = "My Application";
    SdProductName (szProductName);

SdWelcomeLabel:
    szTitle = "SdWelcome Example";

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Display the SdWelcome dialog. The null string in parameter
    // two specifies the default message, which uses the %P place holder.
    SdWelcome (szTitle, "");

    // Enable the Back button in setup dialogs.
    Enable (BACKBUTTON);

SdStartCopy2Label:
    szTitle = "SdStartCopy2 Example";
    // Display the SdStartCopy2 dialog.
    if (SdStartCopy2 (szTitle, szMsg) = BACK) then
        goto SdWelcomeLabel;
    endif;

    // The %P place holder is embedded in several of the string
    // parameters that will be passed to SdFinish.
    szTitle = "SdFinish Example";
    szMsg1 = "%P Setup is almost complete.\n" +
        "Choose the options you want below.";
    szMsg2 = "Click Finish to complete %P Setup.";
    szOpt1 = "I would like to view the README file.";

```

```

szOpt2 = "I would like to launch %P.";

// Display the SdFinish dialog.
SdFinish (szTitle, szMsg1, szMsg2, szOpt1, szOpt2, bvOpt1, bvOpt2);

if (bvOpt1) then
    // Display the read me file.
    LaunchAppAndWait (NOTEPAD, READMEFILE, LAAW_OPTION_WAIT);
endif;

if (bvOpt2) then
    // Because this example does not actually install an
    // application, a message box is displayed in place of
    // the call to LaunchApp that would normally be made here,
    // for example:
    // LaunchApp (TARGETDIR^PROGRAMEXECUTABLE,"");

    sprintfBox (INFORMATION, szTitle, "Launch %s here.", szProductName);
endif;

end;

```

## SdSubstituteProductInfo

The **SdSubstituteProductInfo** function replaces any occurrences of the %P, %VS, and %VI placeholders in svString with the values of the system variables IFX\_PRODUCT\_DISPLAY\_NAME, IFX\_PRODUCT\_DISPLAY\_VERSION, and IFX\_INSTALLED\_DISPLAY\_VERSION. You can use this function before calling a function, such as MessageBox, that does not automatically perform this replacement before displaying strings.

### Syntax

```
SdSubstituteProductInfo ( svString );
```

### Parameters

**Table 107** • SdSubstituteProductInfo Parameters

Parameter	Description
<b>svString</b>	Specifies the string containing placeholders; returns the string with the placeholders replaced.

### Return Values

This function does not return a value.

## SdWelcome



**Project** • This information applies to the following project types:

- *InstallScript*

- *InstallScript MSI*

The **SdWelcome** function creates a dialog that displays a welcome message to the end user.

## Syntax

```
SdWelcome ( szTitle, szMsg );
```

## Parameters

**Table 108** • SdWelcome Parameters

Parameter	Description
<b>szTitle</b>	Specifies the text to display in the title of the dialog. To display the default title ("Welcome"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message to display in the Welcome dialog. To include in this message the product name set by a previous call to <a href="#">SdProductName</a> , insert the place holder %P anywhere in the message string. When the message is displayed, %P is replaced by the product name. To display the default welcome message ("Welcome to the %P Setup program. This program will install %P on your computer"), pass a null string ("") in this parameter.

## Return Values

**Table 109** • SdWelcome Return Values

Return Value	Description
<b>NEXT (1)</b>	Indicates that the Next button was clicked.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdWelcome Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the function SdWelcome.
 *
```

```

* The SdWelcome function displays a dialog that welcomes
* the end user.
*
\*-----*/

#include "ifx.h"

function OnBegin()
    STRING  szTitle, szProductName;
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Set the product name to substitute for the %P place holder.
    szProductName = "My Application";
    SdProductName (szProductName);

    szTitle = "SdWelcome Example";

    // Display the SdWelcome dialog. The null string in parameter
    // two specifies the default message, which uses the %P place holder.
    SdWelcome (szTitle, "");

end;

```

## SdWelcomeMaint



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SdWelcomeMaint** function displays a dialog that is intended for use at the beginning of a maintenance setup (that is, the re-running of a setup that has already been run). The dialog contains Modify, Repair, and Remove option buttons.

### Syntax

```
SdWelcomeMaint (szTitle, szMsg, nType);
```



## Parameters

**Table 110** • SdWelcomeMaint Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title ("Welcome"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message to display in the dialog. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>nType</b>	Specifies which option button is the default selection. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>MODIFY</b>—The Modify button is the default selection.</li> <li>• <b>REPAIR</b>—The Repair button is the default selection.</li> <li>• <b>REMOVEALL</b>—The Remove button is the default selection.</li> </ul>

## Return Values

**Table 111** • SdWelcomeMaint Return Values

Return Value	Description
<b>MODIFY (301)</b>	Indicates that the Modify button was selected when the end user clicked the Next button.
<b>REPAIR (302)</b>	Indicates that the Repair button was selected when the end user clicked the Next button.
<b>REMOVEALL (303)</b>	Indicates that the Remove button was selected when the end user clicked the Next button.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SdWelcomeMaint Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
*
* InstallShield Example Script
```

```

*
* Demonstrates the function SdWelcomeMaint.
*
* The SdWelcomeMaint function displays a dialog that is intended
* for use at the beginning of a maintenance setup (the re-running
* of a setup that has already been run). The dialog contains
* Modify, Repair, and Remove option buttons.
*
\*-----*/

#include "ifx.h"

function OnBegin()
    STRING  szTitle, szProductName;
    NUMBER  nType, nReturn;
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Set the product name to substitute for the %P place holder.
    szProductName = "My Application";
    SdProductName (szProductName);

    szTitle = "SdWelcomeMaint Example";
    //Specifies which option button is the default selection.
    nType = REMOVEALL;

    // Display the SdWelcomeMaint dialog. The null string in
    // parameter two specifies the default message, which uses
    // the %P place holder.
    nReturn = SdWelcomeMaint (szTitle, "", nType);

    switch(nReturn)

    case MODIFY: MessageBox("SdWelcomeMaint selection: Modify", 0);

    case REPAIR: MessageBox("SdWelcomeMaint selection: Repair", 0);

    case REMOVEALL: MessageBox("SdWelcomeMaint selection: Remove", 0);

    endswitch;

end;

```

## SeekBytes

The **SeekBytes** function repositions the file pointer within an open binary file. You can move the file pointer a specific number of bytes relative to its current position or relative to the beginning or end of the file.



**Note** • Before calling *SeekBytes*, you must open the file (which can be a file on the Internet) in binary mode by calling *OpenFileMode* and *OpenFile*. When you are finished writing bytes to the file, call *CloseFile* to close the file.

## Syntax

```
SeekBytes ( nFileHandle, nBytes, nPosition );
```

## Parameters

**Table 112 • SeekBytes Parameters**

Parameter	Description
<b>nFileHandle</b>	Specifies the file handle of a file that has been opened in binary mode.
<b>nBytes</b>	Specifies the number of bytes to move the file pointer relative to the position specified by nPosition. If nBytes is a positive number, the file pointer is moved toward the end of the file. If nBytes is a negative number, the file pointer is moved toward the beginning of the file.
<b>nPosition</b>	Specifies the location in the file from which to move the pointer nBytes. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>FILE_BIN_CUR</b>—Moves the pointer nBytes from its current position.</li> <li>• <b>FILE_BIN_END</b>—Moves the pointer nBytes from the end of the file.</li> <li>• <b>FILE_BIN_START</b>—Moves the pointer nBytes from the beginning of the file.</li> </ul>

## Return Values

**Table 113 • SeekBytes Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully repositioned the pointer.
<b>&lt; 0</b>	Indicates that the function was unable to reposition the pointer.

## SeekBytes Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the ReadBytes and SeekBytes functions.
 *
 * SeekBytes is called to position a file pointer to a
 * specific location in a file that has been opened in binary
 * mode. ReadBytes then reads a specific number of bytes,
 * starting at this location. The bytes are read into a string,
 * which is then displayed in a message box.
 *
 * Note: The defined constants EXAMPLE_DIR and EXAMPLE_BIN must
 *       be set to an existing directory and file on the target
 *       system.
 *
 \*-----*/

#define EXAMPLE_DIR "C:\\\"
#define EXAMPLE_BIN "Example.bin"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_SeekBytes(HWND);

function ExFn_SeekBytes(hMSI)
    STRING svString;
    NUMBER nvFileHandle;
begin

    // Set the file mode to read/write.
    OpenFileMode (FILE_MODE_BINARY);

    // Open a binary file.
    if (OpenFile (nvFileHandle, EXAMPLE_DIR, EXAMPLE_BIN) < 0) then

        // Report an error; then abort.
        SprintfBox (SEVERE, "CopyBytes Example", "Could not open %s.",
                    EXAMPLE_BIN);

        abort;

    endif;

    // Set the file pointer to the 16th byte in the file.
    SeekBytes (nvFileHandle, 15, FILE_BIN_START);

    // Read the next twenty-eight bytes into svString.
    if (ReadBytes (nvFileHandle, svString, 0, 28) < 0) then

        // Report an error.
        MessageBox ("ReadBytes failed.", SEVERE);
    else

        // Display the string.
        SprintfBox (INFORMATION, "ReadBytes Example", "Bytes read: %s",
svString);

    endif;

```

```
// Close the file.
CloseFile (nvFileHandle);

end;
```

## SelectDir



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SelectDir** function has been superseded by the **SelectDirEx** function. You should use the **SelectDirEx** function when creating new installations. You can call the **SelectDirEx** function with `nFlags` set to `BIF_RETURNONLYFSDIRS | BIF_EDITBOX` to achieve the same result as calling the **SelectDir** function with `bCreate` set to `FALSE`.

The **SelectDir** function displays a dialog that lets the end user specify the folder into which the application will be installed. The end user can enter a fully qualified folder name or select an existing folder from a list. If the end user enters an invalid folder name or an unqualified folder name, a message box is displayed to prompt the end user to enter a valid name. The fully qualified name of the selected folder is returned in `svDir`.

If the specified folder does not exist and the parameter `bCreate` is `TRUE`, **SelectDir** automatically creates the specified folder. If the parameter `bCreate` is set to `FALSE` and a non-existent folder is selected, the end user is not informed, and **SelectDir** does not create it. In this case, it is up to you to handle the selection contained in `svDir`.



**Note** • **SelectDir** is called automatically when the end user clicks the *Browse* button in the dialogs that are presented by **AskDestPath**, **SdAskDestPath**, and other *InstallScript* functions that obtain a folder name.

Windows displays this dialog; therefore, the installation cannot change the text of the buttons on the dialog. Windows displays the button text—"Yes" and "No" on English-based systems—in the language of the operating system; no manual localization of this text is required. If you need to display a more flexible dialog, call a Windows API function directly or use a custom dialog.

### Syntax

```
SelectDir ( szTitle, szMsg, svDir, bCreate );
```

## Parameters

**Table 114** • SelectDir Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of this dialog. To display the default title ("Choose Folder"), pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message you want to display in this dialog. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>svDir</b>	Specifies the name of a folder that will appear as the default selection. Returns the fully qualified name of the folder selected by the end user.
<b>bCreate</b>	Specifies whether or not you want InstallShield to create the specified folder if it does not already exist. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>TRUE</b>—Indicates that the folder should be created if it does not exist.</li> <li>• <b>FALSE</b>—Indicates that the folder should not be created if it does not exist.</li> </ul>

## Return Values

If bCreate is FALSE, this function returns one of the following values:

**Table 115** • SelectDir Return Values when bCreate is FALSE

Return Value	Description
<b>IDOK (1)</b>	Indicates that the OK button was selected.
<b>IDCANCEL (2)</b>	Indicates that the CANCEL button was selected.
<b>&lt; 0</b>	Indicates that the function was unable to display the dialog.

If bCreate is TRUE, this function returns one of the following values:

**Table 116** • SelectDir Return Values when bCreate is TRUE

Return Value	Description
<b>IDCANCEL (2)</b>	Indicates that the CANCEL button was selected.
<b>0</b>	Indicates that the OK button was selected and the function was able to create the folder specified, if necessary.
<b>&lt; 0</b>	Indicates that the function failed to display the dialog and/or create the specified folder.

## Additional Information

- The dialog that is displayed by this function cannot be displayed with a skin; it appears the same, regardless of whether you have specified a skin.
- In early versions of InstallShield Professional, when bCreate was set to TRUE and the end user selected a folder that did not exist, a confirmation message box was displayed asking whether the folder should be created. This message box proved to be confusing to many end users, so it has been removed from InstallShield.

## SelectDir Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SelectDir function.
 *
 * This script calls SelectDir to display a dialog that
 * allows the user to specify a folder. If the specified folder
 * does not exist, it is not created. Instead, an error message
 * is displayed and the SelectDir dialog is displayed again.
 *
 \*-----*/

#define TITLE_TEXT "SelectDir Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_SelectDir(HWND);

function ExFn_SelectDir(hMSI)
    STRING  szTitle, szMsg, svDir;
    BOOL    bCreate, bFolderExists;
    NUMBER  nResult;
begin

    // Loop until user cancels or selects an existing folder.
    repeat

        // Set a default folder for the SelectDir dialog.
        svDir = INSTALLDIR;

        // Set the message to display in the SelectDir dialog.
        szMsg  = "Please choose an existing folder.";

        // Get an existing folder name from the user. The fourth
        // parameter indicates that a non-existing folder should
        // not be created.
        nResult = (SelectDir (TITLE_TEXT, szMsg, svDir, FALSE) < 0) ;
```

```

    if nResult = 0 then

        // Determine whether the folder exists.
        bFolderExists = ExistsDir (svDir);

        if bFolderExists = NOTEXISTS then
            // The folder does not exist. Ask user to select again.
            szMsg = "%s does not exist.\nPlease choose an existing folder.";
            sprintfBox (WARNING, szTitle, szMsg, svDir);
        endif;

    endif;

until (nResult = CANCEL) || (bFolderExists = EXISTS);

if (bFolderExists = EXISTS) then

    // Display the name of the selected folder.
    sprintfBox (INFORMATION, szTitle, "You selected %s.", svDir);

endif;

end;

```

## SelectDirEx



**Project** • This information applies to the following project types:

- InstallScript
- InstallScript MSI

The **SelectDirEx** function displays a dialog that enables the end user to select a folder into which the application will be installed. An edit box can also be displayed to enable the end user to specify a new folder.

This function calls the Windows API function SHBrowseForFolder to display the dialog. For more information on SHBrowseForFolder, see the Windows API documentation.



**Note** • Windows displays this dialog; therefore, the installation cannot change the text of the buttons on the dialog. Windows displays the button text—"Yes" and "No" on English-based systems—in the language of the operating system; no manual localization of this text is required. If you need to display a more flexible dialog, call a Windows API function directly or use a custom dialog.

### Syntax

```
SelectDirEx ( szTitle, szMsg, szEditBoxStatusText, szTreeControlStatusText, nFlags, svDir );
```



## Parameters

**Table 117** • SelectDirEx Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of this dialog. To display the default title Choose Folder, pass a null string ("" ) in this parameter.
<b>szMsg</b>	Specifies the message you want to display in this dialog. To display the default instructions for this dialog (Please choose the installation folder), pass a null string ("" ) in this parameter.
<b>szEditBoxStatusText</b>	Specifies the static text that accompanies the edit box when nFlags specifies BIF_EDITBOX. If nFlags does not specify BIF_EDITBOX, this parameter is ignored.
<b>szTreeControlStatusText</b>	Specifies the static text to accompany the dialog's tree control when nFlags specifies BIF_STATUSTEXT. If nFlags does not specify BIF_STATUSTEXT, this parameter is ignored.

**Table 117** • SelectDirEx Parameters (cont.)

Parameter	Description
<b>nFlags</b>	<p>Specifies the appearance and functionality of the dialog that is displayed by the function by specifying the same flags that are used for the Windows structure BROWSEINFO. Note that if you pass BIF_BROWSEFORCOMPUTER or BIF_BROWSEFORPRINTER, no edit box is displayed. Pass any of the following constants:</p> <ul style="list-style-type: none"> <li>• <b>BIF_BROWSEFORCOMPUTER</b>—Enables the end user to select a computer on the network. The Network Neighborhood is preselected in the tree control and the OK button is enabled only if a valid computer name is selected in the tree control. No edit box is displayed, even if BIF_EDITBOX is specified.</li> <li>• <b>BIF_BROWSEFORPRINTER</b>—Enables the end user to select a printer. The My Computer folder is preselected in the tree control. Only computers that include at least one printer are displayed under the Network Neighborhood folder. The OK button is enabled only if a valid printer is selected in the tree control. No edit box is displayed, even if BIF_EDITBOX is specified.</li> <li>• <b>BIF_DONTGOBELOWDOMAIN</b>—Network folders below the domain level are not displayed in the tree control.</li> <li>• <b>BIF_RETURNFSANCESTORS</b>—Shows folders above svDir.</li> <li>• <b>BIF_RETURNONLYFSDIRS</b>—Browses file system folders.</li> </ul> <p>The following constants specify other aspects of the dialog:</p> <ul style="list-style-type: none"> <li>• <b>BIF_STATUSTEXT</b>—Displays status text in the dialog.</li> <li>• <b>BIF_EDITBOX</b>—Adds an edit field to the Browse dialog. The end user can type a folder name in the edit box. The text specified in the SzEditBoxStaticText parameter appears above the edit box, unless BIF_BROWSEFORCOMPUTER or BIF_BROWSEFORPRINTER is specified. When the end user clicks OK, SelectDirEx checks whether a valid folder name is entered. If not, an error message appears and the dialog is not closed.</li> </ul>
<b>svDir</b>	<p>Specifies the name of a folder that will appear as the default selection. Returns the fully qualified name of the folder selected by the end user. If a valid folder name that exists on the target system is specified in this parameter, the specified folder is preselected in the tree control.</p>

## Return Values

**Table 118** • SelectDirEx Return Values

Return Value	Description
<b>IDOK (1)</b>	Indicates that the end user clicked the OK button.
<b>IDCANCEL (2)</b>	Indicates that the end user clicked the Cancel button.
<b>&lt; 0</b>	Indicates that the function was unable to display the dialog.

## Additional Information

The dialog that is displayed by this function cannot be displayed with a skin; it appears the same regardless of whether you have specified a skin.

## SelectDirEx Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SelectDirEx function.
 *
 * This sample script calls SelectDirEx to display a dialog
 * that allows the user to specify a folder. If the specified
 * folder does not exist, it is not created; instead, an error
 * message is displayed and the SelectDirEx dialog is displayed
 * again.
 *
 \*-----*/

#include "ifx.h"

function OnBegin()
    STRING  szTitle, szMsg, svDir;
    BOOL    bCreate, bFolderExists;
    NUMBER  nResult;
begin
    // Loop until user cancels or selects an existing folder.
    repeat
        // Set a default folder for the SelectDirEx dialog.
        svDir = TARGETDIR;

        // Set the title to display in the SelectDirEx dialog.
        szTitle = "SelectDirEx Example";

```

```

// Set the message to display in the SelectDirEx dialog.
szMsg = "Please choose an existing folder.";

// Get an existing folder name from the user.
nResult = (SelectDirEx (szTitle, szMsg, "", "",
    BIF_RETURNONLYFSDIRS | BIF_EDITBOX, svDir ) < 0) ;

if nResult = 0 then
    // Determine whether the folder exists.
    bFolderExists = ExistsDir (svDir);

    if bFolderExists = NOTEXISTS then
        // The folder does not exist. Ask user to select again.
        szMsg = "%s does not exist.\nPlease choose an existing folder.";
        sprintfBox (WARNING, szTitle, szMsg, svDir);
    endif;
endif;
until (nResult = CANCEL) || (bFolderExists = EXISTS);

if (bFolderExists = EXISTS) then
    // Display the name of the selected folder.
    sprintfBox (INFORMATION, szTitle, "You selected %s.", svDir);
endif;
end;

```

## SelectFolder



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SelectFolder** function displays a dialog that enables the end user to enter the name of a program folder in an edit field or select a program folder from a list. The function automatically displays all program folders on the system. A default folder name passed in `svDefFolder` is displayed in the edit field. The selected folder name is returned in `svResultFolder`.



**Caution** • If the specified folder does not exist, you must call *CreateProgramFolder* to create it; *SelectFolder* will not create the folder.

### Syntax

```
SelectFolder ( szTitle, szDefFolder, svResultFolder );
```

## Parameters

**Table 119** • SelectFolder Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of the dialog. To display the default title, Select Program Folder, pass a null string ("") in this parameter.
<b>szDefFolder</b>	Specifies the name of the folder to display as the default folder.
<b>svResultFolder</b>	Returns the name of the folder selected or specified by the end user.

## Return Values

**Table 120** • SelectFolder Return Values

Return Value	Description
<b>NEXT (1)</b>	Indicates that the end user clicked the Next button.
<b>BACK (12)</b>	Indicates that the end user clicked the Back button.
<b>&lt; 0</b>	Indicates that the function was unsuccessful.

## Additional Information

To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SelectFolder Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SelectFolder function.
 *
 * First, SelectFolder is called to get a folder selection from
 * the user. Then the name of the selected folder is displayed.
 *
\*-----*/

#define TITLE_TEXT "SelectFolder Example"

// Include Ifx.h for built-in InstallScript function prototypes.
```

```

#include "Ifx.h"

export prototype ExFn_SelectFolder(HWND);

function ExFn_SelectFolder(hMSI)
    STRING svResultFolder;
    NUMBER nReturn;
begin

    // Get folder selection from user. Make "Startup" the
    // default selection
    nReturn = SelectFolder (TITLE_TEXT, "Startup", svResultFolder);

    if (nReturn < 0) then

        // Report an error.
        MessageBox ("SelectFolder failed.", SEVERE);

    else

        // Display the name of the selected folder.
        sprintfBox (INFORMATION, TITLE_TEXT,
            "Selected folder: %s", svResultFolder);

    endif;

end;

```

## SendMessage

The **SendMessage** function sends a message to one or more windows. SendMessage does not return control to the setup script until the message has been processed. The SendMessage function is a direct pass-through to the Windows API SendMessage. Consult the Windows programming documentation for detailed information.



**Note** • To send a message using the parameter *nMsg* or to handle return values, you must define constants in your script that are equivalent to the constants defined in *Windows.h*. You cannot use *#include* to include *Windows.h* in your script.

### Syntax

```
SendMessage ( hWnd, nMsg, wParam, lParam );
```

## Parameters

**Table 121** • SendMessage Parameters

Parameter	Description
<b>nHwnd</b>	Specifies the handle that identifies the window to receive the message.
<b>nMsg</b>	Specifies the message to send to the window(s).
<b>nwParam</b>	Specifies additional message information.
<b>nlParam</b>	Specifies additional message information.

## Return Values

SendMessage returns the value it receives from calling the Windows API of the same name. The return value depends on the message received by the Windows API SendMessage. Consult Windows programming documentation for detailed information on messages returned by the Windows API SendMessage.

## Additional Information



### Task

**To pass string data in the fourth parameter of SendMessage, do the following:**

- For projects converted from InstallShield Professional, include the required function declaration by doing one of the following:
  - Remove the preprocessor constant `ISINCLUDE_NO_WINAPI_H` from the Preprocessor Defines box, which is available on the Compile/Link tab on the Settings dialog box. (To access this dialog box, click Settings on the Build menu.)
  - Place the following code in your script:

```
prototype USER.SendMessageA(HWND, NUMBER, NUMBER, BYREF STRING);
```

(For projects created with InstallShield, **ISRTWindows.h** is included by default in **lfx.h**, which in turn is included by default in the script.)

- Call the function as in the following example:

```
USER.SendMessageA( hwnd, LB_GETTEXT, nResult, sResult);
```

## SendMessage Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
```

```

* Demonstrates the FindWindow and SendMessage functions.
*
* This script launches Windows Notepad and then calls
* FindWindow to locate the Notepad window. Next, it calls
* SendMessage to maximize the window; after a three-second
* delay, it calls SendMessage again to minimize the
* window. When the script ends, Windows Notepad remains
* open but minimized. Note that the parameters passed to
* SendMessage are Windows system messages whose values
* are defined as constants in this script.
*
* Note: Before running this script, set the preprocessor
*       constant NOTEPAD so that it references the fully-
*       qualified name of the Windows Notepad executable.
*
\*-----*/

#define NOTEPAD "C:\\Windows\\Notepad.exe"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_SendMessage(HWND);

function ExFn_SendMessage(hMSI)
    NUMBER nMsg, nwParam, nlParam;
    HWND nHwnd;
begin

    // Do not display the setup's background window.
    Disable (BACKGROUND);

    // Open the Windows Notepad.
    if (LaunchApp (NOTEPAD, "") < 0 ) then
        MessageBox ("Unable to launch Notepad.", SEVERE);
        abort;
    endif;

    // Wait three seconds so we can view the window before
    // it's maximized.
    Delay (3);

    // Retrieve the handle of the Notepad window. The first
    // parameter is the window class. A null string in the
    // second parameter specifies the topmost Notepad window.
    nHwnd = FindWindow ("Notepad", "");

    if (nHwnd = NULL) then
        MessageBox ("Unable to find the Notepad window.", SEVERE);
    else
        // Send system command to maximize the window.
        SendMessage (nHwnd, WM_SYSCOMMAND, SC_MAXIMIZE, 0);

        // Wait three seconds so we can view the window
        // before it's minimized.
        Delay (3);

        // Send system command to minimize the window.
        SendMessage (nHwnd, WM_SYSCOMMAND, SC_MINIMIZE, nlParam);
    end
end

```



```
endif;  
  
end;
```

## ServiceAddService

The **ServiceAddService** function adds the service specified by `szServiceName` to the list of services registered on the system. You can customize the `SERVICE_IS_PARAMS` structure to control additional service creation elements. **ServiceAddService** does not install any files; you must install the service file yourself during the installation.

If the service specified by `szServiceName` already exists, **ServiceAddService** reconfigures it to the parameters specified in the function call. If the existing service is running when **ServiceAddService** is called, the installation attempts to stop the service before attempting to reconfigure the service. If the service cannot be stopped, **ServiceAddService** reconfigures the service and sets `BATCH_INSTALL` to non-zero so that Windows can complete the service reconfiguration after reboot.

### Syntax

```
ServiceAddService ( szServiceName, szServiceDisplayName, szServiceDescription, szServicePathFile,  
    bStartService, szStartServiceArgs );
```

## Parameters

**Table 122** • ServiceAddService Parameters

Parameter	Description
<b>szServiceName</b>	Specifies the name of the service.
<b>szServiceDisplayName</b>	Specifies the display name of the service.
<b>szServiceDescription</b>	Specifies the service description.
<b>szServicePathFile</b>	Specifies the path to the service executable file and, optionally, command line parameters that are passed to the service whenever it is started.
<b>bStartService</b>	Specifies whether to start the service after adding it.
<b>szStartServiceArgs</b>	Specifies command line arguments that are passed to the service only when it is started by the installation.

## Return Values

**Table 123** • ServiceAddService Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the function successfully added or reconfigured the service.
<b>&lt; ISERR_SUCCESS</b>	<p>Indicates that the function was unable to add or reconfigure the service.</p> <p>If this function fails, additional error information may be available by calling <b>GetExtendedErrInfo</b> and checking the value of its third argument. (This value typically indicates the result of internally calling the Windows API function <b>GetLastError</b> after a call to a Windows API function failed.)</p> <p>You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a>.</p>

## Additional Information

If logging is enabled when **ServiceAddService** is called, the service is logged for uninstallation whether or not the service existed already and is removed when the application is uninstalled. If you are adding a service that should not be uninstalled, disable logging before calling this function.

Note that the publicly defined structures—**SERVICE\_IS\_PARAMS** and **SERVICE\_IS\_STATUS**—are not used when the service is uninstalled automatically. If you want to specify non-default settings when uninstalling the service, disable logging when adding the service and then remove the service manually during uninstallation by calling **ServiceRemoveService** directly from the script.

# ServiceExistsService

The **ServiceExistsService** function determines whether the service specified by `szServiceName` is registered on the system.

## Syntax

```
ServiceExistsService ( szServiceName );
```

## Parameters

**Table 124** • ServiceExistsService Parameters

Parameter	Description
<b>szServiceName</b>	Specifies the name of the service.

## Return Values

**Table 125** • ServiceExistsService Return Values

Return Value	Description
<b>TRUE</b>	Indicates that the service is registered on the system.
<b>FALSE</b>	Indicates that the service is not registered on the system, or that the setup could not determine whether the service is registered on the system.

# ServiceGetServiceState

The **ServiceGetServiceState** function returns in `nvServiceState` the state of the service that is specified by `szServiceName`.

## Syntax

```
ServiceGetServiceState ( szServiceName, nvServiceState );
```

## Parameters

**Table 126** • ServiceGetServiceState Parameters

Parameter	Description
<b>szServiceName</b>	Specifies the name of the service.
<b>nvServiceState</b>	<p>Returns one of the following Windows-defined constants specifying the state of the service:</p> <ul style="list-style-type: none"> <li>• SERVICE_STOPPED</li> <li>• SERVICE_START_PENDING</li> <li>• SERVICE_STOP_PENDING</li> <li>• SERVICE_RUNNING</li> <li>• SERVICE_CONTINUE_PENDING</li> <li>• SERVICE_PAUSE_PENDING</li> <li>• SERVICE_PAUSED</li> </ul>

## Return Values

**Table 127** • ServiceGetServiceState Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the function retrieved the state of the service. Additional status information is available in the SERVICE_IS_STATUS system variable.
<b>&lt; ISERR_SUCCESS</b>	<p>Indicates that the function was unable to retrieve the state of the service.</p> <p>If this function fails, additional error information may be available by calling <b>GetExtendedErrInfo</b> and checking the value of its third argument. (This value typically indicates the result of internally calling the Windows API function <b>GetLastError</b> after a call to a Windows API function failed.)</p> <p>You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a>.</p>

# ServiceInitParams

The **ServiceInitParams** function initializes the SERVICE\_IS\_PARAMS system variable's members to the following default values. This function is called automatically during setup initialization.

## Syntax

```
ServiceInitParams ( );
```

## System Variable Members

**Table 128** • ServiceInitParams System Variable Members

Parameter	Default Value
<b>SERVICE_IS_PARAMS.IpMachineName</b>	NULL
<b>SERVICE_IS_PARAMS.IpDatabaseName</b>	NULL
<b>SERVICE_IS_PARAMS.dwDesiredAccess</b>	SERVICE_ALL_ACCESS
<b>SERVICE_IS_PARAMS.dwServiceType</b>	SERVICE_WIN32_OWN_PROCESS
<b>SERVICE_IS_PARAMS.dwStartType</b>	SERVICE_AUTO_START
<b>SERVICE_IS_PARAMS.dwErrorControl</b>	SERVICE_ERROR_IGNORE
<b>SERVICE_IS_PARAMS.IpLoadOrderGroup</b>	NULL
<b>SERVICE_IS_PARAMS.IpdwTagId</b>	NULL
<b>SERVICE_IS_PARAMS.IpDependencies</b>	NULL
<b>SERVICE_IS_PARAMS.IpServiceStartName</b>	NULL
<b>SERVICE_IS_PARAMS.IpPassword</b>	NULL
<b>SERVICE_IS_PARAMS.nStartServiceWaitCount</b>	INFINITE
<b>SERVICE_IS_PARAMS.nStopServiceWaitCount</b>	INFINITE

## Parameters

None.

## Return Values

**Table 129** • ServiceInitParams Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	This function always returns ISERR_SUCCESS.

# ServiceRemoveService

The **ServiceRemoveService** function removes the service that is specified by szServiceName from the service control manager database. If the service is running when the function is called, the installation stops the service before removing the service.

## Syntax

```
ServiceRemoveService ( szServiceName );
```

## Parameters

**Table 130** • ServiceRemoveService Parameters

Parameter	Description
<b>szServiceName</b>	Specifies the name of the service.

## Return Values

**Table 131** • ServiceRemoveService Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the function removed the service from the service control manager database.
<b>&lt; ISERR_SUCCESS</b>	<p>Indicates that the function was unable to remove the service.</p> <p>If this function fails, additional error information may be available by calling <b>GetExtendedErrInfo</b> and checking the value of its third argument. (This value typically indicates the result of internally calling the Windows API function <b>GetLastError</b> after a call to a Windows API function failed.)</p> <p>You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a>.</p>

# ServiceStartService

The **ServiceStartService** function starts the service that is specified by szServiceName. If the service is running when the function is called, the installation stops it and then restarts it.

This function waits for the service to reach its running state before returning; it waits indefinitely as long as the service is updating the dwCheckPoint member of the SERVICE\_IS\_STATUS structure at least every dwWaitHint milliseconds. To force the function to return after a specific time interval, regardless of whether the service has started, you can change the nStartServiceWaitCount member of the structured variable SERVICE\_IS\_PARAMS to the appropriate time in seconds.

## Syntax

```
ServiceStartService ( szServiceName, szStartServiceArgs );
```

## Parameters

**Table 132** • ServiceStartService Parameters

Parameter	Description
szServiceName	Specifies the name of the service.
szStartServiceArgs	Specifies a comma-delimited string of arguments to the service.

## Return Values

**Table 133** • ServiceStartService Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the function started the service.
<b>&lt; ISERR_SUCCESS</b>	<p>Indicates that the function was unable to start the service.</p> <p>If this function fails, additional error information may be available by calling <b>GetExtendedErrInfo</b> and checking the value of its third argument. (This value typically indicates the result of internally calling the Windows API function <b>GetLastError</b> after a call to a Windows API function failed.)</p> <p>You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a>.</p>

# ServiceStopService

The **ServiceStopService** function stops the service that is specified by szServiceName.

This function waits for the service to stop before returning; it waits indefinitely as long as the service is updating the dwCheckPoint member of the SERVICE\_IS\_STATUS structure at least every dwWaitHint milliseconds. To force the function to return after a specific time interval, regardless of whether the service has stopped, you can change the nStopServiceWaitCount member of the structured variable SERVICE\_IS\_PARAMS to the appropriate time in seconds.

## Syntax

```
ServiceStopService ( szServiceName );
```

## Parameters

**Table 134** • ServiceStopService Parameters

Parameter	Description
<b>szServiceName</b>	Specifies the name of the service.

## Return Values

**Table 135** • ServiceStopService Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the function stopped the service.
<b>&lt; ISERR_SUCCESS</b>	<p>Indicates that the function was unable to stop the service.</p> <p>If this function fails, additional error information may be available by calling <b>GetExtendedErrInfo</b> and checking the value of its third argument. (This value typically indicates the result of internally calling the Windows API function <b>GetLastError</b> after a call to a Windows API function failed.)</p> <p>You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a>.</p>

## Additional Information

# SetColor

The **SetColor** function sets the color of the setup background.



**Note** • This function is not supported for use in Basic MSI setup projects.

## Syntax

```
SetColor ( nObject, nColor );
```




## Parameters

**Table 136 • SetColor Parameters**

Parameter	Description
<b>nObject</b>	<p>Specifies the user interface object to change. Pass the following predefined constant in this parameter:</p> <ul style="list-style-type: none"><li>● BACKGROUND—Indicates the background of the setup window. The default color is solid teal: RGB (0,128,128).</li></ul>

**Table 136 • SetColor Parameters (cont.)**

Parameter	Description
<b>nColor</b>	<p>Specifies a color for the background.</p> <p>For a gradient background color, pass one of the following constants:</p> <ul style="list-style-type: none"> <li>• BK_BLUE</li> <li>• BK_GREEN</li> <li>• BK_MAGENTA</li> <li>• BK_ORANGE</li> <li>• BK_PINK</li> <li>• BK_RED</li> <li>• BK_YELLOW</li> </ul> <p>For a solid background color, pass one of the following constants or call the <a href="#">RGB</a> function to indicate a particular color (using the values provided in the parentheses):</p> <ul style="list-style-type: none"> <li>• BK_SOLIDBLACK (0, 0, 0)</li> <li>• BK_SOLIDBLUE (0, 0, 255)</li> <li>• BK_SOLIDGREEN (0, 255, 0)</li> <li>• BK_SOLIDMAGENTA (255, 0, 127)</li> <li>• BK_SOLIDORANGE (255, 127, 0)</li> <li>• BK_SOLIDPINK (255, 0, 255)</li> <li>• BK_SOLIDRED (255, 0, 0)</li> <li>• BK_SOLIDWHITE (255, 255, 255)</li> <li>• BK_SOLIDYELLOW (255, 255, 0)</li> </ul> <p>For a custom color, pass the function RGB in this parameter.</p>
	 <p><b>Note</b> • When using an RGB value, you can apply the same method as described in the programming manuals for Microsoft Windows. You specify the mixture of RED, GREEN, and BLUE colors to “mix” the color. Use a number from 0 to 255 to represent the amount of color to use. You must use literal numbers in the parameters of the RGB macro. You can also use a long value that represents the RGB color instead of the RGB statement.</p> <p>To achieve a smoothing effect (gradient) when the background is painted with a custom color, bitwise OR the color with the predefined constant BK_SMOOTH. Note that the smoothing effect is better with 256 colors enabled.</p>

## Return Values

**Table 137** • SetColor Return Values

Return Value	Description
0	Indicates that the function successfully set the color of the object.
< 0	Indicates that the function was unable to set the color of the user interface object.

## SetColor Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SetColor function.
 *
 * The first call to SetColor sets the background color to solid
 * blue. The second call sets the background color to gradient
 * red. The last call sets the background color to an RGB value.
 *
 \*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_SetColor(HWND);

function ExFn_SetColor(hMSI)
begin

    // Change the background color to solid blue.
    if (SetColor (BACKGROUND, BK_SOLIDBLUE) < 0) then
        MessageBox ("SetColor failed.", SEVERE);
    endif;

    // Delay for three seconds.
    Delay (3);

    // Change the background color to gradient red.
    if (SetColor (BACKGROUND, BK_RED) < 0) then
        MessageBox ("SetColor failed.", SEVERE);
    endif;

    // Delay for three seconds.
    Delay (3);

    // Change the background color to custom magenta.
    if (SetColor (BACKGROUND, RGB(100, 50, 150)) < 0) then
        MessageBox ("SetColor failed.", SEVERE);
    endif;
end

```

```
endif;  
  
Delay (3);  
  
end;
```

## SetDialogTitle

The **SetDialogTitle** function changes the titles that appear in the title bars of some common built-in dialogs. Specify the dialog using the parameter `nDialogId`. If you do not use `SetDialogTitle`, the default title appears.

When you set the title for a particular dialog, InstallShield uses that title for every instance of that type of dialog until you use `SetDialogTitle` to change the title again. You must call `SetDialogTitle` separately for each type of dialog whose title you wish to change.



---

**Note** • *InstallShield creates message boxes using standard Windows message box functions. Windows determines the OK and Cancel button text for these message boxes. InstallShield cannot control the text used in the buttons inside Windows message boxes.*

### Syntax

```
SetDialogTitle ( nDialogId, szTitle );
```

## Parameters

**Table 138** • SetDialogTitle Parameters

Parameter	Description
<b>nDialogId</b>	<p>Identifies the built-in dialog whose title is to be changed change. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>DLG_ASK_OPTIONS</b>—Changes the title of the <a href="#">AskOptions</a> dialog.</li> <li>• <b>DLG_ASK_PATH</b>—Changes the title of the <a href="#">AskPath</a> dialog.</li> <li>• <b>DLG_ASK_TEXT</b>—Changes the title of the <a href="#">AskText</a> dialog.</li> <li>• <b>DLG_ASK_YESNO</b>—Changes the title of the <a href="#">AskYesNo</a> dialog.</li> <li>• <b>DLG_ENTER_DISK</b>—Changes the title of the <a href="#">EnterDisk</a> dialog.</li> <li>• <b>DLG_MSG_INFORMATION</b>—Changes the title of the Information-style <a href="#">MessageBox</a>.</li> <li>• <b>DLG_MSG_SEVERE</b>—Changes the title of the Severe-style <a href="#">MessageBox</a>.</li> <li>• <b>DLG_STATUS</b>—Changes the title of the dialog-style progress indicator. You must re-enable the dialog-style progress indicator by calling <a href="#">Enable(STATUSDLG)</a> after calling <a href="#">SetDialogTitle</a> with the <b>DLG_STATUS</b> option in order for the title change to take effect.</li> <li>• <b>DLG_MSG_WARNING</b>—Changes the title of the Warning-style <a href="#">MessageBox</a>.</li> <li>• <b>DLG_USER_CAPTION</b>—Changes the <a href="#">MessageBox</a> caption when you use the user-defined message box styles.</li> </ul>
<b>szTitle</b>	Specifies the new title.

## Return Values

**Table 139** • SetDialogTitle Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully changed the title of the dialog.
<b>&lt; 0</b>	Indicates that the function was unable to change the title of the dialog.

## SetDialogTitle Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
```

```

*
* Demonstrates the SetDialogTitle function.
*
* SetDialogTitle is called to change the title of the
* AskYesNo dialog.
*
\*-----*/

#define TITLE_TEXT "SetDialogTitle Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_SetDialogTitle(HWND);

function ExFn_SetDialogTitle(hMSI)
    NUMBER nCheck1, nCheck2;
begin

    // Set the title for the AskYesNo dialog.
    if (SetDialogTitle (DLG_ASK_YESNO, TITLE_TEXT) < 0) then

        // Report an error.
        MessageBox ("SetDialogTitle failed.", SEVERE);

    else

        // Display the AskYesNo dialog with its new title.
        AskYesNo ("Did SetDialogTitle change this title?", YES);

    endif;

end;

```

## SetDisplayEffect



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SetDisplayEffect** function specifies the display effect to be used when displaying bitmaps or metafiles with the **PlaceBitmap** function or when displaying billboards. Once the display effect has been set, all bitmaps subsequently displayed by **PlaceBitmap**, or billboards, are displayed with this effect until a new effect is set by another call to **SetDisplayEffect**.

Display effects occur only when placing bitmaps or billboards. No display effects are used when removing bitmaps or billboards.




**Note** • A bitmap that is displayed by **PlaceBitmap** with the option **BITMAPICON**, **FULLSCREEN**, **FULLSCREENSIZE**, or **TILED** is not displayed with display effects. Instead it is displayed normally. For more information, see [PlaceBitmap](#).

## Syntax

```
SetDisplayEffect ( nEffect );
```

## Parameters

**Table 140** • SetDisplayEffect Parameters

Parameter	Description
<b>nEffect</b>	<p>Specifies a display effect. Pass one of the following predefined constants in this parameter. Note that these constants are exclusive; you cannot use the bitwise OR operator and specify more than one. Moreover, this parameter has no effect when BITMAPICON, FULLSCREEN, FULLSCREENSIZE, or TILED is specified when displaying a bitmap with PlaceBitmap.</p> <ul style="list-style-type: none"> <li>• <b>EFF_FADE</b>—The bitmap or billboard slowly fades in and out.</li> <li>• <b>EFF_REVEAL</b>—The bitmap or billboard gradually fills in from the center toward all sides.</li> <li>• <b>EFF_HORZREVEAL</b>—The bitmap or billboard gradually scrolls out horizontally from its center.</li> <li>• <b>EFF_HORZSTRIPE</b>—A section of the bitmap or billboard fills in horizontally from the outside in; then the remainder fills in from the center out.</li> <li>• <b>EFF_VERTSTRIPE</b>—A section of the bitmap or billboard fills in vertically from the outside in; then the remainder fills in from the center out.</li> <li>• <b>EFF_BOXSTRIPE</b>—A section of the bitmap or billboard fills in from all sides; then the remainder fills in toward all sides.</li> <li>• <b>EFF_NONE</b>—This option is the default setting. Use it to clear the display effects after calling one of the other options.</li> </ul> <p> <b>Note</b> • Only <b>EFF_REVEAL</b> and <b>EFF_HORZREVEAL</b> work with metafiles.</p>

## Return Values

**Table 141** • SetDisplayEffect Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully set the display effect.
<b>&lt; 0</b>	Indicates that the function was unable to set the display effect.

## SetDisplayEffect Example

```
/*-----*\
 *
 * InstallShield Example Script
```

```

*
* Demonstrates the SetDisplayEffect function.
*
* This script displays the same bitmap seven times, with a
* different effect each time.
*
* Note: In order for this script to run correctly, you must
*       set the constant BITMAP_FILE so that it references an
*       existing bitmap file on the target system. The
*       constant BITMAP_ID can be any integer in this example
*       since the bitmap is not loaded from a DLL and there
*       are no other bitmaps on the screen.
*
\*-----*/

#define BITMAP_FILE "C:\\Windows\\Forest.BMP"

// ID to use for the bitmap.
#define BITMAP_ID 1

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_SetDisplayEffect(HWND);

function ExFn_SetDisplayEffect(hMSI)
begin

    // Open a background window so the effect's name can
    // be shown in the title bar each time the bitmap
    // is displayed.
    Enable (FULLWINDOWMODE);
    Enable (BACKGROUND);

    // 1. Display bitmap using the box stripe effect.
    SetDisplayEffect (EFF_FADE);

    SetTitle ("Fade effect", 0, BACKGROUNDCAUTION);
    PlaceBitmap (BITMAP_FILE, BITMAP_ID, 0, 0, CENTERED);
    Delay (3);
    PlaceBitmap ("", BITMAP_ID, 0, 0, REMOVE);

    // 2. Display bitmap using the reveal effect.
    SetDisplayEffect (EFF_REVEAL);

    SetTitle ("Reveal effect", 0, BACKGROUNDCAUTION);
    PlaceBitmap (BITMAP_FILE, BITMAP_ID, 0, 0, CENTERED);
    Delay (3);
    PlaceBitmap ("", BITMAP_ID, 0, 0, REMOVE);

    // 3. Display bitmap using the horizontal reveal effect.
    SetDisplayEffect (EFF_HORZREVEAL);

    SetTitle ("Horizontal reveal effect", 0, BACKGROUNDCAUTION);
    PlaceBitmap (BITMAP_FILE, BITMAP_ID, 0, 0, CENTERED);
    Delay (3);
    PlaceBitmap ("", BITMAP_ID, 0, 0, REMOVE);

    // 4. Display bitmap using the horizontal stripe effect.

```



```

SetDisplayEffect (EFF_HORIZSTRIPE);

SetTitle ("Horizontal stripe effect", 0, BACKGROUNDCAUTION);
PlaceBitmap (BITMAP_FILE, BITMAP_ID, 0, 0, CENTERED);
Delay (3);
PlaceBitmap ("", BITMAP_ID, 0, 0, REMOVE);

// 5. Display bitmap using the vertical stripe effect.
SetDisplayEffect (EFF_VERTSTRIPE);

SetTitle ("Vertical stripe effect", 0, BACKGROUNDCAUTION);
PlaceBitmap (BITMAP_FILE, BITMAP_ID, 0, 0, CENTERED);
Delay (3);
PlaceBitmap ("", BITMAP_ID, 0, 0, REMOVE);

// 6. Display bitmap using the box stripe effect.
SetDisplayEffect (EFF_BOXSTRIPE);

SetTitle ("Box stripe effect", 0, BACKGROUNDCAUTION);
PlaceBitmap (BITMAP_FILE, BITMAP_ID, 0, 0, CENTERED);
Delay (3);
PlaceBitmap ("", BITMAP_ID, 0, 0, REMOVE);
Delay (1);

// 7. Clear all effects.
SetDisplayEffect (EFF_NONE);

SetTitle ("No effect", 0, BACKGROUNDCAUTION);
PlaceBitmap (BITMAP_FILE, BITMAP_ID, 0, 0, CENTERED);
Delay (3);
PlaceBitmap ("", BITMAP_ID, 0, 0, REMOVE);

Delay (1);

end;

```

## SetErrorMsg

When a disk error occurs, the **SetErrorMsg** function sets the corresponding error message that is displayed by the **EnterDiskError** function.

### Syntax

```
SetErrorMsg (nErrorID, szText);
```

## Parameters

**Table 142 • SetLastErrorMsg Parameters**

Parameter	Description
<b>nErrorID</b>	Specifies which error message to customize. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li><b>ERR_BOX_BADPATH</b>—This message appears when <b>EnterDiskError</b> detects a bad path entered by the user.</li> <li><b>ERR_BOX_BADTAGFILE</b>—This message appears when <b>EnterDiskError</b> detects the specified tag file does not exist on the disk.</li> <li><b>ERR_BOX_DISKID</b>—This message appears when <b>EnterDiskError</b> detects the drive specified by the user does not exist.</li> </ul>
<b>szText</b>	Specifies the error message to display in the message box.

## Return Values

**Table 143 • SetLastErrorMsg Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully changed the error message.
<b>&lt; 0</b>	Indicates that the function was unable to change the error message.

## SetErrorMsg Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the SetLastErrorMsg function.
*
* This script customizes the error messages displayed after a
* call to EnterDisk if the next disk in a set of setup disks
* is not ready in the specified drive.
*
\*-----*/

// Define the text messages for EnterDisk errors.
#define MSG_DRIVE_DOOR "The drive door is open."
#define MSG_BAD_PATH   "The path does not exist."
#define MSG_BAD_TAG    "Bad tag file."
#define MSG_BAD_DRIVE  "The specified drive does not exist."
```

```
// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_SetErrorMsg(HWND);

function ExFn_SetErrorMsg(hMSI)
begin

    // Set the messages for the error boxes of the EnterDisk function.
    SetErrorMsg (ERR_BOX_DRIVEOPEN, MSG_DRIVE_DOOR);
    SetErrorMsg (ERR_BOX_BADPATH, MSG_BAD_PATH );
    SetErrorMsg (ERR_BOX_BADTAGFILE, MSG_BAD_TAG);
    SetErrorMsg (ERR_BOX_DISKID, MSG_BAD_DRIVE);

    // Prompt the user to specify a disk.
    EnterDisk ("Please enter the 'Examples' disk:", "Example.exe");

end;
```

## SetErrorTitle

When a disk error occurs, the **SetErrorTitle** function sets the title bar for the error message that is displayed by the **EnterDiskError** function.

### Syntax

```
SetErrorTitle (nErrorID, szText);
```

## Parameters

**Table 144 • SetErrorTitle Parameters**

Parameter	Description
<b>nErrorID</b>	Specifies the error message box whose title you want to customize. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li><b>ERR_BOX_BADPATH</b>—This message appears when <b>EnterDiskError</b> detects a bad path.</li> <li><b>ERR_BOX_BADTAGFILE</b>—This message box appears when <b>EnterDiskError</b> detects that the specified tag file does not exist on the disk.</li> <li><b>ERR_BOX_DISKID</b>—This message box appears when <b>EnterDiskError</b> detects that the specified drive does not exist.</li> </ul>
<b>szText</b>	Specifies the title to display in the error message box.

## Return Values

**Table 145 • SetErrorTitle Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully changes the text in the title bar.
<b>&lt; 0</b>	Indicates that the function was unable to change the text in the title bar.

## SetErrorTitle Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the SetErrorTitle function.
*
* This script customizes the title text for the message box
* displayed after a call to EnterDisk if the next disk in set
* of setup disks is not ready in the specified drive.
*
\*-----*/

// Define the message box title text for EnterDisk errors.
#define MSG_DRIVE_DOOR "Drive door is open."
#define MSG_BAD_PATH "Path not found."
#define MSG_BAD_TAG "Bad tag file."
#define MSG_BAD_DRIVE "Drive not found."
```

```
// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_SetErrorTitle(HWND);

function ExFn_SetErrorTitle(hMSI)
    STRING szText;
    NUMBER nErrorID;
begin

    // Set the message box title for each error that can
    // occur after a call to EnterDisk.
    SetErrorTitle (ERR_BOX_DRIVEOPEN, MSG_DRIVE_DOOR);
    SetErrorTitle (ERR_BOX_DISKID,    MSG_BAD_DRIVE);
    SetErrorTitle (ERR_BOX_BADTAGFILE, MSG_BAD_TAG);
    SetErrorTitle (ERR_BOX_BADPATH,   MSG_BAD_PATH);

    // Make drive A: the default.

    // Prompt the user to specify a disk.
    EnterDisk ("Please enter the 'Examples' disk:", "Example.exe");

end;
```

## SetExtendedErrInfo



**Project** • This information applies to InstallScript projects.

The **SetExtendedErrInfo** function sets error information, which can be retrieved by **GetExtendedErrInfo**.

### Syntax

```
SetExtendedErrInfo ( szScriptFile, nLineNumber, nError );
```

## Parameters

**Table 146** • SetExtendedErrInfo Parameters

Parameter	Description
<b>szScriptFile</b>	Specifies the script file in which the error occurred. To specify the current script file (that is, the script file that contains the call to SetExtendedErrInfo), pass the reserved identifier <code>__FILE__</code> .
<b>nLineNumber</b>	Specifies the line number in which the error occurred. To specify the current line number (that is, the line number of the call to SetExtendedErrInfo), pass the reserved identifier <code>__LINE__</code> .
<b>nError</b>	Specifies the error code for the error.

## Return Values

**Table 147** • SetExtendedErrInfo Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	The function successfully set the error information.
<b>&lt; ISERR_SUCCESS</b>	The function failed to set the error information.

# SetFileInfo

The **SetFileInfo** function sets the modify date or time stamp of an existing file or changes the file's attributes. To change both a file's date and time you must call SetFileInfo twice, once to change the date and once to change the time. However, you can set multiple file attributes with a single call to SetFileInfo by combining constants in nAttribute with the OR (|) operator.



**Note** • You can also use this function to change a folder's attribute. For example, you can use this function to create a hidden folder.

## Syntax

```
SetFileInfo ( szPathFile, nType, nAttribute, szValue );
```

## Parameters

**Table 148 • SetFileInfo Parameters**

Parameter	Description
<b>szPathFile</b>	Specifies the fully qualified name of the file or folder whose date, time, or attributes you want to change.
<b>nType</b>	<p>Specifies the file characteristic to change. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>FILE_ATTRIBUTE</b>—Indicates that one or more of the file's attributes will be changed.</li> <li>• <b>FILE_DATE</b>—Indicates that the file's date stamp will be changed.</li> <li>• <b>FILE_TIME</b>—Indicates the file's time stamp will be changed.</li> </ul>
<b>nAttribute</b>	<p>Specifies the file attribute when nType is FILE_ATTRIBUTE. (When nType is FILE_DATE or FILE_TIME, pass 0 in this parameter.) To specify more than one file attribute, combine one or more of the following predefined constants with the OR ( ) operator:</p> <ul style="list-style-type: none"> <li>• <b>FILE_ATTR_ARCHIVED</b>—Sets the archive attribute.</li> <li>• <b>FILE_ATTR_HIDDEN</b>—Sets the hidden attribute.</li> <li>• <b>FILE_ATTR_READONLY</b>—Sets the read-only attribute.</li> <li>• <b>FILE_ATTR_SYSTEM</b>—Sets the system attribute.</li> <li>• <b>FILE_ATTR_NORMAL</b>—When this constant is specified alone, all file attributes are cleared. When the OR operator is used to combine this constant with other file attribute constants, those attributes not included in the OR expression are cleared.</li> </ul>
<b>szValue</b>	<p>Specifies one of the following, depending on the value you passed in nType:</p> <ul style="list-style-type: none"> <li>• <b>FILE_DATE</b>—Specifies a date in the format YYYY/MM/DD or YYYY\MM\DD. That date must not be earlier than "1980/01/01". When using backslashes as delimiters, remember that a backslash is inserted into a string as an escape sequence, for example, "1980\\01\\01".</li> <li>• <b>FILE_TIME</b>—Specifies a time in the format HH:MM:SS, using a 24-hour clock format; note that the seconds must be a multiple of 2.</li> <li>• <b>FILE_ATTRIBUTE</b>—Pass a null string ("") in this parameter.</li> </ul>

## Return Values

**Table 149** • SetFileInfo Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully set the file's date stamp, time stamp, or attributes.
<b>&lt; 0</b>	Indicates that the function was unable to set the file's date stamp, time stamp, or attributes.

## SetFileInfo Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the SetFileInfo function.
*
* SetFileInfo is called to set a file's date, time, and
* attributes.
*
* Note: Before running this script, create a file called
*       ISExempl.txt in the root of drive C.
*
\*-----*/

#define EXAMPLE_FILE "C:\\ISExempl.txt"

#define TITLE_TEXT   "SetFileInfo Example"

#define NEW_FILE_DATE "2003/09/12"
#define NEW_FILE_TIME "18:30:00"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_SetFileInfo(HWND);

function ExFn_SetFileInfo(hMSI)
    LIST listID;
begin
    // Create a list to hold messages.
    listID = ListCreate (STRINGLIST);

    // If an error occurred, report it; then terminate.
    if (listID = LIST_NULL) then
        MessageBox ("Unable to create list required for this example.", SEVERE);
```



```

        abort;
    endif;

    // Set the file's date.
    if (SetFileInfo (EXAMPLE_FILE, FILE_DATE, 0, NEW_FILE_DATE) < 0) then
        ListAddString (listID, "Unable to change the file\'s date.", AFTER);

    else
        ListAddString (listID, "File\'s date changed to" + NEW_FILE_DATE + ".",
AFTER);

    endif;

    // Set the file's time.
    if (SetFileInfo (EXAMPLE_FILE, FILE_TIME, 0, NEW_FILE_TIME) < 0) then
        ListAddString (listID, "Unable to change the file\'s time.", AFTER);

    else
        ListAddString (listID, "File\'s date changed to" + NEW_FILE_TIME + ".",
AFTER);

    endif;

    // Clear the file's attributes.
    if (SetFileInfo (EXAMPLE_FILE, FILE_ATTRIBUTE, FILE_ATTR_NORMAL, "") < 0)
then
        ListAddString (listID, "Unable to clear file attributes.", AFTER);

    else
        ListAddString (listID, "File attributes cleared.", AFTER);

    endif;

    // Report the results.
    SdShowInfoList (TITLE_TEXT, "Changes to " + EXAMPLE_FILE, listID);

    // Remove the list from memory.
    ListDestroy (listID);

end;

```

## SetFont



**Project** • This information applies to the following project types:

- InstallScript
- InstallScript MSI

The **SetFont** function sets the font and style when displaying text strings. You can use standard Windows fonts with this function.

### Syntax

```
SetFont ( nItemID, nFontStyle, szFontName );
```

## Parameters

**Table 150 • SetFont Parameters**

Parameter	Description
<b>nItemID</b>	Specifies the item whose font and font style is to be set. Pass the following predefined constant in this parameter: <ul style="list-style-type: none"> <li><b>FONT_TITLE</b>—Specifies the main title of the setup procedure, which is displayed in the top left corner of the setup window.</li> </ul>
<b>nFontStyle</b>	Specifies font styles. Pass one or more of the following predefined constants in this parameter. All but STYLE_NORMAL can be bitwise ORed to specify multiple styles. <ul style="list-style-type: none"> <li><b>STYLE_NORMAL</b>—No bold, italic, or shadow (cannot be ORed)</li> <li><b>STYLE_BOLD</b>—Bold text.</li> <li><b>STYLE_ITALIC</b>—Italic text.</li> <li><b>STYLE_SHADOW</b>—Text with shadow drawn.</li> <li><b>STYLE_UNDERLINE</b>—Underlined text.</li> </ul>
<b>szFontName</b>	Specifies the name of an available Windows font. Valid font names include Courier, Helv, Helvetica, Modern, Roman, Script, Terminal, Times, and TmsRmn. If the specified font is not found with the styles specified, Arial is used.

## Return Values

**Table 151 • SetFont Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully set the font.
<b>&lt; 0</b>	Indicates that the function was unable to set the font.

## SetFont Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SetFont function.
 *
 * In this script, three different titles are displayed in
 * the setup's main window; each title remains on the
 * screen for three seconds. The titles are displayed by
 * calls to SetTitle, which also sets the type size and
 * color. To control the font and font style of the titles,

```

```

* the script calls SetFont before each call to SetTitle.
*
*           Font           Size   Style           Color
*
* Title 1   Times New Roman   36    Normal         Red
* Title 2   Courier New       48    Italic          Yellow
* Title 3   Arial             60    Bold, Shadow   Blue
*
\*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_SetFont(HWND);

function ExFn_SetFont(hMSI)
begin

    Enable ( BACKGROUND );

    // Title 1: Times Roman, 36pt, Normal, Red.
    if (SetFont (FONT_TITLE, STYLE_NORMAL, "Times New Roman") < 0) then
        MessageBox ("SetFont failed.", SEVERE);
    endif;

    SetTitle ("SetFont Example 1", 36, RGB(255, 0, 0));
    Delay (3);

    // Title 2: Courier New, 48pt, Italic, Yellow.
    if (SetFont (FONT_TITLE, STYLE_ITALIC, "Courier New") < 0) then
        MessageBox ("SetFont failed.", SEVERE);
    endif;

    SetTitle ("SetFont Example 2", 48, RGB(255, 255, 0));
    Delay (3);

    // Title 3: Arial, 60pt, Bold, Shadow, Blue.
    if (SetFont (FONT_TITLE, STYLE_BOLD | STYLE_SHADOW, "Arial") < 0) then
        MessageBox ("SetFont failed.", SEVERE);
    endif;

    SetTitle ("SetFont Example 3", 60, RGB(0, 0, 255));
    Delay (3);

end;

```

## SetInstallationInfo

The **SetInstallationInfo** function sets the values of the system variables IFX\_COMPANY\_NAME, IFX\_PRODUCT\_NAME, IFX\_PRODUCT\_VERSION, and IFX\_PRODUCT\_KEY. SetInstallationInfo specifies this information for use by CreateInstallationInfo, which in an event-based script is called in the default OnShowUI event handler code to create an application information key and a per application paths key for the program you are installing.

SetInstallationInfo is a special registry-related function, designed to work with certain predefined registry keys. For more information on special registry-related functions, see [Special Registry-Related Functions](#).

## Syntax

```
SetInstallationInfo ( szCompany, szProduct, szVersion, szProductKey );
```

## Parameters

**Table 152 • SetInstallationInfo Parameters**

Parameter	Description
<b>szCompany</b>	Specifies the company name. CreateInstallationInfo uses szCompany to create a \<company> key under the HKEY_LOCAL_MACHINE\Software key in the registry.
<b>szProduct</b>	Specifies the name of the product to be installed. CreateInstallationInfo uses szProduct to create a \<product> key under the HKEY_LOCAL_MACHINE\Software\<company> key in the registry. The value in szProduct is also inserted into the first paragraph of message text in the Welcome dialog.
<b>szVersion</b>	Specifies the version number of the product. CreateInstallationInfo uses szVersion to create a \<version> key under the HKEY_LOCAL_MACHINE\Software\<company>\<product> key in the registry. Together, the \<company> key (szCompany), the \<product> key (szProduct), and the \<version> key (szVersion) are referred to as the application information key. The application information key is created immediately upon calling CreateInstallationInfo.
<b>szProductKey</b>	Specifies the name of the application's main executable file. If your product uses several executables, specify the executable that best represents the product. CreateInstallationInfo uses szProductKey to create a per application paths key under the key HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\Current Version\App Paths. The per application paths key is not actually created in the registry until you call RegDBSetItem to create a value name and value pair under the key.

## Return Values

This function always returns 0.

# SetObjectPermissions

The **SetObjectPermissions** function is used to set permissions for a file, a folder, or a registry key. The file, folder, or registry key can be installed as part of your installation, or it can be already present on the target system.

## Syntax

```
SetObjectPermissions (byval string szObject, byval number nType, byval string szDomain, byval string  
    szUser, byval number nPermissions, byval number nOptions);
```

## Parameters

**Table 153 • SetObjectPermissions Parameters**

Parameter	Description
<b>szObject</b>	<p>Specify the object (file, folder, or registry key) for which you want to set permissions.</p> <p>For files and folders, specify the full path.</p> <p>For registry keys, use one of the following in the path:</p> <ul style="list-style-type: none"> <li>• <b>CLASSES_ROOT</b>—This indicates the HKEY_CLASSES_ROOT hive.</li> <li>• <b>CURRENT_USER</b>—This indicates the HKEY_CURRENT_USER hive.</li> <li>• <b>MACHINE</b>—This indicates the HKEY_LOCAL_MACHINE hive.</li> <li>• <b>USERS</b>—This indicates the HKEY_USERS hive.</li> </ul> <p>The following example set permissions for a key in HKEY_LOCAL_MACHINE:</p> <pre>SetObjectPermissions("MACHINE\\Software\\MyProduct\\Example", IS_PERMISSIONS_TYPE_REGISTRY, "", "Users", KEY_CREATE_SUB_KEY, IS_PERMISSIONS_OPTION_DENY_ACCESS);</pre>
<b>nType</b>	<p>Indicate the type of object that is being passed through the szObject parameter.</p> <p>Valid options are:</p> <ul style="list-style-type: none"> <li>• <b>IS_PERMISSIONS_TYPE_FILE</b>—szObject is a file.</li> <li>• <b>IS_PERMISSIONS_TYPE_FOLDER</b>—szObject is a folder.</li> <li>• <b>IS_PERMISSIONS_TYPE_REGISTRY</b>—szObject is a registry key.</li> </ul>
<b>szDomain</b>	<p>Specify the domain name of the user for which permissions are being set.</p> <p>To use the local machine as the domain, pass an empty string ("") for this parameter.</p>

**Table 153** • SetObjectPermissions Parameters (cont.)

Parameter	Description
<b>szUser</b>	<p>Specify the name of the user for which permissions are being set. Available options are:</p> <ul style="list-style-type: none"> <li>• Administrators</li> <li>• Authenticated Users</li> <li>• Creator Owner</li> <li>• Everyone</li> <li>• Guests</li> <li>• Interactive</li> <li>• IUSR</li> <li>• Local Service</li> <li>• Local System</li> <li>• Network Service</li> <li>• Power Users</li> <li>• Remote Desktop Users</li> <li>• Users</li> </ul> <p>The user can be one that is being created during the installation, or one that already exists on the target system at run time.</p>

**Table 153** • SetObjectPermissions Parameters (cont.)

Parameter	Description
<b>nPermissions</b>	<p>To specify the permissions that are to be applied to the object for the specified user, pass one of the following predefined constants in this parameter. You can combine these constants by using the bitwise OR operator (   ).</p> <p>Available options are:</p> <ul style="list-style-type: none"> <li>• DELETE</li> <li>• GENERIC_ALL</li> <li>• GENERIC_EXECUTE</li> <li>• GENERIC_WRITE</li> <li>• GENERIC_READ</li> <li>• READ_CONTROL</li> <li>• STANDARD_RIGHTS_ALL</li> <li>• STANDARD_RIGHTS_EXECUTE</li> <li>• STANDARD_RIGHTS_READ</li> <li>• STANDARD_RIGHTS_REQUIRED</li> <li>• STANDARD_RIGHTS_WRITE</li> <li>• SYNCHRONIZE</li> <li>• WRITE_DAC</li> <li>• WRITE_OWNER</li> </ul> <p>The following options are applicable to files and folders:</p> <ul style="list-style-type: none"> <li>• FILE_LIST_DIRECTORY (for folders)</li> <li>• FILE_READ_DATA (for files)</li> <li>• FILE_WRITE_DATA (for files)</li> <li>• FILE_ADD_FILE (for folders)</li> <li>• FILE_APPEND_DATA (for files)</li> <li>• FILE_ADD_SUBDIRECTORY (for folders)</li> <li>• FILE_READ_EA (for files and folders)</li> <li>• FILE_WRITE_EA (for files and folders)</li> <li>• FILE_EXECUTE (for files)</li> <li>• FILE_TRAVERSE (for folders)</li> <li>• FILE_DELETE_CHILD (for folders)</li> <li>• FILE_READ_ATTRIBUTES (for files and folders)</li> <li>• FILE_WRITE_ATTRIBUTES (for files and folders)</li> <li>• FILE_ALL_ACCESS</li> </ul>



**Table 153 • SetObjectPermissions Parameters (cont.)**

Parameter	Description
<b>nPermissions (cont.)</b>	<p>The following options are applicable to registry keys:</p> <ul style="list-style-type: none"> <li>• KEY_QUERY_VALUE</li> <li>• KEY_SET_VALUE</li> <li>• KEY_CREATE_SUB_KEY</li> <li>• KEY_ENUMERATE_SUB_KEYS</li> <li>• KEY_NOTIFY</li> <li>• KEY_CREATE_LINK</li> </ul> <p>For information on each value, see “Registry Key Security and Access Rights,” “File Security and Access Rights,” and “Registry Key Security and Access Rights” in the MSDN Library.</p>
<b>nOptions</b>	<p>Pass one or more of the following predefined constants in this parameter.</p> <ul style="list-style-type: none"> <li>• <b>IS_PERMISSIONS_OPTION_64BIT_OBJECT</b>—The set of permissions that are specified in nPermissions should be set for a 64-bit key, regardless of whether the REGDB_OPTION_WOW64_64KEY option is enabled. Note that the IS_PERMISSIONS_OPTION_64BIT_OBJECT constant should not be passed on 32-bit target systems. In addition, this constant does not affect permissions for files or folders.</li> <li>• <b>IS_PERMISSIONS_OPTION_DENY_ACCESS</b>—The set of permissions that are specified in nPermissions should be denied.</li> <li>• <b>IS_PERMISSIONS_OPTION_NO_APPLYDOWN</b>—The permissions should be applied to only the specified object; they should not be propagated to any child objects.</li> <li>• <b>IS_PERMISSIONS_OPTION_ALLOW_ACCESS</b>—The set of permissions that are specified in nPermissions should allow access.</li> </ul> <p>You can combine more than one constant by using the bitwise OR operator ( ). The IS_PERMISSIONS_OPTION_DENY_ACCESS and IS_PERMISSIONS_OPTION_ALLOW_ACCESS constants should not be combined because they are mutually exclusive; if you specify both of these options for nOptions, permissions are denied.</p>

## Return Values

**Table 154** • SetObjectPermissions Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	The function successfully set the permissions.
<b>!= ISERR_SUCCESS</b>	The function could not set the permissions. The return value is a Win32 error. For documentation on Win32 errors, see the MSDN Library.

## Additional Information

**SetObjectPermissions** attempts to set permissions on a 64-bit key if the REGDB\_OPTION\_WOW64\_64KEY option is enabled.

To set permissions for files that are in the 64-bit System32 folder, file system redirection should be disabled before the **SetObjectPermissions** function is called. To disable file system redirection, use the WOW64FSREDIRECTION constant, as shown in the following code:

```
Disable (WOW64FSREDIRECTION);
```

Once the operation is done, disable file system redirection:

```
Enable(WOW64FSREDIRECTION);
```

## SetObjectPermissions Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
//-----
//
// InstallScript Example Script
//
// Demonstrates the SetObjectPermissions function.
//
// This sample shows how to prevent the local administrator
// from changing a file or any of its attributes.
//
//-----

function OnFirstUIAfter()
    STRING szTitle, szMsg1, szMsg2, szOpt1, szOpt2;
    NUMBER bvOpt1, bvOpt2;
begin

    SetObjectPermissions (TARGETDIR+"MyFile.exe",IS_PERMISSIONS_TYPE_FILE, "", "administrator",
        FILE_WRITE_DATA|FILE_APPEND_DATA|FILE_WRITE_EA|FILE_WRITE_ATTRIBUTES,
        IS_PERMISSIONS_OPTION_DENY_ACCESS);

end;
```

# SetShortcutProperty

The **SetShortcutProperty** function sets one or more shortcut properties that you want the Windows Shell to set at installation run time. For example, **SetShortcutProperty** has built-in support for enabling you to set Shell properties that control the following behavior:

- Specify whether a shortcut should be pinned to the Windows 8 Start screen.
- Specify whether end users should be able to pin a shortcut to the taskbar or Start menu on Windows 7 or later systems.
- Prevent a shortcut on the Start menu from being highlighted as newly installed on Windows 7 or later systems.

**SetShortcutProperty** also lets you set additional properties that the Windows Shell supports.




---

**Note** • The shortcut and its target must be present on the target system before **SetShortcutProperty** can be called.

**SetShortcutProperty** does not support the configuration of Internet shortcuts.

## Syntax

```
SetShortcutProperty (szShortcutFolder, szName, szPropertyKey, szValue);
```

## Parameters

**Table 155 • SetShortcutProperty Parameters**

Parameter	Description
<b>szShortcutFolder</b>	<p>Specify the path of the shortcut whose properties you want to configure.</p> <p>To configure a shortcut that is in a specific folder, specify the fully qualified path—for example:</p> <p><b>C:\\ProgramData\\Microsoft\\Windows\\Start Menu\\Programs</b></p> <p>If the shortcut is on the Programs menu of the Start menu, you can pass a null string ("") in this parameter.</p> <p>You can pass one of the following InstallScript system variables in this parameter:</p> <ul style="list-style-type: none"> <li>● <b>FOLDER_DESKTOP</b>—Adds the shortcut to the desktop.</li> <li>● <b>FOLDER_STARTUP</b>—Adds the shortcut to the Startup menu.</li> <li>● <b>FOLDER_STARTMENU</b>—Adds the shortcut to the Start menu.</li> <li>● <b>FOLDER_PROGRAMS</b>—Adds the shortcut to the Start\\Programs menu.</li> </ul> <p>You can also specify a path relative to a folder that is identified by an InstallScript system variable—for example:</p> <p><b>FOLDER_PROGRAMS ^ "ACCESSORIES\\GAMES"</b></p>
<b>szName</b>	Specify the name of the shortcut that you are configuring.

**Table 155 • SetShortcutProperty Parameters (cont.)**

Parameter	Description
<b>szPropertyKey</b>	<p>Specify the property key name that you want to set. The properties that can be set are defined in <b>propkey.h</b>, which is part of the Windows SDK.</p> <p>You can pass any of the following constants for property keys that are predefined in InstallScript:</p> <ul style="list-style-type: none"> <li>• <b>SSP_PROPERTY_PREVENT_PINNING</b>—Do not allow the shortcut to be pinned to the Start menu or taskbar on Windows 7 or later systems. This option hides the context menu commands that enable end users to pin the shortcut to the taskbar and to the Start menu.</li> </ul> <p>You may want to prevent pinning for shortcuts that are for tools and secondary products that are part of your installation.</p> <p>To enable this property, set szValue to 1.</p> <ul style="list-style-type: none"> <li>• <b>SSP_PROPERTY_NO_NEW_INSTALL_HIGHLIGHT</b>—Do not highlight the shortcut as newly installed after end users install your product on Windows 7 or later systems. This has the same effect as clearing the Highlight newly installed programs check box in the Customize Start Menu dialog box for an individual item on a target system.</li> </ul> <p>You may want to use this option for shortcuts that are for tools and secondary products that are part of your installation.</p> <p>To enable this property, set szValue to 1.</p> <ul style="list-style-type: none"> <li>• <b>SSP_PROPERTY_NO_STARTSCREEN_PIN</b>—Do not pin the shortcut to the Start screen by default on Windows 8 target systems. If you pass this constant, the installation sets a Windows Shell property that was introduced in Windows 8.</li> </ul> <p>You may want to prevent pinning for shortcuts that are for tools and secondary products that are part of your installation.</p> <p>To enable this property, set szValue to 1.</p> <p>For more information on SSP_PROPERTY_PREVENT_PINNING and SSP_PROPERTY_NO_STARTSCREEN_PIN, see the <a href="#">Additional Information section</a>.</p>
<b>szValue</b>	<p>Specify the value (represented as a string) of the property that you are setting. For information on the value to use with a given property, see <b>propkey.h</b>, which is part of the Windows SDK.</p> <p>The InstallScript engine converts the string value into the appropriate data type that corresponds with the property that is specified in szPropertyKey.</p>

## Return Values

**Table 156:**

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the function successfully set the shortcut property.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was unable to set the shortcut property.  You can obtain the error message text associated with a large negative return value—for example, -2147024891(0x80070005)—by calling <a href="#">FormatMessage</a> .

## Additional Information

Note the following details about two of the nFlag constants.

### SSP\_PROPERTY\_PREVENT\_PINNING

If you configure the shortcut to prevent pinning to the taskbar and the Start menu, the target of the shortcut is ineligible for inclusion in the most frequently used list on the Start menu.

Shortcuts that contain certain strings cannot be pinned to the taskbar or the Start menu, and they cannot be displayed in the most frequently used list. Examples are:

- Documentation
- Help
- Install
- Remove
- Setup
- Support

### SSP\_PROPERTY\_NO\_STARTSCREEN\_PIN

Note that Windows 8 maintains information about shortcut pinning to the Start screen after a shortcut is removed by uninstalling the application. Therefore, the SSP\_PROPERTY\_NO\_STARTSCREEN\_PIN constant has no effect on the target system if the shortcut has already been installed on it. Thus, when you are testing this functionality, ensure that you test on a clean machine—one on which this shortcut and its target have never been installed.

## SetShortcutProperty Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
```

```

* Demonstrates the SetShortcutProperty function.
*
* This example sets a Shell property for a shortcut. The
* property hides the context menu commands that enable
* end users to pin the shortcut to the taskbar and to the
* Start menu.
*
* Note: In order for this script to run correctly, add an
* executable file to your project, and in the Shortcuts view,
* create a shortcut called My Shortcut. The target of the
* shortcut should be the executable file that you added to
* your project. The location of the shortcut should be the
* desktop.
*
\*-----*/

function OnFirstUIAfter()
    STRING szName, szValue;
begin

    // Set up parameters for the SetShortcutProperty call.
    szName = "My Shortcut";
    szValue = "1";

    // Set the shortcut property that prevents pinning to the taskbar and the Start menu.
    if (SetShortcutProperty (FOLDER_DESKTOP, szName, SSP_PROPERTY_PREVENT_PINNING,
        szValue) < 0) then
        MessageBox ("SetShortcutProperty failed", SEVERE);
    else
        sprintfBox (INFORMATION, "SetShortcutProperty", "%s configured successfully.",
            szName);
    endif;

end;

```

## SetStatus



**Project** • This information applies to InstallScript projects.

The **SetStatus** function is called in an object script to set the object's Status.Number and Status.Description properties. To set the object's other status properties, call SetStatusEx.

### Syntax

```
SetStatus ( nNumber, szDescription );
```

## Parameters

**Table 157** • SetStatus Parameters

Parameter	Description
<b>nNumber</b>	Specifies the value of Status.Number.
<b>szDescription</b>	Specifies the value of Status.Description.

## Return Values

**Table 158** • SetStatus Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	The function successfully set the status properties.
<b>&lt; ISERR_SUCCESS</b>	The function failed to set the status properties.

# SetStatusEx



**Project** • This information applies to *InstallScript* projects.

The **SetStatusEx** function is called in an object script to set the object's status properties.

## Syntax

```
SetStatusEx ( nNumber, szDescription, szSource, szScriptFile, nScriptLine, nScriptError );
```



## Parameters

**Table 159** • SetStatusEx Parameters

Parameter	Description
<b>nNumber</b>	Specifies the value of Status.Number.
<b>szDescription</b>	Specifies the value of Status.Description.
<b>szSource</b>	Specifies the value of Status.szSource.
<b>szScriptFile</b>	Specifies the value of Status.szScriptFile. To specify the current script file (that is, the script file that contains the call to SetStatusEx), pass the reserved identifier <code>__FILE__</code> .
<b>nScriptLine</b>	Specifies the value of Status.nScriptLine. To specify the current line number (that is, the line number of the call to SetStatusEx), pass the reserved identifier <code>__LINE__</code> .
<b>nScriptError</b>	Specifies the value of Status.nScriptError.

## Return Values

**Table 160** • SetStatusEx Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	The function successfully set the status properties.
<b>&lt; ISERR_SUCCESS</b>	The function failed to set the status properties.

# SetStatusExStaticText


The **SetStatusExStaticText** function sets the static text displayed in the STATUSEX dialog above the status text. The events OnFirstUIBefore, OnMaintUIBefore, and OnUpdateUIBefore automatically call this function to set the status text appropriately.

## Syntax

```
SetStatusExStaticText ( szString);
```

## Parameters

**Table 161** • SetStatusExStaticText Parameters

Parameter	Description
<b>szString</b>	<p>The text to be set as the static text. The following standard strings are available using the SdLoadString function:</p> <ul style="list-style-type: none"> <li>• <b>IDS_IFX_STATUSEX_STATICTEXT_FIRSTUI</b>—The InstallShield Wizard is installing %P.</li> <li>• <b>IDS_IFX_STATUSEX_STATICTEXT_MAINTUI_MODIFY</b>—The InstallShield Wizard is modifying %P.</li> <li>• <b>IDS_IFX_STATUSEX_STATICTEXT_MAINTUI_REPAIR</b>—The InstallShield Wizard is repairing %P.</li> <li>• <b>IDS_IFX_STATUSEX_STATICTEXT_MAINTUI_REMOVEALL</b>—The InstallShield Wizard is removing %P.</li> <li>• <b>IDS_IFX_STATUSEX_STATICTEXT_UPDATEUI</b>—The InstallShield Wizard is updated %VI of %P to version %VS.</li> </ul> <p> <b>Note</b> • The status dialog supports the text substitution, %P, but does not support any other text substitutions such as %VI or %VS. If you are setting the static text to a string with any other text substitution than %P, you must call the SdSubstituteProductInfo function to update these text substitutions to the correct value before calling SetStatusExStaticText.</p>

## Return Values

**Table 162** • SetStatusExStaticText Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	Indicates that the function was successful.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was not successful.

# SetStatusWindow

The **SetStatusWindow** function sets an initial or current value for the percentage complete indicator of the progress indicator (status bar) and specifies the current message to display on the top line of the progress indicator.

Setups that install files using the FeatureMoveData function must call SetStatusWindow before calling FeatureMoveData in order to set the percentage complete indicator to 0% and clear the top line of the indicator. No additional calls to SetStatusWindow are required. The 'Status Text' string for each feature is displayed on the top line of the status bar automatically while the files of the feature are being installed.

Before calling `FeatureMoveData`, your setup should also call the `StatusUpdate` function to enable automatic updating of the percentage complete indicator during file transfer. To enable the display of the name and path of the file being installed on the second line of the status bar, call `Enable` with the `INDVFILESTATUS` parameter before calling `FeatureMoveData`. After these calls, the percentage complete indicator is updated smoothly during file transfer and the names of individual files are displayed as they are transferred when the `FeatureMoveData` function is called.

Setups that install files using the `CopyFile` or `XCopyFile` functions may need to make multiple calls to `SetStatusWindow` in order to change the message on the top line of the indicator between successive calls to `CopyFile` or `XCopyFile`. `StatusUpdate` and `Enable` (with the parameter `INDVFILESTATUS`) work normally with `CopyFile` and `XCopyFile`. It is not necessary to call `SetStatusWindow` to change the percentage complete indicator between successive calls to `CopyFile` or `XCopyFile` if you have called `StatusUpdate` to enable automatic updating.

## Syntax

```
SetStatusWindow ( nPercent, szString );
```

## Parameters

**Table 163** • SetStatusWindow Parameters

Parameter	Description
<b>nPercent</b>	Specifies a value between 0 and 100 that represents the percentage to be displayed by the percentage complete indicator. To reset the indicator before calling <code>FeatureMoveData</code> , specify 0. To change the message displayed on the top line of the status bar without changing the percentage complete indicator, specify -1 in this parameter.
<b>szString</b>	Specifies a string to display on the top line of the status bar. Note that if a 'DisplayText' parameter has been specified for this feature (in the IDE), that string automatically overwrites any text specified in this parameter when <code>FeatureMoveData</code> is called.

## Return Values

This function does not return a value.

## SetStatusWindow Example

This function cannot be used in a Basic MSI installation.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SetStatusWindow function.
```

```

*
* SetStatusWindow is called to set the progress bar and
* to display text in the progress indicator.
*
* Note: Before running this script, create the directories
*       and file referenced by the preprocessor constants.
*
\*-----*/

#define SOURCE_DIR  "C:\\Source"
#define TARGET_DIR  "C:\\Target"
#define TARGET_FILE "ISExamp1.txt"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_SetStatusWindow(HWND);

function ExFn_SetStatusWindow(hMSI)
begin

    // Enable progress indicator.
    Enable (STATUS);

    // Set the progress bar to 33% and display a message.
    SetStatusWindow (33, "Now copying file...");

    // Delay for two seconds to ensure the window is displayed correctly.
    Delay (2);

    // Copy the file in the source directory to the target directory
    CopyFile (SOURCE_DIR ^ TARGET_FILE, TARGET_DIR ^ TARGET_FILE);

    // Set the progress bar to 66% and display a message.
    SetStatusWindow (66, "Now deleting file...");
    Delay (2);

    // Delete copied file in the Target directory.
    DeleteFile (TARGET_FILE);

    // Set the percent indicator to 100% and display a message.
    SetStatusWindow (100, "SetStatusWindow example completed.");
    Delay (2);

end;

```

## SetTitle



**Project** • This information does not apply to Basic MSI projects.

The **SetTitle** function displays a title either in the main window's title bar or inside the main window, depending on the value of nColor.



---

**Tip** • To set the background color of your setup, call *SetColor*.


To set the font and font style of the title displayed inside the background window, call *SetFont*.

## Syntax



```
SetTitle ( szTitle, nPointSize, nColor );
```

## Parameters

**Table 164 • SetTitle Parameters**

Parameter	Description
<b>szTitle</b>	<p>Specifies a title to appear either in the main window's title bar or inside the main window. If the title is to appear in the window's title bar, you must specify the predefined constant <code>BACKGROUNDCAPTION</code> in the third parameter. A title bar title that does not fit the available space will be displayed truncated on the right and terminated with ellipses. The default title bar title is "Setup".</p> <p></p> <p><b>Note •</b> The <code>BACKGROUNDCAPTION</code> option is supported for backward compatibility only. The recommended method for setting the main window's title bar is to set the value of the system variable <code>IFX_SETUP_TITLE</code>.</p> <p>When a color value is passed in the third parameter, the title is displayed left aligned at the top of the main window. Titles that are too wide for the main window will be displayed truncated on the right. To create a title that occupies more than one line, embed newline characters where you want the line to break.</p>
<b>nPointSize</b>	<p>Specifies the size, in points, of a title to be displayed in the main window. The suggested size is 24 points. Note that this parameter is ignored when the third parameter is <code>BACKGROUNDCAPTION</code>.</p>

**Table 164 • SetTitle Parameters (cont.)**

Parameter	Description
<b>nColor</b>	<p>Specifies either a color or the predefined constant BACKGROUNDCAPTION.</p> <p></p> <p><b>Note</b> • Calling <i>SetTitle</i> with the <i>nColor</i> set to <i>BACKGROUNDCAPTION</i> has no effect on setups that do not run in window mode. To run your setup in a standard window, you must first call <i>Enable</i> with the parameter <i>DEFWINDOWMODE</i> or <i>FULLWINDOWMODE</i> and then display the window by calling <i>Enable</i> with the parameter <i>BACKGROUND</i>.</p> <ul style="list-style-type: none"> <li>• To indicate that the value of <i>szTitle</i> should be displayed in the title bar of the main window, pass the predefined constant <i>BACKGROUNDCAPTION</i> in this parameter. When you specify <i>BACKGROUNDCAPTION</i>, <i>nPointSize</i> is ignored; the color and size of the title bar title are determined by the end user's system settings. Note that this option has no effect on setups that do not run in window mode.</li> <li>• To indicate that the value of <i>szTitle</i> should be displayed inside the setup window, specify a color for that title by passing one of the following constants (whose corresponding RGB values are given in parentheses) or calling the <i>RGB</i> function to indicate a particular custom color.</li> </ul> <p><b>BLACK</b>—(0, 0, 0)</p> <p><b>BLUE</b>—(0, 0, 255)</p> <p><b>GREEN</b>—(0, 255, 0)</p> <p><b>MAGENTA</b>—(255, 0, 127)</p> <p><b>RED</b>—(255, 0, 0)</p> <p><b>WHITE</b>—(255, 255, 255)</p> <p><b>YELLOW</b>—(255, 255, 0)</p> <p></p> <p><b>Note</b> • <i>Magenta</i> is not part of the standard Windows 16-color palette.</p>

## Return Values

**Table 165 • SetTitle Return Values**

Return Value	Description
0	Indicates that the function successfully set the title of the setup.
< 0	Indicates that the function was unable to set the title of the setup.

## SetTitle Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SetTitle function.
 *
 * This script displays a title in yellow, 24-point type.
 *
 \*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_SetTitle(HWND);

function ExFn_SetTitle(hMSI)
begin
    // Make the background window a standard maximized window.
    Enable (FULLWINDOWMODE);

    // Set the background color to blue.
    SetColor (BACKGROUND, BK_BLUE);

    // Display a title in the window. The newline character
    // in the title string forces "Example" to the next line.
    SetTitle ("SetTitle\nExample", 24, YELLOW);

    // Display the background window
    Enable (BACKGROUND);

    // Leave the window open for 3 seconds.
    Delay (3);

end;

```

## SetUpdateStatus

The **SetUpdateStatus** function is obsolete. If this function is called, it returns ISERR\_NOT\_IMPL.



## Syntax

```
void SetUpdateStatus( BOOL );
```

# SetUpdateStatusReboot

The **SetUpdateStatusReboot** function is obsolete. If this function is called, it returns ISERR\_NOT\_IMPL.

## Syntax

```
void SetUpdateStatusReboot( BOOL );
```

# SetupType




---

**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SetupType** function displays a dialog that enables the end user to select one of the three standard setup types: Typical, Compact, or Custom. These setup options are displayed with standard description text. If you want to add other setup types or change the displayed setup type names or descriptions, call [SdSetupTypeEx](#) instead.




---

**Caution** • If an end user returns to the Setup Type dialog after using a feature dialog to select and deselect features associated with the selected setup type, those choices are lost. This occurs because the SetupType function automatically resets the default feature selections for the selected setup type each time it is called.

## Syntax

```
SetupType ( szTitle, szMsg, szReserved, nType, nReserved );
```

## Parameters

**Table 166** • SetupType Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of this dialog. To display the default title Setup Type, pass a null string ("") in this parameter.
<b>szMsg</b>	Specifies the message you want to display at the top of the dialog. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>szReserved</b>	Pass a null string ("") in this parameter—no other value is allowed.
<b>nType</b>	Specifies the default setup type when the dialog is opened. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>TYPICAL</b>—Defines the default setup type as Typical.</li> <li>• <b>COMPACT</b>—Defines the default setup type as Compact.</li> <li>• <b>CUSTOM</b>—Defines the default setup type as Custom.</li> </ul>
<b>nReserved</b>	Pass zero in this parameter. No other value is allowed.

## Return Values

**Table 167** • SetupType Return Values

Return Value	Description
<b>TYPICAL (301)</b>	Indicates that the end user selected the Typical setup type.
<b>COMPACT (302)</b>	Indicates that the end user selected the Compact setup type.
<b>CUSTOM (303)</b>	Indicates that the end user selected the Custom setup type.
<b>BACK (12)</b>	Indicates that the end user clicked the Back button.

## Additional Information

- In an InstallScript project, if you do not display a setup type dialog, your script must do one of the following:
  - Select a setup type.
  - Call a feature selection dialog function such as SdFeatureTree.
  - Directly select features.
- To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SetupType Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SetupType function.
 *
 * Comments: To run this example script, create a project (or
 *           insert into a project) with several features and
 *           subfeatures with components containing files.
 *           This example includes setup of uninstallation
 *           functionality.
 *
 \*-----*/

// Specify your feature name here. These are the names you gave to your
// features in the IDE. A NULL ("" ) string specifies base features.
#define ASKDESTITITLE      "Choose Destination Location"
#define ASKDESTMSG        "Choose a destination location for the application."
#define SETUPTYPE_TITLE    "Choose Setup Type"
#define SETUPTYPEMSG       "Select a setup type."
#define FEATURE            ""
#define SDFEATDLGTITLE    "Feature Selection"
#define SDFEATDLGMSG       "Select features to install and destination location."
#define APPBASE_PATH       "Your Company\\Word Processor"
#define COMPANY_NAME       "Your Company"
#define PRODUCT_NAME       "Word Processor"
#define PRODUCT_VERSION    "1.0"
#define PRODUCT_KEY        "Word Processor"
#define DEINSTALL_KEY      "Word Processor"
#define UNINSTALL_NAME     "Word Processor"

    prototype HandleFeatureError (NUMBER);

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_SetupType(HWND);

function ExFn_SetupType(hMSI)
    STRING svLogFile;
    NUMBER nvDisk, nResult;
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Get the destination location.
    INSTALLDIR = PROGRAMFILES ^ APPBASE_PATH;
```

```

AskDestPath (ASKDESTITITLE, ASKDESTMSG, INSTALLDIR , 0);

// Get setup type and target location with SdSetupType.
INSTALLDIR = PROGRAMFILES ^ APPBASE_PATH;
nResult = SetupType (SETUPTYPEITITLE, SETUPTYPEMSG, "", TYPICAL, 0);

// If Custom setup type is selected, let user select features
// and change location if desired.
if (nResult = CUSTOM) then
    SdFeatureDialogAdv (SDFEATDLGTITLE, SDFEATDLGMSG,
                        INSTALLDIR, FEATURE);
endif;

// Set up uninstallation.
InstallationInfo (COMPANY_NAME, PRODUCT_NAME,
                  PRODUCT_VERSION, PRODUCT_KEY);

svLogFile = "Uninst.isu";
DeinstallStart (INSTALLDIR, svLogFile, DEINSTALL_KEY, 0);
RegDBSetItem (REGDB_UNINSTALL_NAME, UNINSTALL_NAME);

// Transfer files based on feature selection. Handle errors.
Enable (STATUSDLG);
Enable (INDVFILESTATUS);
StatusUpdate (ON, 100);
nResult = FeatureMoveData (MEDIA, nvDisk, 0);
HandleFeatureError (nResult);

Disable (INDVFILESTATUS);
Disable (STATUSDLG);

end;

/*-----*\
*
* Function:  HandleFeatureError
*
* Purpose:  This function evaluates the value returned by a feature
*           function and if the value is less than zero, displays the error
*           number and aborts the setup.
*
*\-----*/
function HandleFeatureError (nResult)
    NUMBER  nvError;
    STRING  svFeatureSource, svFeature, svComponent, svFile;
begin
    if (nResult < 0) then
        ComponentError (svFeatureSource, svFeature, svComponent, svFile, nvError);
        sprintfBox (INFORMATION, "Data Transfer Error Information",
                    "FeatureError returned the " +
                    "following data transfer error.\n" +
                    "Setup will now abort.\n\n" +
                    "Media Name: %s\nFeature: %s\nComponent: %s\n" +
                    "File: %s\nError Number: %ld",
                    svFeatureSource, svFeature, svComponent, svFile, nvError);
        abort;
    endif;
end;
end;

```

# SetupType2



---

**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SetupType2** function displays a dialog that enables the end user to select one of the two standard setup types: Complete or Custom. These setup options are displayed with standard description text. If you want to add other setup types or change the displayed setup type names or descriptions, call **SdSetupTypeEx** instead.



---

**Caution** • If an end user returns to the Setup Type dialog after using a feature dialog to select and deselect features associated with the selected setup type, those choices are lost. This occurs because the SetupType2 function automatically resets the default feature selections for the selected setup type each time it is called.

## Syntax

```
SetupType2 ( szTitle, szMsg, szReserved, nType, nReserved );
```

## Parameters

**Table 168** • SetupType2 Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of this dialog. To display the default title Setup Type2, pass a null string ("" ) in this parameter.
<b>szMsg</b>	Specifies the message you want to display at the top of the dialog. To display the default instructions for this dialog, pass a null string ("" ) in this parameter.
<b>szReserved</b>	Pass a null string ("" ) in this parameter. No other value is allowed.
<b>nType</b>	Specifies the default setup type when the dialog is opened. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>COMPLETE</b>—Defines the default setup type as Complete.</li> <li>• <b>CUSTOM</b>—Defines the default setup type as Custom.</li> </ul>
<b>nReserved</b>	Pass zero in this parameter. No other value is allowed.

## Return Values

**Table 169** • SetupType2 Return Values

Return Value	Description
<b>COMPLETE (304)</b>	Indicates that the Complete setup type was selected.
<b>CUSTOM (303)</b>	Indicates that the end user selected the Custom setup type.
<b>BACK (12)</b>	Indicates that the end user clicked the Back button.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the dialog could not be displayed.

## Additional Information

- In an InstallScript project, if you do not display a setup type dialog, your script must do one of the following:
  - Select a setup type.
  - Call a feature selection dialog function such as SdFeatureTree.
  - Directly select features.
- To view an example of this or other dialogs for your installation, use the Dialog Sampler. In InstallShield, on the Tools menu, point to InstallScript, then click Standard Dialog Sampler or Skinned Dialog Sampler.

## SetupType2 Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SetupType2 function.
 *
 * Comments: To run this example script, create a project (or
 *           insert into a project) with several features and
 *           subfeatures with components containing files.
 *           This example includes setup of uninstallation
 *           functionality.
 *
 \*-----*/

// Specify your feature name here. These are the names you gave to
// your features in InstallShield. A NULL ("") string
// specifies base features.
#define ASKDESTITLE      "Choose Destination Location"
#define ASKDESTMMSG      "Choose a destination location for the application."
#define SETUPTYPETITLE   "Choose Setup Type"
#define SETUPTYPEMSG     "Select a setup type."
#define FEATURE          ""
#define SDFEATDLGTITLE   "Feature Selection"
#define SDFEATDLGMSG     "Select features to install and destination location."
#define APPBASE_PATH     "Your Company\\Word Processor"
#define COMPANY_NAME     "Your Company"
#define PRODUCT_NAME     "Word Processor"
#define PRODUCT_VERSION  "1.0"
#define PRODUCT_KEY      "Word Processor"
#define DEINSTALL_KEY    "Word Processor"
#define UNINSTALL_NAME   "Word Processor"

    prototype HandleFeatureError (NUMBER);

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_SetupType2(HWND);

function ExFn_SetupType2(hMSI)
    STRING svLogFile;
    NUMBER nvDisk, nResult;
begin

    // Disable the Back button in setup dialogs.
    Disable (BACKBUTTON);

    // Get the destination location.
```

```

INSTALLDIR = PROGRAMFILES ^ APPBASE_PATH;
AskDestPath (ASKDESTITUTE, ASKDESTMMSG, INSTALLDIR , 0);

// Get setup type and target location with SdSetupType2.
INSTALLDIR = PROGRAMFILES ^ APPBASE_PATH;
nResult = SetupType2 (SETUPTYPETITLE, SETUPTYPEMSG, "", COMPLETE, 0);

// If Custom setup type is selected, let user select features
// and change location if desired.
if (nResult = CUSTOM) then
    SdFeatureDialogAdv (SDFEATDLGTITLE, SDFEATDLGMSG,
                        INSTALLDIR, FEATURE);
endif;

// Set up uninstallation.
InstallationInfo (COMPANY_NAME, PRODUCT_NAME,
                  PRODUCT_VERSION, PRODUCT_KEY);

svLogFile = "Uninst.isu";
DeinstallStart (INSTALLDIR, svLogFile, DEINSTALL_KEY, 0);
RegDBSetItem (REGDB_UNINSTALL_NAME, UNINSTALL_NAME);

// Transfer files based on feature selection. Handle errors.
Enable (STATUSDLG);
Enable (INDVFILESTATUS);
StatusUpdate (ON, 100);
nResult = FeatureMoveData (MEDIA, nvDisk, 0);
HandleFeatureError (nResult);

Disable (INDVFILESTATUS);
Disable (STATUSDLG);

end;

/*-----*\
*
* Function:  HandleFeatureError
*
* Purpose:  This function evaluates the value returned by a feature
*           function and if the value is less than zero, displays the error
*           number and aborts the installation.
*
*-----*/
function HandleFeatureError (nResult)
    NUMBER  nvError;
    STRING  svFeatureSource, svFeature, svComponent, svFile;
begin
    if (nResult < 0) then
        ComponentError (svFeatureSource, svFeature, svComponent, svFile, nvError);
        sprintfBox (INFORMATION, "Data Transfer Error Information",
                    "FeatureError returned the " +
                    "following data transfer error.\n" +
                    "Setup will now abort.\n\n" +
                    "Media Name: %s\nFeature: %s\nComponent: %s\n" +
                    "File: %s\nError Number: %ld",
                    svFeatureSource, svFeature, svComponent, svFile, nvError);
        abort;
    endif;
end;
end;

```



# ShowObjWizardPages



**Project** • This information applies to InstallScript projects.

The **ShowObjWizardPages** function is called in a project's UI event handler to execute each included object's corresponding UI event handler. For example, calling **ShowObjWizardPages** in a project's OnFirstUIBefore event handler executes each included object's OnFirstUIBefore code.

## Syntax

```
ShowObjWizardPages ( nDirection );
```

## Parameters

**Table 170** • ShowObjWizardPages Parameters

Parameter	Description
<b>nDirection</b>	<p>Pass one of the following predefined constants to indicate the measurement unit:</p> <ul style="list-style-type: none"> <li> <b>NEXT</b>—Specifies the following:            The first UI event handler called is that of the object listed first in the project's Components pane, then that of the object listed second, and so on.             The <a href="#">WizardDirection</a> function, when called in an object's UI event handler, returns the value NEXT.         </li> <li> <b>BACK</b>—Specifies the following:            The first UI event handler called is that of the object listed last in the project's Components pane, then that of the object listed next to last, and so on.             The <a href="#">WizardDirection</a> function, when called in an object's UI event handler, returns the value BACK.         </li> </ul>

## Return Values

The value returned by the most recently executed object UI event handler.

## Additional Information

The objects' dialogs are displayed in one of two possible orders, depending on the value of nDirection. For greater control over the order in which the objects' dialogs are displayed, call the objects' ShowxxxxUlyyyy methods. For more information on object UI event handlers, see [Creating the Object's Run-Time User Interface](#).

# ShowProgramFolder

The **ShowProgramFolder** function displays a program folder.

## Syntax

```
ShowProgramFolder ( szFolder, nCommand );
```

## Parameters

**Table 171** • ShowProgramFolder Parameters

Parameter	Description
<b>szFolder</b>	Specifies the name of the folder to display.
<b>nCommand</b>	Specifies the state of the folder. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>SW_SHOW</b>—Show folder in normal state.</li> <li>• <b>SW_MAXIMIZE</b>—Maximize the folder.</li> <li>• <b>SW_MINIMIZE</b>—Minimize the folder.</li> <li>• <b>SW_RESTORE</b>—Restore the folder to original size.</li> </ul>

## Return Values

This function does not return a value.

## ShowProgramFolder Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the ShowProgramFolder function.
*
* ShowProgramFolder displays a folder, then changes the state
* of the folder.
*
* Note: This script will not run properly if the program folder
*       "Startup" does not exist. Either create a program
*       folder called "Startup" or set the constant FOLDER so
*       that it references an existing program folder. In
*       addition, the specified folder should be closed or
*       minimized when you run this script.
*
\*-----*/

#define FOLDER "Startup"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"
```

```

export prototype ExFn_ShowProgramFolder(HWND);

function ExFn_ShowProgramFolder(hMSI)
begin

    // Display the specified folder.
    ShowProgramFolder (FOLDER, SW_SHOW);
    Delay (3);

    // Maximize the folder.
    ShowProgramFolder (FOLDER, SW_MAXIMIZE);
    Delay (3);

    // Restore the folder to its previous state.
    ShowProgramFolder (FOLDER, SW_RESTORE);
    Delay (3);

    // Minimize the folder.
    ShowProgramFolder (FOLDER, SW_MINIMIZE);
    Delay (3);

    // Restore the folder to its previous state.
    ShowProgramFolder (FOLDER, SW_RESTORE);
    Delay (3);

end;

```

## ShowWindow

The **ShowWindow** function sets the specified window's show state.

### Syntax

```

BOOL ShowWindow(
    HWND hWnd, // handle to window
    int nCmdShow // show state
);

```

## Parameters

**Table 172** • ShowWindow Parameters

Parameter	Description
<b>hWnd</b>	Specifies the handle to the window.

**Table 172** • ShowWindow Parameters (cont.)

Parameter	Description
<b>nCmdShow</b>	<p>Specifies the show state of the window. If the program that launched the application includes a STARTUPINFO structure, this parameter is ignored the first time the application calls ShowWindow. Otherwise, the first time ShowWindow is called, the value should be the value obtained by the WinMain function in its <i>nCmdShow</i> parameter. In subsequent calls, this parameter can be one of the following values.</p> <ul style="list-style-type: none"> <li>• <b>SW_FORCEMINIMIZE</b>—Minimizes a window, even if the thread that owns the window is not responding. This value should only be used when minimizing windows from a different thread.</li> <li>• <b>SW_HIDE</b>—Hides the window and activates another window.</li> <li>• <b>SW_MAXIMIZE</b>—Maximizes the specified window.</li> <li>• <b>SW_MINIMIZE</b>—Minimizes the specified window and activates the next top-level window in the Z order. The Z order is the position of a window in a stack of overlapping windows on the z axis, which indicates depth.</li> <li>• <b>SW_RESTORE</b>—Activates and displays the window. If the window is minimized or maximized, the system restores it to its original size and position. An application should specify this value when restoring a minimized window.</li> <li>• <b>SW_SHOW</b>—Activates the window and displays it in its current size and position.</li> <li>• <b>SW_SHOWDEFAULT</b>—Sets the show state based on the SW_ value specified in the STARTUPINFO structure passed to the CreateProcess function by the program that started the application.</li> <li>• <b>SW_SHOWMAXIMIZED</b>—Activates the window and displays it as a maximized window.</li> <li>• <b>SW_SHOWMINIMIZED</b>—Activates the window and displays it as a minimized window.</li> <li>• <b>SW_SHOWMINNOACTIVE</b>—Displays the window as a minimized window. The active window remains active.</li> <li>• <b>SW_SHOWNA</b>—Displays the window in its current size and position. The active window remains active.</li> <li>• <b>SW_SHOWNOACTIVATE</b>—Displays a window in its most recent size and position. The active window remains active.</li> <li>• <b>SW_SHOWNORMAL</b>—Activates and displays a window. If the window is minimized or maximized, it is restored it to its original size and position. An application should specify this value when displaying the window for the first time.</li> </ul>

## Return Values

**Table 173** • ShowWindow Return Values

Return Value	Description
<b>Nonzero</b>	The return value is nonzero if the window was visible.
<b>0</b>	The return value is 0 if the window was hidden.

## Additional Information

This information was adapted from the MSDN topic [ShowWindow Function](#).

# SilentReadData

The **SilentReadData** function instructs InstallShield Silent on how to read the .iss file dialog data for a custom dialog when an installation runs in silent mode (when using the -s switch with **Setup.exe**). Note that you can create an .iss file by calling **SilentWriteData**.

To use **SilentReadData** in your script, construct the logic so that it first checks to make sure that the installation is running in silent mode. Place the **SilentReadData** function call inside an if-else statement, based on a test of the system variable MODE, as shown below:

```
if (MODE=SILENTMODE) then
    // Call SilentReadData here
else
    // Make a normal, non-silent function call here
endif;
```

Custom dialogs can be resources that you call and handle in your installation script using functions like **EzDefineDialog** and **WaitOnDialog**, or they can be completely external, executed as calls to functions in DLLs. In either case, you must use **SilentReadData** to retrieve from the .iss file the dialog button's return value (Next, Back, Cancel, and so on) and any values set or returned in variables.

## Syntax

```
SilentReadData (szSection, szValName, nValType, svVal, nvVal);
```

## Parameters

**Table 174 • SilentReadData Parameters**


Parameter	Description
<b>szSection</b>	Specifies the name of the dialog data section in the .iss file. Do not include the square brackets ( [ ] ). The parameter szSection takes the form <functionname>-<number>, where <functionname> is the name of the dialog function as it is used in the script, and <number> is the number of the occurrence of that dialog in the script, beginning with 0 (zero). For example, the first occurrence of the MyDialog function dialog would have a value of "MyDialog-0" in szSection, the second occurrence "MyDialog-1," the third "MyDialog-2," and so on.
<b>szValName</b>	Specifies the value name that appears in the dialog data section of the .iss file. Every dialog has at least one value for szValName—"Result"—which identifies the value returned by the dialog button controls (BACK, NEXT, OK, or CANCEL). Other value names are used to identify values and data associated with the other dialog controls.
<b>nValType</b>	Identifies the data type of the value assigned to the value name in szValName. The value itself is stored in either svVal or nvVal, depending upon the value of nValType. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>DATA_STRING</b>—The value assigned to the value name in szValName is of type STRING. Its value will be stored in svVal.</li> <li>• <b>DATA_NUMBER</b>—The value assigned to the value name in szValName is of type NUMBER. Its value will be stored in nvVal.</li> <li>• <b>DATA_COMPONENT</b>—The value assigned to the value name in szValName is the name of a component It will be stored in svVal.</li> <li>• <b>DATA_LIST</b>—The value assigned to the value name in szValName is the list ID for an InstallScript list. It will be stored in nvVal.</li> </ul>
<b>svVal</b>	Specifies the value assigned to the value name in szValName when nValType is DATA_STRING or DATA_COMPONENT.
<b>nvVal</b>	Specifies the value assigned to the value name in szValName when nValType is DATA_NUMBER or DATA_LIST.

## Return Values

**Table 175 • SilentReadData Return Values**

Return Value	Description
<b>0</b>	<b>SilentReadData</b> successfully instructed InstallShield Silent on how to read the dialog data for the custom dialog.

**Table 175 • SilentReadData Return Values (cont.)**

Return Value	Description
< 0	<b>SilentReadData</b> was called in an installation that is not running silently.
	 <p><b>Note •</b> If <b>SilentReadData</b> cannot read the custom dialog data in the response file, the installation is aborted.</p>

## SilentReadData Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the functions SdMakeName, SilentReadData, and
 * SilentWriteData.
 *
 * This example script shows how to handle a custom dialog
 * in a silent installation. The resource .dll for the example
 * custom dialog shown below should be stored in a compressed
 * form under the Support Files/Billboards view of InstallShield.
 *
 * The example dialog was built from the custom dialog
 * template provided with InstallShield.
 *
 * Dialog control IDs and other information are included in
 * the RESOURCE.H file (not shown). This file, which is included
 * in the first line of the example, must be inserted in the
 * InstallScript view of InstallShield.
 *
 * The example creates a text file called Cominit.txt. If the
 * installation runs in silent mode, SilentReadData is called
 * and the custom dialog control selections are read from
 * the .ISS file. The selections are then saved in the file
 * Cominit.txt as a means of demonstrating that they were
 * successfully read from the .iss file. If the installation
 * runs in normal mode, the custom dialog is displayed
 * and the selections are recorded in the .iss file and
 * displayed in message boxes. The initial .ISS file text is
 * shown after the example script.
 *
 \*-----*/

#include "Resource.h"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_SilentReadData(HWND);

```



```

function ExFn_SilentReadData(hMSI)
    BOOL bDone;
    STRING svSection, svComPort, svPulse, svTone, svDial9, svVal;
    NUMBER nvCommDialog, nCmdValue, nPulseState, nToneState;
    NUMBER nDial9State, nResult, nvHandle;
    LIST listID;
    HWND hwndDlg;
begin

    // Open the text file COMINIT.TXT so custom dialog selections
    // from the .ISS file can be stored in it.
    OpenFileMode (FILE_MODE_APPEND);
    OpenFile (nvHandle, "c:\\rul", "cominit.txt");

    // If operating in silent mode, then read from the .ISS file.
    if (MODE=SILENTMODE) then
        SdMakeName (svSection, "COMM_DIALOG", "", nvCommDialog);

        SilentReadData (svSection, "Result", DATA_NUMBER, svVal, nResult);

        if (nResult = 1) then
            // Read the data from the .ISS file. For purposes of
            // writing the results to a text file, read the
            // data as strings.
            SilentReadData (svSection, "nPulseState", DATA_STRING,
                            svPulse, nResult);

            SilentReadData (svSection, "nToneState", DATA_STRING,
                            svTone, nResult);

            SilentReadData (svSection, "nDial9State", DATA_STRING,
                            svDial9, nResult);

            // Store the custom dialog selections in
            // the text file COMINIT.TXT.
            svVal = "Pulse box is: " ^ svPulse;
            WriteLine(nvHandle, svVal);

            svVal = "Tone box is: " ^ svTone;
            WriteLine(nvHandle, svVal);

            svVal = "Dial9 box is: " ^ svDial9;
            WriteLine(nvHandle, svVal);
        endif;

    // If not in silent mode, then call and handle the custom dialog
    // as you normally would.
    else
        listID = ListCreate (STRINGLIST);
        ListAddString (listID, "COMM1:", AFTER);
        ListAddString (listID, "COMM2:", AFTER);
        ListAddString (listID, "COMM3:", AFTER);
        ListAddString (listID, "COMM4:", AFTER);

        EzDefineDialog ("MYCOMDIALOG", SUPPORTDIR^"RESOURCE.DLL",
                        "COMM_DIALOG", 0);

        bDone = FALSE;
    end
end

```

```

while (bDone=FALSE)
    nCmdValue = WaitOnDialog ("MYCOMDIALOG");

    switch (nCmdValue)
    case DLG_INIT:
        // Initialize the back, next, and cancel button enable/disable states
        // for this dialog and replace %P, %VS, %VI with
        // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
        // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724
        // and 202.
        hwndDlg = CmdGetHwndDlg("MYCOMDIALOG");
        SdGeneralInit("MYCOMDIALOG", hwndDlg, 0, "");
        CtrlSetState ("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);
        CtrlSetList ("MYCOMDIALOG", ID_COMPORT, listID);
        CtrlSetState ("MYCOMDIALOG", ID_DIAL9, BUTTON_CHECKED);
    case OK:
        CtrlGetCurSel ("MYCOMDIALOG", ID_COMPORT, svComPort);
        nPulseState = CtrlGetState ("MYCOMDIALOG", ID_PULSE);
        nToneState = CtrlGetState ("MYCOMDIALOG", ID_TONE);
        nDial9State = CtrlGetState ("MYCOMDIALOG", ID_DIAL9);
        nResult = NEXT;
        bDone = TRUE;
    case BACK:
        nResult = BACK;
        bDone = TRUE;
    case RES_PBUT_CANCEL:
        // The user clicked the Cancel button.
        Do (EXIT);
    case DLG_CLOSE:
        // The user clicked the window's close button.
        Do (EXIT);
    case ID_PULSE:
        nPulseState = CtrlGetState ("MYCOMDIALOG", ID_PULSE);

        if (nPulseState = BUTTON_CHECKED) then
            CtrlSetState ("MYCOMDIALOG", ID_TONE, BUTTON_UNCHECKED);
            CtrlSetState ("MYCOMDIALOG", ID_PULSE, BUTTON_CHECKED);
        else
            CtrlSetState ("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);
            CtrlSetState ("MYCOMDIALOG", ID_PULSE, BUTTON_UNCHECKED);
        endif;
    case ID_TONE:
        nToneState = CtrlGetState ("MYCOMDIALOG", ID_TONE);

        if (nPulseState = BUTTON_CHECKED) then
            CtrlSetState ("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);
            CtrlSetState ("MYCOMDIALOG", ID_PULSE, BUTTON_UNCHECKED);
        else
            CtrlSetState ("MYCOMDIALOG", ID_TONE, BUTTON_UNCHECKED);
            CtrlSetState ("MYCOMDIALOG", ID_PULSE, BUTTON_CHECKED);
        endif;
    case DLG_ERR:
        MessageBox ("Unable to display dialog. Setup canceled.", SEVERE);
        abort;
    endswitch;

endwhile;

```

```

EndDialog ("MYCOMDIALOG");
ReleaseDialog ("MYCOMDIALOG");

SdMakeName (svSection, "COMM_DIALOG", "", nvCommDialog);

SilentWriteData (svSection, "nPulseState", DATA_NUMBER,
                 svPulse, nPulseState);

SilentWriteData (svSection, "nToneState", DATA_NUMBER,
                 svTone, nToneState);
SilentWriteData (svSection, "nDial9State", DATA_NUMBER,
                 svDial9, nDial9State);

if (nPulseState = BUTTON_CHECKED) then
    MessageBox ("The Pulse button was checked.", INFORMATION);
else
    MessageBox ("The Pulse button was unchecked.", INFORMATION);
endif;

if (nToneState = BUTTON_CHECKED) then
    MessageBox ("The Tone button was checked.", INFORMATION);
else
    MessageBox ("The Tone button was unchecked.", INFORMATION);
endif;

if (nDial9State = BUTTON_CHECKED) then
    MessageBox ("The Dial9 button was checked.", INFORMATION);
else
    MessageBox ("The Dial9 button was unchecked.", INFORMATION);
endif;

endif;

// Close the text file COMINIT.TXT
CloseFile (nvHandle);

end;

/*The following is the initial .iss file text for the above example, where <PRODUCT_GUID> represents
your project's GUID, including the
surrounding braces. Note that -1001 is the numeric value of BUTTON_CHECKED and -1002 is the numeric value
of BUTTON_UNCHECKED.

[InstallShield Silent]
Version=v9.00
File=Response File
[Application]
Name=MyDialog
Version=4.0
Company=My Software Company
[<PRODUCT_GUID>-DlgOrder]
Dlg0=<PRODUCT_GUID>-COMM_DIALOG-0
Count=1
[<PRODUCT_GUID>-COMM_DIALOG-0]
nPulseState=-1001
nToneState=-1002
nDial9State=-1001
Result=1

```

\*/

## SilentWriteData

The **SilentWriteData** function records selections made in custom dialogs during the installation. This selection data is written to an .iss file for use by InstallShield Silent. To write to an .iss file during an installation, run the installation using the -r switch with Setup.exe.

Custom dialogs can be resources that you call and handle in your installation script using functions like EzDefineDialog and WaitOnDialog, or they can be completely external, executed as calls to functions in DLLs. In either case, you must use SilentWriteData to record the dialog's button return value (Next, Back, Cancel, and so on) and any values set or returned in variables.

### Syntax

```
SilentWriteData ( szSection, szValName, nValType, szVal, nVal );
```

## Parameters

**Table 176 • SilentWriteData Parameters**

Parameter	Description
<b>szSection</b>	Specifies the name of the dialog data section in the .iss file. Do not include the square brackets ( [ ] ). The parameter szSection takes the form <functionname>-<number>, where <functionname> is the name of the dialog function as it is used in the script, and <number> is the number of the occurrence of that dialog in the script, beginning with 0 (zero). For example, the first occurrence of the MyDialog function dialog would have a value of "MyDialog-0" for szSection, the second occurrence "MyDialog-1," the third "MyDialog-2," and so on.
<b>szValName</b>	Specifies the value name that appears in the dialog data section of the .iss file. Every dialog has at least one value for szValName—"Result"—which identifies the value returned by the dialog button controls (BACK, NEXT, OK, or CANCEL). Other value names are used to identify values and data associated with the other dialog controls.
<b>nValType</b>	Identifies the data type of the value assigned to the value name in szValName. The value itself is stored in either szVal or nVal, depending upon the value of nValType. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>DATA_STRING</b>—The value assigned to the value name in szValName is of type STRING. Its value will be stored in szVal.</li> <li>• <b>DATA_NUMBER</b>—The value assigned to the value name in szValName is of type NUMBER. Its value will be stored in nVal.</li> <li>• <b>DATA_COMPONENT</b>—The value assigned to the value name in szValName is the name of a component. It will be stored in szVal.</li> <li>• <b>DATA_LIST</b>—The value assigned to the value name in szValName is the list ID for an InstallShield list. It is stored in nVal.</li> </ul>
<b>szVal</b>	Specifies the value assigned to the value name in szValName when nValType is DATA_STRING or DATA_COMPONENT.
<b>nVal</b>	Specifies the value assigned to the value name in szValName when nValType is DATA_NUMBER or DATA_LIST.

## Return Values

**Table 177** • SilentWriteData Return Values

Return Value	Description
0	SilentWriteData successfully wrote the dialog data for the custom dialog to Setup.iss.
< 0	SilentWriteData was unable to write the dialog data for the custom dialog to Setup.iss.

## SilentWriteData Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the functions SdMakeName, SilentReadData, and
* SilentWriteData.
*
* This example script shows how to handle a custom dialog
* in a silent installation. The resource .dll for the example
* custom dialog shown below should be stored in a compressed
* form under the Support Files/Billboards view of InstallShield.
*
* The example dialog was built from the custom dialog
* template provided with InstallShield.
*
* Dialog control IDs and other information are included in
* the RESOURCE.H file (not shown). This file, which is included
* in the first line of the example, must be inserted in the
* InstallScript view of InstallShield.
*
* The example creates a text file called Cominit.txt. If the
* installation runs in silent mode, SilentReadData is called
* and the custom dialog control selections are read from
* the .ISS file. The selections are then saved in the file
* Cominit.txt as a means of demonstrating that they were
* successfully read from the .iss file. If the installation
* runs in normal mode, the custom dialog is displayed
* and the selections are recorded in the .iss file and
* displayed in message boxes. The initial .ISS file text is
* shown after the example script.
*
\*-----*/
```

```
#include "Resource.h"
```

```
// Include Ifx.h for built-in InstallScript function prototypes.
```

```

#include "Ifx.h"

export prototype ExFn_SilentWriteData(HWND);

function ExFn_SilentWriteData(hMSI)
    BOOL bDone;
    STRING svSection, svComPort, svPulse, svTone, svDial9, svVal;
    NUMBER nvCommDialog, nCmdValue, nPulseState, nToneState;
    NUMBER nDial9State, nResult, nvHandle;
    LIST listID;
    HWND hwndDlg;
begin

    // Open the text file COMINIT.TXT so custom dialog selections
    // from the .ISS file can be stored in it.
    OpenFileMode (FILE_MODE_APPEND);
    OpenFile (nvHandle, "c:\\rul", "cominit.txt");

    // If operating in silent mode, then read from the .ISS file.
    if (MODE=SILENTMODE) then
        SdMakeName (svSection, "COMM_DIALOG", "", nvCommDialog);

        SilentReadData (svSection, "Result", DATA_NUMBER, svVal, nResult);

        if (nResult = 1) then

            // Read the data from the .ISS file. For purposes of
            // writing the results to a text file, read the
            // data as strings.
            SilentReadData (svSection, "nPulseState", DATA_STRING,
                           svPulse, nResult);

            SilentReadData (svSection, "nToneState", DATA_STRING,
                           svTone, nResult);

            SilentReadData (svSection, "nDial9State", DATA_STRING,
                           svDial9, nResult);

            // Store the custom dialog selections in
            // the text file COMINIT.TXT.
            svVal = "Pulse box is: " ^ svPulse;
            WriteLine(nvHandle, svVal);

            svVal = "Tone box is: " ^ svTone;
            WriteLine(nvHandle, svVal);

            svVal = "Dial9 box is: " ^ svDial9;
            WriteLine(nvHandle, svVal);
        endif;

    // If not in silent mode, then call and handle the custom dialog
    // as you normally would.
    else
        listID = ListCreate (STRINGLIST);
        ListAddString (listID, "COMM1:", AFTER);
        ListAddString (listID, "COMM2:", AFTER);
        ListAddString (listID, "COMM3:", AFTER);
        ListAddString (listID, "COMM4:", AFTER);
    endif;
endfunction

```

```

EzDefineDialog ("MYCOMDIALOG", SUPPORTDIR^"RESOURCE.DLL",
               "COMM_DIALOG",0);

bDone = FALSE;

while (bDone=FALSE)
    nCmdValue = WaitOnDialog ("MYCOMDIALOG");

    switch (nCmdValue)
        case DLG_INIT:
            // Initialize the back, next, and cancel button enable/disable states
            // for this dialog and replace %P, %VS, %VI with
            // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
            // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724
            // and 202.
            hwndDlg = CmdGetHwndDlg("MYCOMDIALOG");
            SdGeneralInit("MYCOMDIALOG", hwndDlg, 0, "");
            CtrlSetState ("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);
            CtrlSetList ("MYCOMDIALOG", ID_COMPORT, listID);
            CtrlSetState ("MYCOMDIALOG", ID_DIAL9, BUTTON_CHECKED);
        case OK:
            CtrlGetCurSel ("MYCOMDIALOG", ID_COMPORT, svComPort);
            nPulseState = CtrlGetState ("MYCOMDIALOG", ID_PULSE);
            nToneState = CtrlGetState ("MYCOMDIALOG", ID_TONE);
            nDial9State = CtrlGetState ("MYCOMDIALOG", ID_DIAL9);
            nResult = NEXT;
            bDone = TRUE;
        case BACK:
            nResult = BACK;
            bDone = TRUE;
        case RES_PBUT_CANCEL:
            // The user clicked the Cancel button.
            Do (EXIT);
        case DLG_CLOSE:
            // The user clicked the window's close button.
            Do (EXIT);
        case ID_PULSE:
            nPulseState = CtrlGetState ("MYCOMDIALOG", ID_PULSE);

            if (nPulseState = BUTTON_CHECKED) then
                CtrlSetState ("MYCOMDIALOG", ID_TONE, BUTTON_UNCHECKED);
                CtrlSetState ("MYCOMDIALOG", ID_PULSE, BUTTON_CHECKED);
            else
                CtrlSetState ("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);
                CtrlSetState ("MYCOMDIALOG", ID_PULSE, BUTTON_UNCHECKED);
            endif;
        case ID_TONE:
            nToneState = CtrlGetState ("MYCOMDIALOG", ID_TONE);

            if (nPulseState = BUTTON_CHECKED) then
                CtrlSetState ("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);
                CtrlSetState ("MYCOMDIALOG", ID_PULSE, BUTTON_UNCHECKED);
            else
                CtrlSetState ("MYCOMDIALOG", ID_TONE, BUTTON_UNCHECKED);
                CtrlSetState ("MYCOMDIALOG", ID_PULSE, BUTTON_CHECKED);
            endif;
        case DLG_ERR:
            MessageBox ("Unable to display dialog. Setup canceled.", SEVERE);
            abort;

```



```

        endswitch;

    endwhile;

    EndDialog ("MYCOMDIALOG");
    ReleaseDialog ("MYCOMDIALOG");

    SdMakeName (svSection, "COMM_DIALOG", "", nvCommDialog);

    SilentWriteData (svSection, "nPulseState", DATA_NUMBER,
        svPulse, nPulseState);

    SilentWriteData (svSection, "nToneState", DATA_NUMBER,
        svTone, nToneState);
    SilentWriteData (svSection, "nDial9State", DATA_NUMBER,
        svDial9, nDial9State);

    if (nPulseState = BUTTON_CHECKED) then
        MessageBox ("The Pulse button was checked.", INFORMATION);
    else
        MessageBox ("The Pulse button was unchecked.", INFORMATION);
    endif;

    if (nToneState = BUTTON_CHECKED) then
        MessageBox ("The Tone button was checked.", INFORMATION);
    else
        MessageBox ("The Tone button was unchecked.", INFORMATION);
    endif;

    if (nDial9State = BUTTON_CHECKED) then
        MessageBox ("The Dial9 button was checked.", INFORMATION);
    else
        MessageBox ("The Dial9 button was unchecked.", INFORMATION);
    endif;

endif;

// Close the text file COMINIT.TXT
CloseFile (nvHandle);

end;

/*The following is the initial .iss file text for the above example, where <PRODUCT_GUID> represents
your project's GUID, including the
surrounding braces. Note that -1001 is the numeric value of BUTTON_CHECKED and -1002 is the numeric value
of BUTTON_UNCHECKED.

[InstallShield Silent]
Version=v9.00
File=Response File
[Application]
Name=MyDialog
Version=4.0
Company=My Software Company
[<PRODUCT_GUID>-DlgOrder]
Dlg0=<PRODUCT_GUID>-COMM_DIALOG-0
Count=1
[<PRODUCT_GUID>-COMM_DIALOG-0]

```

```

nPulseState=-1001
nToneState=-1002
nDial9State=-1001
Result=1
*/

```

## SizeOf

The **SizeOf** function retrieves the number of elements in an InstallScript array, or the size of a variable passed as an argument.

To determine the number of characters in the value of a string variable, use [StrLength](#).

### Syntax

```
SizeOf ( Item );
```

### Parameters

**Table 178 • SizeOf Parameters**

Parameter	Description
<b>Item</b>	Specifies the name of the variable.

### Return Values

**SizeOf** returns the number of elements in an array, or the size of a variable passed as an argument.

## SizeWindow

Use the **SizeWindow** function to change the size of a specific user interface element. Specify the new size in pixels.

The installation may run under many different screen resolutions. To account for this, you need to use the **GetExtents** function to determine the overall size of the screen; then use ratios in the parameters of your **SizeWindow** function call to specify the size of your user interface object.




**Note** • *This function is recommended for advanced developers only.*

### Syntax

```
SizeWindow (nObject, nDx, nDy);
```

## Parameters

**Table 179 • SizeWindow Parameters**

Parameter	Description
<b>nObject</b>	<p>Specifies the object to resize. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>BACKGROUND</b>—Indicates the main background window.</li> <li>• <b>METAFILE</b>—Indicates a billboard used during the file transfer process. <b>SizeWindow</b> does not support bitmap (.bmp) files.</li> </ul> <p></p> <p><b>Note •</b> This parameter does not have any effect on metafiles that are displayed with <i>SdBitmap</i> function. <i>SdBitmap</i> automatically resizes metafiles when they are displayed.</p> <p>In addition, this parameter does not have any effect on the type of billboard that is displayed on a progress dialog when <i>Enable(BBRD)</i> is called.</p> <ul style="list-style-type: none"> <li>• <b>MMEDIA_AVI</b>—Sets the window size for next AVI file to be played. All the AVI videos are displayed with a default size. Changing the size may change the resolution and crispness of the video.</li> <li>• <b>MMEDIA_SWF</b>—Sets the window size for the Adobe Flash application file (.swf) to be played.</li> </ul>
<b>nDx</b>	Specifies the horizontal size of the object in pixels.
<b>nDy</b>	Specifies the vertical size of the object in pixels.

## Return Values

**Table 180 • SizeWindow Return Values**

Return Value	Description
<b>0</b>	The function successfully changed the size of the window.
<b>&lt; 0</b>	The function was unable to change the size of the window.

## Additional Information

Use the **PlayMMedia** function if you want your installation to play an Adobe Flash application file (.swf) or an AVI file.

## SizeWindow Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the SizeWindow function.
 *
 * GetExtents is called to retrieve the extents of the screen.
 * SizeWindow is then called to resize the background to half
 * the original size.
 *
 \*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_SizeWindow(HWND);

function ExFn_SizeWindow(hMSI)
    NUMBER nDx, nDy, nObject;
begin

    // Enable the background.
    Enable (BACKGROUND);

    MessageBox ("Background at original size.", INFORMATION);

    // Get the extents of the screen.
    GetExtents (nDx, nDy);

    // Set the object to be resized.
    nObject = BACKGROUND;

    // Resize the background window to half of its original size.
    if (SizeWindow (nObject, (nDx / 2), (nDy / 2)) < 0) then
        MessageBox ("SizeWindow failed.", SEVERE);
    endif;

    MessageBox ("Background after call to SizeWindow.", INFORMATION);

end;
```

## Sprintf

The **Sprintf** function creates a string from variable data using format specifier and matching variables. Sprintf works like the Microsoft Windows API `wsprintf`.

### Syntax

```
Sprintf ( svResult, szFormat [,arg] [,...] );
```

## Parameters

**Table 181** • Sprintf Parameters

Parameter	Description
<b>svResult</b>	Returns a string created from the arguments passed in the third and subsequent parameters and formatted according the specifications in the second parameter, szFormat.
<b>szFormat</b>	Specifies a string than can include literal text and must include one format specifier for each argument to be embedded in the string returned by svResult.
<b>arg</b>	<p>You may specify up to nine arguments to be included in the message. You must have one argument for each format specifier in the message; the type of each argument must match the type of its respective format specifier.</p> <p>Sprintf generates a compiler error or fail at run time under the following conditions:</p> <ul style="list-style-type: none"> <li>• More than nine format specifiers and arguments are specified. This results in a compiler error.</li> <li>• The number of arguments does not match the number of format specifiers. When a specifier does not have a corresponding argument, the resulting string contains unpredictable characters in the specifier's position. When there are more arguments than specifiers, the excess arguments are not inserted into the resulting string.</li> <li>• A variable does not match the type of its respective format specifier. The resulting string contains unpredictable characters in the specifier's position.</li> </ul>

## Return Values

If the Sprintf function is successful, the return value is the length—the number of characters—in the string stored in the variable svResult, not including the terminating null character.

## Sprintf Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the Sprintf function.
```

```

*
* This script gets the capacity of drive C: and then calls
* Sprintf to create a formatted message. That message
* includes the drive letter and disk size, which are stored
* in variables. The message is then displayed.
*
\*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_Sprintf(HWND);

function ExFn_Sprintf(hMSI)
    STRING  svResult, svMssg;
    NUMBER  nvResult;
begin

    // Set up the string parameter for the call to GetSystemInfo.
    svResult = "C:";

    // Get the capacity of drive C.
    GetSystemInfo (DISK_TOTALSPACE, nvResult, svResult);

    // Build a message that incorporates the values
    // returned by GetSystemInfo.
    Sprintf (svMssg, "Total disk space on drive %s is %ld bytes.",
            svResult, nvResult);

    // Display the message.
    MessageBox (svMssg, INFORMATION);

end;

```

## SprintfBox



**Project** • This information applies to the following project types:

- Basic MSI
- InstallScript
- InstallScript MSI

The **SprintfBox** function presents a message box containing one of three icons, a title, and a formatted message. The message can contain variables that are formatted according to commands that you enter.

**SprintfBox** is similar to **MessageBox**, but **SprintfBox** permits much more flexible control over displayed items.



**Caution** • Multiple null-delimited strings should not be used, except for passing to external functionality.

The **SprintfBox** function uses the Microsoft Windows API **MessageBox** to create the message box. The operating environment, not InstallShield, determines the size and location of the message box. The operating environment also generates the text for the OK button in the local language (the language that the operating system is running under). You cannot change the text in this button. For more information regarding the use of Windows **MessageBox** types, consult the description of the **MessageBox** Windows API function in the appropriate Windows SDK.

Note the following when using Windows message box constants:


- Some Windows **MessageBox** type constants are predefined in the **ISRTWindows.h** file that is provided in the *InstallShield Program Files Folder\Script\Isrt\Include* folder. This file is automatically included in your installation when you include **lfx.h** in your script. You do not need to redefine any constants that are defined in **ISRTWindows.h**; doing so will result in a compiler warning. To determine which constants are predefined, refer to the **ISRTWindows.h** file.
- To use constants that are not defined in **ISRTWindows.h**, you must define them (using **#define**) in the declaration block of your installation script. You cannot simply include the **Windows.h** file that is usually part of a C++ program. The values that you need to assign to the undefined constants can generally be found in an include file that is provided with the appropriate Windows SDK or development tool. (For Microsoft Visual C++, most constants can be found in the **Winuser.h** file, which is located in the *InstallShield Program Files Folder\Script\Resource* folder.)
- Windows and InstallShield message box constants cannot be used together in an installation. If an InstallShield message box constant is combined with a Windows message box constant using the OR operator (**|**), the Windows message box constant is ignored.
- Some Windows message box styles are not supported on some Windows platforms. To determine whether a particular style is supported on the operating systems targeted by the installation, consult the appropriate Windows SDK.

## Syntax

```
SprintfBox ( nType, szTitle, szFormat [,arg] [,...] );
```

## Parameters

**Table 182 • SprintfBox Parameters**

Parameter	Description
<b>nType</b>	<p>Specifies the type of icon to display in the message box. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• INFORMATION</li> <li>• WARNING</li> <li>• SEVERE</li> </ul> <p>Advanced developers familiar with the Windows APIs can specify any style of message box by using the native message box style constants in the parameter nType. See the description for the MessageBox or WinMessageBox API in the programming manual for your operating environment. If you are using any of the native message box styles, the InstallShield SprintfBox function returns the return value from the Windows API. Therefore, you must use the Windows API return values in your script.</p> <p>For example, if you pass YES NO CANCEL as the first parameter, the SprintfBox message box has Yes, No, and Cancel buttons. The respective button return values, as defined by the Windows API MessageBox, are 6 (IDYES), 7 (IDNO), and 2 (IDCANCEL). You must use the appropriate constant values in your script, either as number literals, or as constants defined as the number literals in the declare block of your script.</p> <div>  <p><b>Tip</b> • Advanced developers can use <code>MB_STYLE</code> directly as the first parameter of the <code>SprintfBox</code> function to replace the constants <code>SEVERE</code>, <code>WARNING</code>, or <code>INFORMATION</code>. The value of <code>MB_STYLE</code> is listed in the <code>Windows.h</code> file. You can either enter the value directly in the parameter <code>nType</code>, or you can use the <code>#define</code> preprocessor directive to define the constants associated with the value.</p> </div>
<b>szTitle</b>	Specifies the title of the message box. To display the default title Error, pass a null string ("") in this parameter.
<b>szFormat</b>	Specifies a string that includes a format specifier for each argument to be included in the message.



**Table 182 • sprintfBox Parameters (cont.)**

Parameter	Description
<b>arg</b>	<p>Specifies up to 10 arguments to be included in the message. There must be one argument for each format specifier in the message; the type of each argument must match the type of its respective format specifier. sprintfBox generates a compiler error or fails at run time under the following conditions:</p> <ul style="list-style-type: none"> <li>• More than 10 format specifiers and arguments are specified: compiler error.</li> <li>• The number of arguments does not match the number of format specifiers. When a specifier does not have a corresponding argument, the resulting string contains unpredictable characters in the specifier's position. When there are more arguments than specifiers, the excess arguments are not inserted into the resulting string.</li> <li>• A variable does not match the type of its respective format specifier.</li> <li>• A character itself is passed, rather than its numeric ASCII value or the return from the <a href="#">STRTOCHAR</a> function, with the format specifier %c.</li> </ul>

## Return Values

The return value is not significant unless you are using standard Microsoft Windows message box styles. If you are using these styles, the return value is the same as the return value from the MessageBox API functions.

## Additional Information

The dialog that is displayed by the sprintfBox function cannot be displayed with a skin; it appears the same regardless of whether you have specified a skin.

## sprintfBox Example



**Project •** This information applies to the following project types:

- Basic MSI
- InstallScript
- InstallScript MSI



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the sprintfBox function.
*
* This script gets the capacity of drive C: and then calls
* sprintfBox to create and display a formatted message. That

```

```

* message includes the drive letter and disk size, which are
* stored in variables.
*
\*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_SprintfBox(HWND);

function ExFn_SprintfBox(hMSI)
    STRING  svResult;
    NUMBER  nvResult;
begin

    // Set up the string parameter for the call to GetSystemInfo.
    svResult = "C:";

    // Get the capacity of drive C.
    GetSystemInfo (DISK_TOTALSPACE, nvResult, svResult);

    // Build and display a message that incorporates
    // the values returned by GetSystemInfo.
    SprintfBox (INFORMATION, "System Information",
                "Total disk space on drive %s is %ld bytes.",
                svResult, nvResult);

end;

```

## SprintfMsiLog



**Project** • This information applies to the following project types:

- Basic MSI
- InstallScript MSI

The **SprintfMsiLog** function writes a message directly to the Windows Installer log file.

### Syntax

```
SprintfMsiLog ( szFormat [,arg] [... ] );
```

## Parameters

**Table 183** • SprintfMsiLog Parameters

Parameter	Description
<b>szFormat</b>	Specifies a string that can include literal text and must include one format specifier for each argument to be embedded in the string returned by <code>svResult</code> .
<b>arg</b>	<p>You can specify up to nine arguments to be included in the message. You must have one argument for each format specifier in the message. The type of each argument must match the type of its respective format specifier.</p> <p>SprintfMsiLog generates a compiler error or fail at run time under the following conditions:</p> <ul style="list-style-type: none"> <li>• More than nine format specifiers and arguments are specified. This results in a compiler error.</li> <li>• The number of arguments does not match the number of format specifiers. When a specifier does not have a corresponding argument, the resulting string contains unpredictable characters in the specifier's position. When there are more arguments than specifiers, the excess arguments are not inserted into the resulting string.</li> <li>• A variable does not match the type of its respective format specifier. The resulting string contains unpredictable characters in the specifier's position.</li> </ul>

## Return Values

There are no return values for this function. If the function is successful, the values are written to the Windows Installer log file. If the function fails, the values are not written to the Windows Installer log file.

# SQLBrowse



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The [SQLBrowse2](#) function supersedes the **SQLBrowse** function.

The **SQLBrowse** function creates a dialog that lets an end user display a list of all SQL Servers that are available on the network.

This function is in the **SQLRT.obl** file for InstallScript projects, and in the **SQLCONV.obl** file for InstallScript MSI projects. If you are using the SQL Scripts view in InstallShield, the appropriate file is automatically added to your linker settings. However, if you are not using this view, add the appropriate file to your linker settings: On the Build menu, click Settings, and then add it to the Libraries (.obl) box.

## Syntax

```
SQLBrowse( svServer );
```

## Parameters

**Table 184** • SQLBrowse Parameters

Parameter	Description
<b>svServer</b>	<p>When the function returns, this parameter contains the string with the server name that the end user selected.</p> <p>If an alias name was used to connect to a SQL Server database, this parameter contains the alias name.</p>

## Return Values

**Table 185** • SQLBrowse Return Values

Return Value	Description
<b>NEXT</b>	The end user clicked the OK button.
<b>CANCEL</b>	The end user clicked the Cancel button.
<b>&lt; ISERR_SUCCESS</b>	The dialog could not be displayed.

# SQLBrowse2



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SQLBrowse2** function creates a dialog that lets an end user display a list of all database servers that are available on the network for the database technologies specified for a connection.

## Syntax

```
SQLBrowse2( szConnection, svServer );
```

## Parameters

**Table 186** • SQLBrowse2 Parameters

Parameter	Description
<b>szConnection</b>	Specifies the name of the connection that you created in the SQL Scripts view.
<b>svServer</b>	<p>When the function returns, this parameter contains the string with the server name that the end user selected.</p> <p>If an alias name was used to connect to a SQL Server database, this parameter contains the alias name.</p>

## Return Values

**Table 187** • SQLBrowse2 Return Values

Return Value	Description
<b>NEXT</b>	The end user clicked the OK button.
<b>CANCEL</b>	The end user clicked the Cancel button.
<b>&lt; ISERR_SUCCESS</b>	The dialog could not be displayed.

# SQLDatabaseBrowse



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SQLDatabaseBrowse** function creates a dialog that lets the end user display a list of all database catalogs that are available on the specified database server. This function calls **SQLRTGetDatabases**, which uses **SQLRT.dll** for InstallScript projects and **ISSQLSRV.dll** for InstallScript MSI projects.



**Note** • The **SQLDatabaseBrowse** function uses settings from the SQL settings file; therefore, it can be called only after [SQLRTInitialize2](#) has already been called.

## Syntax

```
SQLDatabaseBrowse( szConnection, szServer, bvWindowsLogin, szUser, szPassword, svDBCatalog );
```

## Parameters

**Table 188** • SQLDatabaseBrowse Parameters

Parameter	Description
<b>szConnection</b>	Specifies the name of the connection that you created in the SQL Scripts view.
<b>svServer</b>	When the function returns, this parameter contains the string with the server name the end user selected.
<b>bvWindowsLogin</b>	<p>Specifies the initial state of the radio buttons specifying login using Windows credentials vs. SQL Server credentials. When this is TRUE, the password and login ID fields are disabled. When it is FALSE, indicating that SQL Server credentials are to be used, the password and login fields are enabled.</p> <p>Once the function returns, this parameter contains TRUE if the Windows credentials radio button was specified by the end user and FALSE if the SQL Server credentials radio was selected instead.</p>
<b>szUser</b>	Specifies the default string to display in the Login ID edit box. Once the dialog function returns, this parameter contains the Login ID that the end user entered in the edit field. This information is only pertinent if the end user selected to use SQL Login instead of Windows credentials.
<b>szPassword</b>	Specifies the default string to display in the Password edit field. Once the dialog function returns, this parameter contains the password that the end user entered in the edit field. This information is only pertinent if the end user selected to use SQL Login instead of Windows credentials.
<b>svDBCatalog</b>	Specifies the name of the SQL database catalog.

## Return Values

**Table 189** • SQLDatabaseBrowse Return Values

Return Value	Description
<b>NEXT</b>	The end user clicked the OK button.
<b>CANCEL</b>	The end user clicked the Cancel button.
<b>&lt; ISERR_SUCCESS</b>	The dialog could not be displayed.

# SQLRTComponentInstall



**Project** • This information applies to InstallScript projects.

The **SQLRTComponentInstall** function executes the SQL script that is associated with the specified component if the script is scheduled to run during installation.



**Note** • If you want to call any built-in SQL-related function before the *OnSQLServerInitialize* event handler is called in an InstallScript project or the *OnSQLLogin* event handler is called in an InstallScript MSI project, call the [SQLRTInitialize2](#) function first. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

## Syntax

```
SQLRTComponentInstall( szComponent );
```

## Parameters

**Table 190** • SQLRTComponentInstall Parameters

Parameter	Description
<b>szComponent</b>	Specifies the name of the component that contains the SQL script.

## Return Values

**Table 191** • SQLRTComponentInstall Return Values

Return Value	Description
<b>&gt;=ISERR_SUCCESS</b>	This function was able to execute the SQL script.
<b>&lt;ISERR_SUCCESS</b>	This function failed to execute the SQL script.
<b>SQL_ERROR_NOT_INITIALIZED (-10)</b>	<b>SQLRT.dll</b> is not loaded.

# SQLRTComponentUninstall



**Project** • This information applies to InstallScript projects.

The **SQLRTComponentUninstall** function executes the SQL script that is associated with the specified component if the script is scheduled to run during uninstallation.



**Note** • If you want to call any built-in SQL-related function before the *OnSQLServerInitialize* event handler is called in an InstallScript project or the *OnSQLLogin* event handler is called in an InstallScript MSI project, call the

[SQLRTInitialize2](#) function first. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

## Syntax

```
SQLRTComponentUninstall( szComponent );
```

## Parameters

**Table 192** • SQLRTComponentUninstall Parameters

Parameter	Description
<b>szComponent</b>	Specifies the name of the component that contains the SQL script.

## Return Values

**Table 193** • SQLRTComponentUninstall Return Values

Return Value	Description
<b>&gt;=ISERR_SUCCESS</b>	This function was able to execute the SQL script.
<b>&lt;ISERR_SUCCESS</b>	This function failed to execute the SQL script.
<b>SQL_ERROR_NOT_INITIALIZED (-10)</b>	<b>SQLRT.dll</b> is not loaded.

# SQLRTConnect



**Project** • This information applies to *InstallScript* projects.

The [SQLRTConnect2](#) function supersedes the **SQLRTConnect** function.

The **SQLRTConnect** function establishes a connection using the specified credential.



**Note** • If you want to call any built-in SQL-related function before the *OnSQLServerInitialize* event handler is called in an *InstallScript* project or the *OnSQLLogin* event handler is called in an *InstallScript MSI* project, call the [SQLRTInitialize2](#) function first. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

## Syntax

```
SQLRTConnect( szConnection, szServer, bTrust, szUserName, szPassword );
```



## Parameters

**Table 194** • SQLRTConnect Parameters

Parameter	Description
<b>szConnection</b>	Specifies the name of the connection that you created in the SQL Scripts view.
<b>szServer</b>	Specifies the SQL Server.
<b>bTrust</b>	<p>Specifies the initial state of the radio buttons specifying login using Windows credentials vs. SQL Server credentials. When this is TRUE, the password and login ID fields are disabled. When it is FALSE, indicating that SQL Server credentials are to be used, the password and login fields are enabled.</p> <p>Once the function returns, this parameter contains TRUE if the Windows credentials radio button was specified by the end user and FALSE if the SQL Server credentials radio was selected instead.</p>
<b>szUserName</b>	Specifies the SQL Server login information.
<b>szPassword</b>	Specifies the password associated with a user account.

## Return Values

**Table 195** • SQLRTConnect Return Values

Return Value	Description
<b>&gt;=ISERR_SUCCESS</b>	This function was able to establish a connection.
<b>&lt;ISERR_SUCCESS</b>	This function failed to establish a connection.
<b>SQL_ERROR_NOT_INITIALIZED (-10)</b>	<b>SQLRT.dll</b> is not loaded.

# SQLRTConnect2



**Project** • This information applies to InstallScript projects.

The **SQLRTConnect2** function establishes a connection. This function must be called before file transfer if the connection is to be used to run scripts during installation. **SQLRTConnect2** returns the database server name when it fails to establish the connection.



**Note** • The **SQLRTConnect2** function uses **SQLRT.dll**; therefore, it can be called only after [SQLRTInitialize2](#) has already been called. To learn more, see [Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects](#).

## Syntax

```
SQLRTConnect2( szConnection, szServer, bWinLogin, szUser, szPassword, szDatabaseServer );
```

## Parameters

**Table 196 • SQLRTConnect2 Parameters**

Parameter	Description
<b>szConnection</b>	Specifies the name of the connection that you created in the SQL Scripts view.
<b>szServer</b>	Specifies the SQL Server.
<b>bWinLogin</b>	<p>Specifies the initial state of the radio buttons specifying login using Windows credentials vs. SQL Server credentials. When this is TRUE, the password and login ID fields are disabled. When it is FALSE, indicating that SQL Server credentials are to be used, the password and login fields are enabled.</p> <p>Once the function returns, this parameter contains TRUE if the Windows credentials radio button was specified by the end user and FALSE if the SQL Server credentials radio was selected instead.</p>
<b>szUser</b>	Specifies the SQL Server login information.
<b>szPassword</b>	Specifies the password associated with a user account.
<b>szDatabaseServer</b>	Specifies the database server.

## Return Values

**Table 197 • SQLRTConnect2 Return Values**

Return Value	Description
<b>&gt;=ISERR_SUCCESS</b>	This function was able to establish a connection.
<b>&lt;ISERR_SUCCESS</b>	This function failed to establish a connection.
<b>SQL_ERROR_NOT_INITIALIZED (-10)</b>	<b>SQLRT.dll</b> is not loaded.

## Additional Information

If you want to be able to specify which catalog to connect to, replace the **SQLRTConnect2** function call in the OnSQLServerInitialize event with the **SQLRTConnectDB** function call. For more information, see [SQLRTConnectDB](#).

# SQLRTConnectDB



---

**Project** • This information applies to InstallScript projects.

To establish a connection to a specific catalog, replace the **SQLRTConnect2** function call in the OnSQLServerInitialize event with the **SQLRTConnectDB** function call. Pass the catalog name as the szDB parameter for **SQLRTConnectDB**.

If you want end users to be able to specify the catalog at run time, pass an end user-defined variable as szDB.



---

**Note** • The **SQLRTConnectDB** function uses **SQLRT.dll**; therefore, it can be called only after [SQLRTInitialize2](#) has already been called. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

## Syntax

```
SQLRTConnectDB( szConnection, szDB, szServer, bWinLogin, szUser, szPassword );
```

## Parameters

**Table 198** • SQLRTConnectDB Parameters

Parameter	Description
<b>szConnection</b>	Specifies the SQL Server connection.
<b>szDB</b>	<p>Specifies the catalog that you want to connect to.</p> <p>To specify multiple catalogs, separate each with a semicolon (;).</p> <p>If you want end users to be able to specify the catalog at run time, pass an end user-defined variable as szDB.</p>
<b>szServer</b>	Specifies the SQL Server.
<b>bWinLogin</b>	<p>Specifies the initial state of the radio buttons specifying login using Windows credentials vs. SQL Server credentials. When this is TRUE, the password and login ID fields are disabled. When it is FALSE, indicating that SQL Server credentials are to be used, the password and login fields are enabled.</p> <p>Once the function returns, this parameter contains TRUE if the Windows credentials radio button was specified by the end user and FALSE if the SQL Server credentials radio was selected instead.</p>
<b>szUser</b>	Specifies the SQL Server login information.
<b>szPassword</b>	Specifies the password associated with a user account.

## Return Values

**Table 199** • SQLRTConnectDB Return Values

Return Value	Description
<b>&gt;=ISERR_SUCCESS</b>	This function was able to establish a connection.
<b>&lt;ISERR_SUCCESS</b>	This function failed to establish a connection.
<b>SQL_ERROR_NOT_INITIALIZED (-10)</b>	<b>SQLRT.dll</b> is not loaded.

# SQLRTDoRollbackAll



**Project** • This information applies to InstallScript projects.

The **SQLRTDoRollbackAll** function executes all of SQL scripts scheduled to run during rollback.



**Note** • The **SQLRTDoRollbackAll** function uses **SQLRT.dll**; therefore, it can be called only after **SQLRTInitialize2** has already been called. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

## Syntax

```
SQLRTDoRollbackAll();
```

## Parameters

**SQLRTDoRollbackAll** takes no parameters.

## Return Values

**Table 200** • SQLRTDoRollbackAll Return Values

Return Value	Description
>=ISERR_SUCCESS	This function was able to execute a script.
<ISERR_SUCCESS	This function failed to execute a script.
SQL_ERROR_NOT_INITIALIZED (-10)	<b>SQLRT.dll</b> is not loaded.

# SQLRTGetBatchList



**Project** • This information applies to *InstallScript* projects.

The **SQLRTGetBatchList** function returns the list of components that are associated with SQL scripts that need to be run when batch mode is enabled.



**Note** • The **SQLRTGetBatchList** function uses **SQLRT.dll**; therefore, it can be called only after **SQLRTInitialize2** has already been called. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

## Syntax

```
SQLRTGetBatchList( nOperation );
```

## Parameters

**Table 201** • SQLRTGetBatchList Parameters

Parameter	Description
<b>nOperation</b>	<p>Pass either of the following constants through this parameter:</p> <ul style="list-style-type: none"> <li>• <b>SQL_BATCH_INSTALL</b>—Obtains the list of SQL scripts scheduled to be run during installation.</li> <li>• <b>SQL_BATCH_UNINSTALL</b>—Obtains the list of SQL scripts scheduled to be run during uninstallation.</li> </ul>

## Return Values

**Table 202** • SQLRTGetBatchList Return Values

Return Value	Description
<b>listConnections</b>	Returns a list of component names.

## Additional Information

For details about batch mode, see [Specifying the Order for Running Multiple SQL Scripts That Are Associated with a Connection](#).

# SQLRTGetBatchMode



**Project** • This information applies to InstallScript projects.

The **SQLRTGetBatchMode** function returns whether the batch mode is enabled or disabled.



**Note** • The **SQLRTGetBatchMode** function uses **SQLRT.dll**; therefore, it can be called only after [SQLRTInitialize2](#) has already been called. To learn more, see [Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects](#).

## Syntax

```
SQLRTGetBatchMode();
```

## Parameters

**SQLRTGetBatchMode** takes no parameters.

## Return Values

**Table 203** • SQLRTGetBatchMode Return Values

Return Value	Description
<b>TRUE</b>	Batch mode is enabled.
<b>FALSE</b>	Batch mode is disabled.

## Additional Information

For details about batch mode, see [Specifying the Order for Running Multiple SQL Scripts That Are Associated with a Connection](#).

# SQLRTGetBrowseOption



**Project** • This information applies to InstallScript projects.

The **SQLRTGetBrowseOption** function returns the current value of the browse option for the SQL Server browse combo box and list box controls, which can display local servers, remote servers, server aliases, or a combination of these types.



**Note** • If you want to call any built-in SQL-related function before the *OnSQLServerInitialize* event handler is called in an InstallScript project or the *OnSQLLogin* event handler is called in an InstallScript MSI project, call the [SQLRTInitialize2](#) function first. To learn more, see [Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects](#).

## Syntax

```
SQLRTGetBrowseOption( );
```

## Parameters

**SQLRTGetBrowseOption** takes no parameters.

## Return Values

**Table 204 • SQLRTGetBrowseOption Return Values**

Return Value	Description
<b>0</b>	<p>The SQL Server browse combo box and list box controls show all available SQL Servers (local servers, remote servers , and server aliases).</p> <p>This is the return value if <b>SQL_BROWSE_ALL</b> or zero (<b>0</b>) is passed for the nBrowseOption parameter of <b>SQLRTSetBrowseOption</b>, or if the <b>SQLRTSetBrowseOption</b> function is not called.</p>
<b>1</b>	<p>The SQL Server browse combo box and list box controls show all available local SQL Servers.</p> <p>This is the return value if <b>SQL_BROWSE_LOCAL</b> or <b>1</b> is passed for the nBrowseOption parameter of <b>SQLRTSetBrowseOption</b>.</p>
<b>2</b>	<p>The SQL Server browse combo box and list box controls show all available remote SQL Servers.</p> <p>This is the return value if <b>SQL_BROWSE_REMOTE</b> or <b>2</b> is passed for the nBrowseOption parameter of <b>SQLRTSetBrowseOption</b>.</p>
<b>3</b>	<p>The SQL Server browse combo box and list box controls show all available local SQL Servers and remove SQL Servers.</p> <p>This is the return value if <b>SQL_BROWSE_LOCAL   SQL_BROWSE_REMOTE</b> or <b>3</b> is passed for the nBrowseOption parameter of <b>SQLRTSetBrowseOption</b>.</p>
<b>4</b>	<p>The SQL Server browse combo box and list box controls show all available SQL Server aliases.</p> <p>This is the return value if <b>SQL_BROWSE_ALIAS</b> or <b>4</b> is passed for the nBrowseOption parameter of <b>SQLRTSetBrowseOption</b>.</p>
<b>5</b>	<p>The SQL Server browse combo box and list box controls show all available local SQL Servers and SQL Server aliases.</p> <p>This is the return value if <b>SQL_BROWSE_LOCAL   SQL_BROWSE_ALIAS</b> or <b>5</b> is passed for the nBrowseOption parameter of <b>SQLRTSetBrowseOption</b>.</p>
<b>6</b>	<p>The SQL Server browse combo box and list box controls show all available remote SQL Servers and SQL Server aliases.</p> <p>This is the return value if <b>SQL_BROWSE_REMOTE   SQL_BROWSE_ALIAS</b> or <b>6</b> is passed for the nBrowseOption parameter of <b>SQLRTSetBrowseOption</b>.</p>
<b>7</b>	<p>The SQL Server browse combo box and list box controls show all available SQL Servers (local servers, remote servers , and server aliases).</p> <p>This is the return value if <b>SQL_BROWSE_LOCAL   SQL_BROWSE_REMOTE   SQL_BROWSE_ALIAS</b> or <b>7</b> is passed for the nBrowseOption parameter of <b>SQLRTSetBrowseOption</b>.</p>



# SQLRTGetComponentScriptError



---

**Project** • This information applies to InstallScript projects.

The [SQLRTGetComponentScriptError2](#) function supersedes the **SQLRTGetComponentScriptError** function.

The **SQLRTGetComponentScriptError** function retrieves the last error while executing a SQL script that is associated with the component.



---

**Note** • The **SQLRTGetComponentScriptError** function uses **SQLRT.dll**; therefore, it can be called only after [SQLRTInitialize2](#) has already been called. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

## Syntax

```
SQLRTGetComponentScriptError( szComponent, szMessage, nvErrorType, nvErrorLine );
```

## Parameters

**Table 205** • SQLRTGetComponentScriptError Parameters

Parameter	Description
<b>szComponent</b>	Specifies the name of the component that is associated with the script.
<b>szMessage</b>	Specifies the error message that the database server returned.
<b>nvErrorType</b>	Specifies the type of error. Valid types are: <ul style="list-style-type: none"> <li>• <b>SQL_ERROR_SCRIPT_UNABLE_OPEN_FILE</b>—Failed to open the script file.</li> <li>• <b>SQL_ERROR_SCRIPT_CONNECTION_NOT_OPEN</b>—The specified connection is not opened.</li> <li>• <b>SQL_ERROR_GET_SCHEMA_VERSION</b>—Error while attempting to get the schema version from the target database.</li> <li>• <b>SQL_ERROR_SET_SCHEMA_VERSION</b>—Error while attempting to set the schema version to the target database.</li> <li>• <b>SQL_ERROR_SCRIPT_COMMAND_ERROR</b>—Failed to execute a SQL batch.</li> </ul>
<b>nvErrorLine</b>	Specifies the line number of the error.

## Return Values

**Table 206** • SQLRTGetComponentScriptError Return Values

Return Value	Description
<b>TRUE</b>	One or more errors occurred.
<b>FALSE</b>	No errors occurred.

# SQLRTGetComponentScriptError2



**Project** • This information applies to InstallScript projects.

The **SQLRTGetComponentScriptError2** function retrieves the last error while executing a SQL script that is associated with the component. This function takes several parameters (szScriptName, szTechnology, szServer, and szDB) that the **SQLRTGetComponentScriptError** function does not.



---

**Note** • The **SQLRTGetComponentScriptError** function uses **SQLRT.dll**; therefore, it can be called only after [SQLRTInitialize2](#) has already been called. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

## Syntax

```
SQLRTGetComponentScriptError2( szComponent, szMessage, nvErrorType, nvErrorLine, szScriptName,  
    szTechnology, szServer, szDB );
```

## Parameters

**Table 207** • SQLRTGetComponentScriptError2 Parameters

Parameter	Description
<b>szComponent</b>	Specifies the name of the component that is associated with the script.
<b>szMessage</b>	Specifies the error message that the database server returned.
<b>nvErrorType</b>	Specifies the type of error. Valid types are: <ul style="list-style-type: none"> <li>• <b>SQL_ERROR_SCRIPT_UNABLE_OPEN_FILE</b>—Failed to open the script file.</li> <li>• <b>SQL_ERROR_SCRIPT_CONNECTION_NOT_OPEN</b>—The specified connection is not opened.</li> <li>• <b>SQL_ERROR_GET_SCHEMA_VERSION</b>—Error while attempting to get the schema version from the target database.</li> <li>• <b>SQL_ERROR_SET_SCHEMA_VERSION</b>—Error while attempting to set the schema version to the target database.</li> <li>• <b>SQL_ERROR_SCRIPT_COMMAND_ERROR</b>—Failed to execute a SQL batch.</li> </ul>
<b>nvErrorLine</b>	Specifies the line number of the error.
<b>szScriptName</b>	Specifies the name of the script.
<b>szTechnology</b>	Specifies the name of the database server product: Microsoft SQL Server, Oracle, or MySQL.
<b>szServer</b>	Specifies the name of the target database server.
<b>szDB</b>	Specifies the name of the target database catalog.

## Return Values

**Table 208** • SQLRTGetComponentScriptError Return Values

Return Value	Description
<b>TRUE</b>	One or more errors occurred.
<b>FALSE</b>	No errors occurred.

# SQLRTGetConnectionAuthentication



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SQLRTGetConnectionAuthentication** function gets the default SQL Server connection authentication type.



**Note** • If you want to call any built-in SQL-related function before the *OnSQLServerInitialize* event handler is called in an *InstallScript* project or the *OnSQLLogin* event handler is called in an *InstallScript MSI* project, call the [SQLRTInitialize2](#) function first. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

## Syntax

```
SQLRTGetConnectionAuthentication( szConnection );
```

## Parameters

**Table 209** • SQLRTGetConnectionAuthentication Parameters

Parameter	Description
<b>szConnection</b>	Specifies the SQL Server connection.

## Return Values

**Table 210** • SQLRTGetConnectionAuthentication Return Values

Return Value	Description
<b>TRUE</b>	Perform Windows authentication.
<b>FALSE</b>	Perform SQL authentication.

# SQLRTGetConnectionInfo



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SQLRTGetConnectionInfo** function retrieves strings containing the connection information (the default server, database, default user name, and default password).



**Note** • The **SQLRTGetConnectionInfo** function uses **SQLRT.dll** in *InstallScript* projects and **ISSQLSRV.dll** in *InstallScript MSI* projects; therefore, it can be called only after [SQLRTInitialize2](#) has already been called. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

## Syntax

```
SQLRTGetConnectionInfo( szConnection, szServer, szDB, szUser, szPassword );
```

## Parameters

**Table 211** • SQLRTGetConnectionInfo Parameters

Parameter	Description
<b>szConnection</b>	Specifies the SQL Server connection
<b>szServer</b>	Specifies the SQL Server.
<b>szDB</b>	Specifies the SQL Server database.
<b>szUser</b>	Specifies the SQL Server login information.
<b>szPassword</b>	Specifies the password associated with a user account.

## Return Values

**Table 212** • SQLRTGetConnectionInfo Return Values

Return Value	Description
<b>&gt;=ISERR_SUCCESS</b>	This function was able to establish a connection.
<b>&lt;ISERR_SUCCESS</b>	This function failed to establish a connection.
<b>SQL_ERROR_NOT_INITIALIZED (-10)</b>	<b>SQLRT.dll</b> is not loaded.

# SQLRTGetConnections



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SQLRTGetConnections** function retrieves a string list of connections that are present in the settings file.



**Note** • The **SQLRTGetConnections** function uses **SQLRT.dll** in *InstallScript* projects and **ISSQLSRV.dll** in *InstallScript MSI* projects; therefore, it can be called only after [SQLRTInitialize2](#) has already been called. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

## Syntax

```
SQLRTGetConnections( );
```

## Parameters

**SQLRTGetConnections** takes no parameters.

## Return Values

**Table 213** • SQLRTGetConnections Return Values

Return Value	Description
<b>listConnections;</b>	Returns a list of connection names.

# SQLRTGetDatabases



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SQLRTGetDatabases** function returns a list of database catalogs that are available on the specified database server.



**Note** • If you want to call any built-in SQL-related function before the *OnSQLServerInitialize* event handler is called in an *InstallScript* project or the *OnSQLLogin* event handler is called in an *InstallScript MSI* project, call the [SQLRTInitialize2](#) function first. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

## Syntax

```
SQLRTGetDatabases( szConnection, szServer, bTrust, szUserName, szPassword );
```

## Parameters

**Table 214** • SQLRTGetDatabases Parameters

Parameter	Description
<b>szConnection</b>	Specifies the name of the connection that you created in the SQL Scripts view.
<b>szServer</b>	Specifies the SQL Server.
<b>bTrust</b>	<p>Specifies the initial state of the radio buttons specifying login using Windows credentials vs. SQL Server credentials. When this is TRUE, the password and login ID fields are disabled. When it is FALSE, indicating that SQL Server credentials are to be used, the password and login fields are enabled.</p> <p>Once the function returns, this parameter contains TRUE if the Windows credentials radio button was specified by the end user and FALSE if the SQL Server credentials radio was selected instead.</p>
<b>szUserName</b>	Specifies the SQL Server login information.
<b>szPassword</b>	Specifies the password associated with a user account.

## Return Values

Returns a list of the database catalogs that are available on the target database server.

# SQLRTGetErrorMessage



**Project** • This information applies to InstallScript projects.

The **SQLRTGetErrorMessage** function returns the descriptive message of the last error encountered by the SQL run time when a connection is being opened.



**Note** • **SQLRTGetErrorMessage** calls [SQLRTGetLastError2](#), which uses **SQLRT.dll** and settings from the SQL settings file; therefore, it can be called only after [SQLRTInitialize2](#) has already been called. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

## Syntax

```
SQLRTGetErrorMessage( svMessage );
```



## Parameters

**Table 215** • SQLRTGetErrorMessage Parameters

Parameter	Description
<b>svMessage</b>	Specifies the SQL error message.

## Return Values

**Table 216** • SQLRTGetErrorMessage Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	The function was successfully executed.
<b>SQL_ERROR_NOT_INITIALIZED</b>	<b>SQLRT.dll</b> is not loaded.

# SQLRTGetLastError



**Project** • This information applies to InstallScript projects.

The [SQLRTGetLastError2](#) function supersedes the **SQLRTGetLastError** function.

The **SQLRTGetLastError** function returns the text of the last error encountered by the SQL run time.

**SQLRTGetLastError** also loads the proper SQL error message.



**Note** • The **SQLRTGetLastError** function uses **SQLRT.dll** and settings from the SQL settings file; therefore, it can be called only after [SQLRTInitialize2](#) has already been called. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

## Syntax

```
SQLRTGetLastError( szError );
```

## Parameters

**Table 217** • SQLRTGetLastError Parameters

Parameter	Description
<b>szError</b>	Specifies the SQL error.

## Return Values

None

# SQLRTGetLastError2



**Project** • This information applies to InstallScript projects.

The **SQLRTGetLastError2** function returns detailed information about the last error encountered by the SQL run time. **SQLRTGetLastError2** also loads the proper SQL error message.



**Note** • The **SQLRTGetLastError2** function uses **SQLRT.dll** and settings from the SQL settings file; therefore, it can be called only after [SQLRTInitialize2](#) has already been called. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

## Syntax

```
SQLRTGetLastError2( nvErrorCode, svMessage, svTechnology, svConnection, svServer, svDBCatalog );
```

## Parameters

**Table 218** • SQLRTGetLastError2 Parameters

Parameter	Description
<b>nvErrorCode</b>	Specifies the SQL error code.
<b>svMessage</b>	Specifies the SQL error message.
<b>svTechnology</b>	Specifies the type of server (such as Microsoft SQL Server or Oracle) that was used.
<b>svConnection</b>	Specifies the SQL error code.
<b>svServer</b>	Specifies the name of the SQL database server.
<b>svDBCatalog</b>	Specifies the name of the SQL database catalog.

## Return Values

None

# SQLRTGetScriptErrorMessage



**Project** • This information applies to InstallScript projects.

The **SQLRTGetScriptErrorMessage** function returns the descriptive message of the last error encountered by the SQL run time when a SQL script is executing.



**Note** • **SQLRTGetScriptErrorMessage** calls [SQLRTGetComponentScriptError2](#), which uses **SQLRT.dll** and settings from the SQL settings file; therefore, it can be called only after [SQLRTInitialize2](#) has already been called. To learn more, see [Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects](#).

## Syntax

```
SQLRTGetScriptErrorMessage( svMessage );
```

## Parameters

**Table 219** • SQLRTGetScriptErrorMessage Parameters

Parameter	Description
<b>svMessage</b>	Specifies the SQL error message.

## Return Values

**Table 220** • SQLRTGetScriptErrorMessage Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	The function was successfully executed.
<b>SQL_ERROR_NOT_INITIALIZED</b>	<b>SQLRT.dll</b> is not loaded.

# SQLRTGetServers



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SQLRTGetServers** function returns a list of database servers on the network for all database technologies included in the installation. When **bFilter** is **TRUE**, it returns only local database servers.



**Note** • If you want to call any built-in SQL-related function before the *OnSQLServerInitialize* event handler is called in an *InstallScript* project or the *OnSQLLogin* event handler is called in an *InstallScript MSI* project, call the [SQLRTInitialize2](#) function first. To learn more, see [Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects](#).

## Syntax

```
SQLRTGetServers( bFilter );
```

## Parameters

**Table 221** • SQLRTGetServers Parameters

Parameter	Description
<b>bFilter</b>	<p>Specifies whether the list of database servers that is returned includes only local database servers:</p> <ul style="list-style-type: none"> <li>• <b>TRUE</b>—Return only local database servers.</li> <li>• <b>FALSE</b>—Return all database servers.</li> </ul>

## Return Values

Returns a list of the database servers available on the network for all database technologies included in the installation.

# SQLRTGetServers2



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SQLRTGetServers2** function returns a list of database servers for the database technologies that are specified for a connection. When `szConnection` is empty, this function behaves as [SQLRTGetServers](#).



**Note** • If you want to call any built-in SQL-related function before the *OnSQLServerInitialize* event handler is called in an *InstallScript* project or the *OnSQLLogin* event handler is called in an *InstallScript MSI* project, call the [SQLRTInitialize2](#) function first. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

## Syntax

```
SQLRTGetServers2( szConnection, bFilter );
```

## Parameters

**Table 222** • SQLRTGetServers2 Parameters

Parameter	Description
<b>szConnection</b>	Specifies a connection. If this parameter is empty, the function returns a list of database servers on the network for all database technologies included in the installation.
<b>bFilter</b>	Specifies whether the list of database servers that is returned includes only local database servers: <ul style="list-style-type: none"> <li>• <b>TRUE</b>—Return only local database servers.</li> <li>• <b>FALSE</b>—Return all database servers.</li> </ul>

## Return Values

Returns a list of the database servers available on the network for database technologies specified for a connection.

# SQLRTInitialize



**Project** • This information applies to InstallScript projects.

The [SQLRTInitialize2](#) function supersedes the **SQLRTInitialize** function.

If the **SQLRTInitialize** function is used, it must be the first function called in SQLRT. **SQLRTInitialize** loads the **SQLRT.dll** and initializes it using the settings file.

## Syntax

```
SQLRTInitialize( szSettingsFile );
```

## Parameters

**Table 223** • SQLRTInitialize Parameters

Parameter	Description
<b>szSettingsFile</b>	<p>Specifies the path to the <b>SQLRT.ini</b> file. The following path should be specified for this parameter:</p> <p><b>SUPPORTDIR ^ "SQLRT.ini"</b></p> <p>The <b>SQLRT.ini</b> file should not be modified; otherwise, unexpected behavior may occur at run time. InstallShield creates this file based on SQL settings that are entered in the SQL Scripts view.</p>

## Return Values

None

# SQLRTInitialize2



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SQLRTInitialize2** must be the first function called in the SQL run time. **SQLRTInitialize2** loads the **SQLRT.dll** file for InstallScript projects and the **ISSQLSRV.dll** file for InstallScript MSI projects, and it uses the settings file to initialize the .dll file.



**Note** • If you want to call any built-in SQL-related function before the *OnSQLServerInitialize* event handler is called in an InstallScript project or the *OnSQLLogin* event handler is called in an InstallScript MSI project, call the [SQLRTInitialize2](#) function first. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

## Syntax

```
SQLRTInitialize2 ();
```

## Parameters

**SQLRTInitialize2** takes no parameters.

## Return Values

None

# SQLRTPutConnectionAuthentication



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SQLRTPutConnectionAuthentication** function sets the default SQL Server connection authentication type.



**Note** • If you want to call any built-in SQL-related function before the *OnSQLServerInitialize* event handler is called in an *InstallScript* project or the *OnSQLLogin* event handler is called in an *InstallScript MSI* project, call the [SQLRTInitialize2](#) function first. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

## Syntax

```
SQLRTPutConnectionAuthentication( szConnection, bWinLogin );
```

## Parameters

**Table 224** • SQLRTPutConnectionAuthentication Parameters

Parameter	Description
<b>szConnection</b>	Specifies the name of the connection that you created in the SQL Scripts view.
<b>bWinLogin</b>	<p>Specifies the initial state of the radio buttons specifying login using Windows credentials vs. SQL Server credentials. When this is TRUE, the password and login ID fields are disabled. When it is FALSE, indicating that SQL Server credentials are to be used, the password and login fields are enabled.</p> <p>Once the function returns, this parameter contains TRUE if the Windows credentials radio button was specified by the end user and FALSE if the SQL Server credentials radio was selected instead.</p>

## Return Values

None

# SQLRTPutConnectionInfo



**Project** • This information applies to the following project types:

- *InstallScript*

- *InstallScript MSI*

The [SQLRTPutConnectionInfo2](#) function supersedes the **SQLRTPutConnectionInfo** function.

The **SQLRTPutConnectionInfo** function sets the connection information (the default server, default user name, and default password). This is useful in situations when you need to recall what an end user previously entered, like use of the Back button.



**Note** • The **SQLRTPutConnectionInfo** function uses **SQLRT.dll** in *InstallScript* projects and **ISSQLSRV.dll** in *InstallScript MSI* projects; therefore, it can be called only after [SQLRTInitialize2](#) has already been called. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

## Syntax

```
SQLRTPutConnectionInfo( szConnection, szServer, szUser, szPassword );
```

## Parameters

**Table 225** • SQLRTPutConnectionInfo Parameters

Parameter	Description
<b>szConnection</b>	Specifies the SQL Server connection.
<b>szServer</b>	Specifies the SQL Server.
<b>szUser</b>	Specifies the SQL Server login information.
<b>szPassword</b>	Specifies the password associated with a user account.

## Return Values

**Table 226** • SQLRTPutConnectionInfo Return Values

Return Value	Description
<b>&gt;=ISERR_SUCCESS</b>	This function was able to establish a connection.
<b>&lt;ISERR_SUCCESS</b>	This function failed to establish a connection.
<b>SQL_ERROR_NOT_INITIALIZED (-10)</b>	<b>SQLRT.dll</b> is not loaded.

# SQLRTPutConnectionInfo2



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*



The **SQLRTPutConnectionInfo2** function sets the connection information (the default server, default database catalog, default user name, and default password). This is useful in situations when you need to recall what an end user previously entered, like use of the Back button.



**Note** • The **SQLRTPutConnectionInfo2** function uses **SQLRT.dll** for InstallScript projects and **ISSQLSRV.dll** for InstallScript MSI projects; therefore, it can be called only after [SQLRTInitialize2](#) has already been called. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

## Syntax

```
SQLRTPutConnectionInfo2( szConnection, szServer, svDBCatalog, szUser, szPassword );
```

## Parameters

**Table 227** • SQLRTPutConnectionInfo2 Parameters

Parameter	Description
<b>szConnection</b>	Specifies the SQL Server connection.
<b>szServer</b>	Specifies the SQL Server.
<b>svDBCatalog</b>	Specifies the name of the SQL database catalog.
<b>szUser</b>	Specifies the SQL Server login information.
<b>szPassword</b>	Specifies the password associated with a user account.

## Return Values

**Table 228** • SQLRTPutConnectionInfo2 Return Values

Return Value	Description
<b>&gt;=ISERR_SUCCESS</b>	This function was able to establish a connection.
<b>&lt;ISERR_SUCCESS</b>	This function failed to establish a connection.
<b>SQL_ERROR_NOT_INITIALIZED (-10)</b>	<b>SQLRT.dll</b> is not loaded.

# SQLRTServerValidate



**Project** • This information applies to InstallScript MSI projects.

The **SQLRTServerValidate** function tests connections specified in the installation.



**Note** • If you want to call any built-in SQL-related function before the *OnSQLServerInitialize* event handler is called in an *InstallScript* project or the *OnSQLLogin* event handler is called in an *InstallScript MSI* project, call the [SQLRTInitialize2](#) function first. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

## Syntax

```
SQLRTServerValidate\(\ hInstall );
```

## Parameters

**Table 229** • SQLRTServerValidate Parameters

Parameter	Description
<b>hInstall</b>	Specifies the handle to the installation.  When <b>IS_SQLSERVER_CONNECTIONS_TO_VALIDATE</b> is specified, only the connections that are specified as the value of that Windows Installer property are tested.

## Return Values

**Table 230** • SQLRTServerValidate Return Values

Return Value	Description
<b>=ISERR_SUCCESS</b>	This function was able to establish a connection.
<b>SQL_ERROR_NOT_INITIALIZED</b>	<b>ISSQLSRV.dll</b> is not loaded.

## Additional Information

The result is used to set the value of the **IS\_SQLSERVER\_STATUS** property. Possible values are listed in the following table.

**Table 231** • IS\_SQLSERVER\_STATUS Property Values

Return Value	Description
<b>0</b>	The function was able to establish one or more connections.
<b>1</b>	The function failed to establish a connection.
<b>2</b>	The function was able to establish a connection, but it failed to retrieve the product version from the target database server.
<b>3</b>	The function was able to establish a connection, but the version requirement was not met.

**Table 231** • IS\_SQLSERVER\_STATUS Property Values (cont.)

Return Value	Description
4	The function was able to establish a connection, but MSDE or the Express version was not allowed.
5	The function was able to establish a connection, but it failed to retrieve the schema version from the target database catalog.
6	The function was able to establish a connection, but it failed to set the schema version to the target database catalog.
7	The function could not find the specified ODBC driver.
8	The function was able to establish a connection to the default database catalog, but it failed to create the target database catalog (when <b>Create Catalog If Absent</b> check box in the SQL Scripts view is selected).
9	The function was able to establish a connection to the default database catalog, but it failed to connect to the target database catalog.
10	The function could not find a valid record in <b>ISSQLDBMetaData</b> table.

## SQLRTSetBrowseOption



**Project** • This information applies to InstallScript projects.

The **SQLRTSetBrowseOption** function lets you specify whether the SQL Server browse combo box and list box controls show local servers, remote servers, server aliases, or a combination of these types.



**Note** • If you want to call any built-in SQL-related function before the *OnSQLServerInitialize* event handler is called in an InstallScript project or the *OnSQLLogin* event handler is called in an InstallScript MSI project, call the [SQLRTInitialize2](#) function first. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

### Syntax

```
SQLRTSetBrowseOption( nBrowseOption );
```

## Parameters

**Table 232** • SQLRTSetBrowseOption Parameters

Parameter	Description
<b>nBrowseOption</b>	<p>Pass one or more of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>SQL_BROWSE_ALL (0)</b>—Show all of the available SQL Servers. Specifying this constant has the same effect as selecting the SQL_BROWSE_LOCAL, SQL_BROWSE_REMOTE, and SQL_BROWSE_ALIAS constants.</li> <li>• <b>SQL_BROWSE_LOCAL (1)</b>—Show local SQL Servers.</li> <li>• <b>SQL_BROWSE_REMOTE (2)</b>—Show remote SQL Servers.</li> <li>• <b>SQL_BROWSE_ALIAS (4)</b>—Show SQL Server aliases.</li> </ul> <p>You can specify multiple browse options by combining constants with the bitwise OR operator ( ).</p>

## Return Values

None

# SQLRTTestConnection



**Project** • This information applies to InstallScript MSI projects.

The [SQLRTTestConnection2](#) function supersedes the **SQLRTTestConnection** function.

The **SQLRTTestConnection** function tests all of the connections specified in the installation using the specified credential.



**Note** • If you want to call any built-in SQL-related function before the *OnSQLServerInitialize* event handler is called in an InstallScript project or the *OnSQLLogin* event handler is called in an InstallScript MSI project, call the [SQLRTInitialize2](#) function first. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

## Syntax

```
SQLRTTestConnection( szServer, szDB, szUserName, szPassword, bTrust );
```

## Parameters

**Table 233 • SQLRTTestConnection Parameters**

Parameter	Description
<b>szServer</b>	Specifies the SQL Server.
<b>szDB</b>	Specifies the catalog.
<b>szUserName</b>	Specifies the SQL Server login information.
<b>szPassword</b>	Specifies the password associated with a user account.
<b>bTrust</b>	<p>Specifies the initial state of the radio buttons specifying login using Windows credentials vs. SQL Server credentials. When this is TRUE, the password and login ID fields are disabled. When it is FALSE, indicating that SQL Server credentials are to be used, the password and login fields are enabled.</p> <p>Once the function returns, this parameter contains TRUE if the Windows credentials radio button was specified by the end user and FALSE if the SQL Server credentials radio was selected instead.</p>

## Return Values

**Table 234 • SQLRTTestConnection Return Values**

Return Value	Description
<b>=ISERR_SUCCESS</b>	This function was able to establish a connection.
<b>SQL_ERROR_NOT_INITIALIZED</b>	<b>ISSQLSRV.dll</b> is not loaded.

## Additional Information

The result is used to set the value of the **IS\_SQLSERVER\_STATUS** property. Possible values are listed in the following table.

**Table 235 • IS\_SQLSERVER\_STATUS Property Values**

Return Value	Description
<b>0</b>	The function was able to establish one or more connections.
<b>1</b>	The function failed to establish a connection.
<b>2</b>	The function was able to establish a connection, but it failed to retrieve the product version from the target database server.
<b>3</b>	The function was able to establish a connection, but the version requirement was not met.

**Table 235** • IS\_SQLSERVER\_STATUS Property Values (cont.)

Return Value	Description
4	The function was able to establish a connection, but MSDE or the Express version was not allowed.
5	The function was able to establish a connection, but it failed to retrieve the schema version from the target database catalog.
6	The function was able to establish a connection, but it failed to set the schema version to the target database catalog.
7	The function could not find the specified ODBC driver.
8	The function was able to establish a connection to the default database catalog, but it failed to create the target database catalog (when <b>Create Catalog If Absent</b> check box in the SQL Scripts view is selected).
9	The function was able to establish a connection to the default database catalog, but it failed to connect to the target database catalog.
10	The function could not find a valid record in <b>ISSQLDBMetaData</b> table.

## SQLRTTestConnection2



**Project** • This information applies to InstallScript MSI projects.

The **SQLRTTestConnection2** function establishes a connection.



**Note** • The **SQLRTTestConnection2** function uses **ISSQLSRV.dll**; therefore, it can be called only after [SQLRTInitialize2](#) has already been called. To learn more, see *Using the SQL Run-Time Functions in InstallScript and InstallScript MSI Projects*.

### Syntax

```
SQLRTTestConnection2( szConnection, szServer, szDB, szUserName, szPassword, bTrust );
```

## Parameters

**Table 236** • SQLRTTestConnection2 Parameters

Parameter	Description
<b>szConnection</b>	Specifies the SQL Server connection.
<b>szServer</b>	Specifies the SQL Server.
<b>svDB</b>	Specifies the catalog.
<b>szUser</b>	Specifies the SQL Server login information.
<b>szPassword</b>	Specifies the password associated with a user account.
<b>bTrust</b>	<p>Specifies the initial state of the radio buttons specifying login using Windows credentials vs. SQL Server credentials. When this is TRUE, the password and login ID fields are disabled. When it is FALSE, indicating that SQL Server credentials are to be used, the password and login fields are enabled.</p> <p>Once the function returns, this parameter contains TRUE if the Windows credentials radio button was specified by the end user and FALSE if the SQL Server credentials radio was selected instead.</p>

## Return Values

**Table 237** • SQLRTTestConnection2 Return Values

Return Value	Description
<b>=ISERR_SUCCESS</b>	This function was able to establish a connection.
<b>SQL_ERROR_NOT_INITIALIZED</b>	<b>ISSQLSRV.dll</b> is not loaded.

## Additional Information

The result is used to set the value of the **IS\_SQLSERVER\_STATUS** property. Possible values are listed in the following table.

**Table 238** • IS\_SQLSERVER\_STATUS Property Values

Return Value	Description
<b>0</b>	The function was able to establish one or more connections.
<b>1</b>	The function failed to establish a connection.
<b>2</b>	The function was able to establish a connection, but it failed to retrieve the product version from the target database server.
<b>3</b>	The function was able to establish a connection, but the version requirement was not met.

**Table 238 • IS\_SQLSERVER\_STATUS Property Values (cont.)**

Return Value	Description
4	The function was able to establish a connection, but MSDE or the Express version was not allowed.
5	The function was able to establish a connection, but it failed to retrieve the schema version from the target database catalog.
6	The function was able to establish a connection, but it failed to set the schema version to the target database catalog.
7	The function could not find the specified ODBC driver.
8	The function was able to establish a connection to the default database catalog, but it failed to create the target database catalog (when <b>Create Catalog If Absent</b> check box in the SQL Scripts view is selected).
9	The function was able to establish a connection to the default database catalog, but it failed to connect to the target database catalog.
10	The function could not find a valid record in <b>ISSQLDBMetaData</b> table.

## SQLServerLogin



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SQLServerLogin** function creates a dialog that is used by the script to specify SQL login credentials. These credentials include the login ID and password.

This function is in the **SQLRT.obl** file for InstallScript projects, and in the **SQLCONV.obl** file for InstallScript MSI projects. If you are using the SQL Scripts view in InstallShield, the appropriate file is automatically added to your linker settings. However, if you are not using this view, add the appropriate file to your linker settings: On the Build menu, click Settings, and then add it to the Libraries (.obl) box.

### Syntax

```
SQLServerLogin( szMsg, svUser, svPassword );
```



## Parameters

**Table 239** • SQLServerLogin Parameters

Parameter	Description
<b>szMsg</b>	Specifies the message to display in the dialog. To display the default instructions for this dialog, pass a null string ("" ) in this parameter.
<b>svUser</b>	Specifies the default string to display in the Login ID edit box. Once the dialog function returns, this parameter contains the Login ID that the end user entered in the edit box. This information is only pertinent if the end user selected to use SQL Login instead of Windows credentials.
<b>svPassword</b>	Specifies the default string to display in the Password edit field. Once the dialog function returns, this parameter contains the password that the end user entered in the edit field. This information is only pertinent if the end user selected to use SQL Login instead of Windows credentials.

## Return Values

**Table 240** • SQLServerLogin Return Values

Return Value	Description
<b>NEXT</b>	The end user selected the Next button.
<b>BACK</b>	The end user selected the Back button.
<b>&lt; ISERR_SUCCESS</b>	The dialog could not be displayed.

# SQLServerSelect



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SQLServerSelect** function creates a dialog that lets the end user select a database server.

This function is in the **SQLRT.obl** file for InstallScript projects, and in the **SQLCONV.obl** file for InstallScript MSI projects. If you are using the SQL Scripts view in InstallShield, the appropriate file is automatically added to your linker settings. However, if you are not using this view, add the appropriate file to your linker settings: On the Build menu, click Settings, and then add it to the Libraries (.obl) box.

## Syntax

```
SQLServerSelect( szMsg, svServer );
```

## Parameters

**Table 241** • SQLServerSelect Parameters

Parameter	Description
<b>szMsg</b>	Specifies the message to display in the dialog. To display the default instructions for this dialog, pass a null string ("") in this parameter.
<b>szServer</b>	Specifies the default server to initially display in the Server combo box. Once the dialog function returns, this parameter contains the server name that the end user selected or entered in the Server combo box.

## Return Vaues

**Table 242** • SQLServerSelect Return Values

Return Value	Description
<b>NEXT</b>	The end user selected the Next button.
<b>BACK</b>	The end user selected the Back button.
<b>&lt; ISERR_SUCCESS</b>	The dialog could not be displayed.

# SQLServerSelectLogin



**Project** • This information applies to the following project types:

- InstallScript
- InstallScript MSI

The [SQLServerSelectLogin2](#) function supersedes the **SQLServerSelectLogin** function.

The **SQLServerSelectLogin** function creates a dialog that is used by the default script. It allows the targeted end user to specify which SQL Server is to be used for the current connection, along with which login credential should be used. When the dialog is displayed, a combo box shows a list of SQL Servers accessed through DSNs. A Browse button is available to bring up a list of all SQL Servers available on the network or the end user can type a server name into the combo box. Also, the end user has the option to use Windows login credentials or enter a SQL Server login ID and password.

This function is in the **SQLRT.obl** file for InstallScript projects, and in the **SQLCONV.obl** file for InstallScript MSI projects. If you are using the SQL Scripts view in InstallShield, the appropriate file is automatically added to your linker settings. However, if you are not using this view, add the appropriate file to your linker settings: On the Build menu, click Settings, and then add it to the Libraries (.obl) box.

## Syntax

```
SQLServerSelectLogin (byref string svServer, byref string svUser, byref string svPassword, byref BOOL
    bvWindowsLogin );
```

## Parameters

**Table 243** • SQLServerSelectLogin Parameters

Parameter	Description
<b>szServer</b>	<p>Specifies the default server to initially display in the Server combo box. Once the dialog function returns, this parameter contains the server name the end user selected or entered in the Server combo box.</p> <p>This parameter supports alias names for SQL Server database connections.</p>
<b>svUser</b>	<p>Specifies the default string to display in the Login ID edit box. Once the dialog function returns, this parameter contains the Login ID that the end user entered in the edit box. This information is only pertinent if the end user selected to use SQL Login instead of Windows credentials.</p>
<b>svPassword</b>	<p>Specifies the default string to display in the Password edit field. Once the dialog function returns, this parameter contains the Password that the end user entered in the edit field. This information is only pertinent if the end user selected to use SQL Login instead of Windows credentials.</p>
<b>bvWindowsLogin</b>	<p>Specifies the initial state of the radio buttons specifying login using Windows credentials vs. SQL Server credentials. When this is TRUE, the password and login ID fields are disabled. When it is FALSE, indicating that SQL Server credentials are to be used, the password and login fields are enabled.</p> <p>Once the function returns, this parameter contains TRUE if the Windows credentials radio button was specified by the end user and FALSE if the SQL Server credentials radio was selected instead.</p>

## Return Values

**Table 244** • SQLServerSelectLogin Return Values

Return Value	Description
<b>NEXT</b>	The end user selected the Next button.
<b>BACK</b>	The end user selected the Back button.
<b>&lt; ISERR_SUCCESS</b>	The dialog could not be displayed.

# SQLServerSelectLogin2



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **SQLServerSelectLogin2** function creates a login dialog that is used by the default script. It lets the targeted end user specify which SQL Server should be used for the current connection, as well as which login credential should be used. The dialog displays a combo box that contains a list of SQL Servers accessed through DSNs. The end user can type a server name in the combo box or click the Browse button next to the Server Name combo box; clicking this button displays a list of all SQL Servers that are available on the network. The end user has the option to use Windows login credentials or enter a SQL Server login ID and password.

When `bShowCxnName` is set to `TRUE`, the dialog shows the connection name that is associated with the connection information. In addition, when `bShowDBCatalog` is set to `TRUE`, the dialog allows the end user to specify which database catalog should be used for the current connection. The end user can type a catalog name in the edit box or click the Browse button next to the name of database catalog edit box. Clicking the Browse button displays a list of all database catalogs that are available on the specified database server.

## Syntax

```
SQLServerSelectLogin2( szConnection, svServer, svUser, svPassword, bvWindowsLogin, svCatalog,  
    bShowCxnName, bShowDBCatalog );
```

## Parameters

**Table 245 • SQLServerSelectLogin2 Parameters**

Parameter	Description
<b>szConnection</b>	Specifies the SQL Server connection.
<b>szServer</b>	<p>Specifies the default server to initially display in the Server combo box. Once the dialog function returns, this parameter contains the server name the end user selected or entered in the Server combo box.</p> <p>This parameter supports alias names for SQL Server database connections.</p>
<b>svUser</b>	Specifies the default string to display in the Login ID edit box. Once the dialog function returns, this parameter contains the Login ID that the end user entered in the edit box. This information is only pertinent if the end user selected to use SQL Login instead of Windows credentials.
<b>svPassword</b>	Specifies the default string to display in the Password edit field. Once the dialog function returns, this parameter contains the Password that the end user entered in the edit field. This information is only pertinent if the end user selected to use SQL Login instead of Windows credentials.
<b>bvWindowsLogin</b>	<p>Specifies the initial state of the radio buttons specifying login using Windows credentials vs. SQL Server credentials. When this is TRUE, the password and login ID fields are disabled. When it is FALSE, indicating that SQL Server credentials are to be used, the password and login fields are enabled.</p> <p>Once the function returns, this parameter contains TRUE if the Windows credentials radio button was specified by the end user and FALSE if the SQL Server credentials radio was selected instead.</p>
<b>svCatalog</b>	Specifies the name of the database catalog.
<b>bShowCxnName</b>	<p>Specifies whether the dialog should show the name of the connection that is associated with the connection information. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>TRUE</b>—Show the name of the connection.</li> <li>• <b>FALSE</b>—Do not show the name of the connection.</li> </ul>

**Table 245** • SQLServerSelectLogin2 Parameters (cont.)

Parameter	Description
<b>bShowDBCatalog</b>	<p>Specifies whether the end user can select which database catalog should to be used for the current connection. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>TRUE</b>—Allow the end user to select which database catalog should to be used for the current connection. The end user can type a catalog name in the edit box or click the Browse button next to the name of database catalog edit box. Clicking this Browse button displays a list of all database catalogs that are available on the specified database server.</li> <li>• <b>FALSE</b>—Do not allow the end user to select which database catalog should to be used for the current connection.</li> </ul>

## Return Values

**Table 246** • SQLServerSelectLogin2 Return Values

Return Value	Description
<b>NEXT</b>	The end user selected the Next button.
<b>BACK</b>	The end user selected the Back button.
<b>&lt; ISERR_SUCCESS</b>	The dialog could not be displayed.

# SQLServerSelectLoginEx



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The [SQLServerSelectLogin2](#) function supersedes the **SQLServerSelectLoginEx** function.

The **SQLServerSelectLoginEx** function creates a login dialog that is used by the default script. It lets the targeted end user specify which SQL Server should be used for the current connection, as well as which login credential should be used. The dialog also shows the connection name that is associated with the connection information.

The dialog displays a combo box that contains a list of SQL Servers accessed through DSNs. The end user can type a server name in the combo box or click the Browse button next to the Server Name combo box; clicking this button displays a list of all SQL Servers that are available on the network. The end user has the option to use Windows login credentials or enter a SQL Server login ID and password.

## Syntax

```
SQLServerSelectLoginEx (byval string szConnection, byref string svServer, byref string svUser,
    byref string svPassword, byref BOOL bvWindowsLogin)
```

## Parameters

**Table 247** • SQLServerSelectLoginEx Parameters

Parameter	Description
<b>szConnection</b>	Specifies the SQL Server connection.
<b>szServer</b>	<p>Specifies the default server to initially display in the Server combo box. Once the dialog function returns, this parameter contains the server name the end user selected or entered in the Server combo box.</p> <p>This parameter supports alias names for SQL Server database connections.</p>
<b>svUser</b>	Specifies the default string to display in the Login ID edit box. Once the dialog function returns, this parameter contains the Login ID that the end user entered in the edit box. This information is only pertinent if the end user selected to use SQL Login instead of Windows credentials.
<b>svPassword</b>	Specifies the default string to display in the Password edit field. Once the dialog function returns, this parameter contains the Password that the end user entered in the edit field. This information is only pertinent if the end user selected to use SQL Login instead of Windows credentials.
<b>bvWindowsLogin</b>	<p>Specifies the initial state of the radio buttons specifying login using Windows credentials vs. SQL Server credentials. When this is TRUE, the password and login ID fields are disabled. When it is FALSE, indicating that SQL Server credentials are to be used, the password and login fields are enabled.</p> <p>Once the function returns, this parameter contains TRUE if the Windows credentials radio button was specified by the end user and FALSE if the SQL Server credentials radio was selected instead.</p>

## Return Values

**Table 248** • SQLServerSelectLoginEx Return Values

Return Value	Description
<b>NEXT</b>	The end user selected the Next button.
<b>BACK</b>	The end user selected the Back button.
<b>&lt; ISERR_SUCCESS</b>	The dialog could not be displayed.

## StatusUpdate

The **StatusUpdate** function enables or disables the link between file transfer operations and the progress indicator of the status bar. When **bLink** is ON, the link is enabled and **nFinalPercent** specifies a final percentage to be displayed at the end of the next file transfer. During that file transfer, the status bar is updated smoothly from its current value to the value specified by **nFinalPercent**. When **bLink** is OFF, the link is disabled and the progress

indicator of the status bar is not updated automatically during subsequent file transfers.

This function works by computing the total number of bytes to be transferred by any of the file transfer functions. It then computes how often it will increment the progress bar starting from its current location to the maximum value in `nFinalPercent`.

The `StatusUpdate` function does not work with the functions `VerUpdateFile` and `VerSearchAndUpdateFile`. When calling those functions, you should disable the status bar or update it manually.

To set the status bar to an initial percentage, call `SetStatusWindow` before calling `StatusUpdate`.



---

**Tip** • If the status bar is enabled during file transfer, call `StatusUpdate` before each call to `CopyFile` or `XCopyFile`). Before calling `FeatureMoveData` to transfer files, call `StatusUpdate` with the parameters `ON` and `100`. This sets the status bar to update smoothly to 100% during the file transfer stage of the setup.


## Syntax

```
StatusUpdate ( bLink, nFinalPercent );
```



## Parameters

**Table 249 • StatusUpdate Parameters**

Parameter	Description
<b>bLink</b>	<p>Specifies whether to enable or disable the link between file transfer operations and the progress indicator of the status bar. Pass one of the following predefined constants in this parameter:</p> <ul style="list-style-type: none"> <li>• <b>ON</b>—Specifies that the progress indicator of the status bar should be linked to file transfer operations.</li> <li>• <b>OFF</b>—Specifies that link between file transfer operations and the progress indicator of the status bar should be disabled. The link remains disabled until it is reestablished by a subsequent call to StatusUpdate with bLink set to ON.</li> </ul>
 <p><b>Note</b> • The status bar can be updated manually by calling <i>SetStatusWindow</i>.</p>	
<b>nFinalPercent</b>	<p>Specifies the final percentage value that the progress indicator of the status bar should reach at the end of the next file transfer operation if bLink is ON. If the value passed in nFinalPercent is less than the current value in the progress indicator of the status bar, the progress indicator will not change. When bLink is OFF, this parameter is ignored.</p>

## Return Values

**Table 250 • StatusUpdate Return Values**

Return Value	Description
<b>0</b>	Indicates that the function was successful.
<b>&lt; 0</b>	Indicates that the function was not successful.

## StatusUpdate Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the StatusUpdate function.
*
```

```

* This script copies all files from a source directory to a
* target directory. StatusUpdate is called to set the limit
* of the progress bar and to regulate the progress indicator
* as the files are copied.
*
* Note: Before running this script, create the directories
*       referenced by SOURCE_DIR and TARGET_DIR and create two
*       or more files in SOURCE_DIR.
*
/*-----*/

#define SOURCE_DIR "C:\\ISExampl\\Source"
#define TARGET_DIR "C:\\ISExampl\\Target";

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_StatusUpdate(HWND);

function ExFn_StatusUpdate(hMSI)
begin

    // Enable the progress bar.
    Enable(STATUS);

    // Set the limit of the progress bar to 99% completion.
    StatusUpdate (ON, 99);

    // Copy the files.
    if (XCopyFile (SOURCE_DIR ^ ".*", TARGET_DIR ^ ".*", COMP_NORMAL) < 0 ) then
        MessageBox ("An error occurred while copying files.",SEVERE);
    endif;

    // Display a message; do not change the progress bar.
    SetStatusWindow (-1, "File Copying completed at 99%.");
    Delay (3);

    // Set the progress bar to 100% and displays a message.
    SetStatusWindow (100, "StatusUpdate example completed, now exiting...");
    Delay (3);

end;

```

## StrAddLastSlash

The **StrAddLastSlash** function adds a trailing backslash to a path specification if it does not already have one.

### Syntax

```
StrAddLastSlash ( svPath );
```

## Parameters

**Table 251** • StrAddLastSlash Parameters

Parameter	Description
<b>svPath</b>	Specifies a string whose value must be a path specification; it returns the path with a trailing backslash. Note that if the path includes a trailing backslash, it is returned unchanged.

## Return Values

**Table 252** • StrAddLastSlash Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the function successfully added the trailing backslash or that the path contains a trailing backslash.

# StrCompare

The **StrCompare** function compares two strings. The comparison is not case-sensitive.

## Syntax

```
StrCompare ( szStringA, szStringB );
```

## Parameters

**Table 253 • StrCompare Parameters**

Parameter	Description
szStringA	Specifies the first string to compare.
szStringB	Specifies the second string to compare.

## Return Values

**Table 254 • StrCompare Return Values**

Return Value	Description
< 0	Indicates that the string in szStringA has a lower value than the string in szStringB.
0	Indicates that the two strings are equal.
> 0	Indicates that the string in szStringA has a greater value than the string in szStringB.

## Additional Information

The **StrCompare** function compares the strings by checking the first character in each string, the second character in each string, and so on—until it finds an inequality or reaches the ends of the strings.

The language driver for the language that you select determines which string is greater or if the strings are the same. If you do not use a language driver, Windows uses an internal function. With a double-byte character set (DBCS) version of Windows, this function can compare two DBCS strings.

## StrCompare Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the StrCompare function.
*
* StrCompare is called to compare two strings. The result of
* the comparison is displayed in a message box.
*
\*-----*/

#define TITLE_TEXT "StrCompare Example"
#define MSG_TEXT  "Please enter two strings to compare:"
```

```

#define FIELD_A    "String A:"
#define FIELD_B    "String B:"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_StrCompare(HWND);

function ExFn_StrCompare(hMSI)
    STRING svStringA, svStringB;
    NUMBER nResult;
begin
    // Get two strings from the user.
    SdShowDlgEdit2 (TITLE_TEXT, MSG_TEXT, FIELD_A, FIELD_B, svStringA, svStringB);

    // Compare the strings.
    nResult = StrCompare (svStringA, svStringB);

    // Display the result.
    if (nResult = 0) then
        MessageBox (svStringA + " and " + svStringB + " are equal.", INFORMATION);
    elseif (nResult < 0) then
        MessageBox (svStringB + " is greater than " + svStringA, INFORMATION);
    else
        MessageBox (svStringA + " is greater value than " + svStringB, INFORMATION);
    endif;

end;

```

## StrConvertSizeUnit

The **StrConvertSizeUnit** function returns the appropriate display string for the InstallScript size unit constant that is specified.

### Syntax

```
StrConvertSizeUnit (byval string nUnit);
```

## Parameters

**Table 255** • StrConvertSizeUnit Parameters

Parameter	Description
nUnit	Specify the unit constant that you want to be converted to a UI string. The following constants are supported: <ul style="list-style-type: none"> <li>• BYTES</li> <li>• KBYTES</li> <li>• MBYTES</li> <li>• GBYTES</li> <li>• TBYTES</li> </ul>

## Return Values

**Table 256** • StrConvertSizeUnit Return Values

Return Value	Description
<b>bytes</b>	The function converted the BYTES constant.
<b>KB</b>	The function converted the KBYTES constant.
<b>MB</b>	The function converted the MBYTES constant.
<b>GB</b>	The function converted the GBYTES constant.
<b>TB</b>	The function converted the TBYTES constant.
<b>Null ("" )</b>	The function failed to convert nUnit.

# StreamFileFromBinary

The **StreamFileFromBinary** function streams a binary key with a file.

## Syntax

```
StreamFileFromBinary ( hInstall(HWND), svObjectBinaryKey, szFileName );
```

## Parameters

**Table 257** • StreamFileFromBinary Parameters

Parameter	Description
<b>hInstall (HWND)</b>	Provides a handle to the currently running application.
<b>svObjectBinaryKey</b>	Indicates the key to be streamed out of the binary table.
<b>szFileName</b>	Indicates the full path to the extracted file.

## Return Values

**Table 258** • StreamFileFromBinary Return Values

Return Value	Description
<b>0</b>	The function successfully streamed the binary key with the file.
<b>-1</b>	The function failed to stream the binary key with the file.

# StrFind

The **StrFind** function determines whether the string passed in the parameter `szFindMe` is found within the string passed in the parameter `szString`. If `szFindMe` is found in `szString`, **StrFind** returns the position within `szString` of the first character of `szFindMe`. (Note that the position of the first character in `szString` is zero.) This function is not case-sensitive and can be used only to find the first occurrence of `szFindMe` in `szString`.

StrFind calls the following:

```
StrFindEx(szString, szFindMe, 0);
```

For more information about parameters and return values for **StrFind**, see [StrFindEx](#).

## StrFind Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the StrFind function.
 *
```

```

* StrFind is called to search for a substring within a string.
* If successful, StrFind returns the location of the substring.
*
\*-----*/

#define TITLE_TEXT "StrFind Example";

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_StrFind(HWND);

function ExFn_StrFind(hMSI)
    STRING szString, szFindMe, szTitle, szMsg;
    NUMBER nLocation;
begin

    // Set up parameters for call to StrFind.
    szString = "This is a sample string to be searched.";
    szFindMe = "Sample String";

    // Find the substring specified by szFindMe.
    nLocation = StrFind (szString, szFindMe);

    // Display the location of the text if it was found.
    if (nLocation < 0) then
        MessageBox (szFindMe + " not found in " + szString + ".", WARNING);
    else
        szMsg = "'%s' starts at byte %d of\n'%s'";
        sprintfBox (INFORMATION, szTitle, szMsg, szFindMe, nLocation, szString);
    endif;

end;

```

## StrFindEx

The **StrFindEx** function determines whether the string passed in the parameter `szFindMe` is found within the string passed in the parameter `szString`; the function begins its search at the location specified by `nStart`. If `szFindMe` is found, `StrFind` returns the position within `szString` of the first character of `szFindMe`. This function is not case-sensitive and can be used only to find the first occurrence of `szFindMe` in `szString`.

### Syntax

```
StrFindEx ( szString, szFindMe, nStart );
```



## Parameters

**Table 259** • StrFindEx Parameters

Parameter	Description
<b>szString</b>	Specifies the string to search.
<b>szFindMe</b>	Specifies the string to find in szString.
<b>nStart</b>	Specifies an offset into szString that identifies the character at which to begin the search for szFindMe. Note that the position of the first character in szString is 0 (zero).

## Return Values

**Table 260** • StrFindEx Return Values

Return Value	Description
<b>X</b>	If szString contains szFindMe, X is the numeric position of the first character in szFindMe. The first character in szString is in position 0 (zero).
<b>&lt; 0</b>	Indicates szString does not contain szFindMe.

## Additional Information

In the example below, StrFind will return the value 19.

```
nStartPos = StrFind("Scripting means having fun","ing",7);
```

If you need only a TRUE or FALSE return value indicating whether one string contains another string, (that is, if the location if the substring is unimportant), use the string find operator (%) as shown below:

```
if (szString % szFindMe) then ...
```

You can use the string find operator (%) only in Boolean expressions that are resolved in if statements. You cannot use it in repeat statements or while statements.

# StrGetTokens

The **StrGetTokens** function extracts substrings (tokens) from the string specified by szString and places them into the list specified by listID. The substrings in szString must be delimited—separated from one another—by one or more of the characters specified by szDelimiterSet.

For example, if you call StrGetTokens with the string "One;Two;Three;Four;Five" as the second parameter and ";" as the third parameter, listID will be returned with five strings: "One", "Two", "Three", "Four", and "Five". Use the list functions, such as ListGetFirstString and ListGetNextString to access each token in the list.



**Note** • If the first (or last) character of *szString* matches a character in *szDelimiterSet*, a null string ("" ) is not inserted in the list as the first (or last) element. Instead, the characters between the first and second (or last and next to last) delimiters are inserted in the list as the first (or last) element.


Syntax

StrGetTokens ( listID, szString, szDelimiterSet );

Parameters

Table 261 • StrGetTokens Parameters

Parameter	Description
listID	Returns a list of tokens. The string list identified by listID must already have been initialized by a call to <a href="#">ListCreate</a> .
szString	Specifies the string to be parsed.
szDelimiterSet	Specifies a set of one or more delimiters. Each delimiter is one character (1 byte). If you pass a null string ("" ) in this parameter, the function searches for null characters as the delimiters. This is useful if you are using the <a href="#">GetProfString</a> function.



**Note** • When a space is specified as the delimiter, *StrGetTokens* treats consecutive spaces as a single delimiter.

Return Values

Table 262 • StrGetTokens Return Values

Return Value	Description
0	Indicates that the function successfully separated the string and inserted the tokens into the specified list.
< 0	Indicates that the function was unable to separate the string and insert the tokens into the list.

StrGetTokens Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the StrGetTokens function.
*
* First, the value of a key is retrieved from an initialization
* file. The value returned is a search path. Then
* StrGetTokens is called to build a list of the paths found in
* the search path. Finally, the paths are displayed.
*
* In order for this script to run correctly, you must create
* an initialization (text) file and add the following lines to
* it:
*
* [Test Section]
* searchpath=C:\Windows;C:\Windows\System;C:\Windows\Command
*
* Then set the constant EXAMPLE_INI to the fully qualified name
* of that file.
*
\*-----*/

#define FILE_NAME    "C:\\ISExempl.ini"
#define SECTION_NAME "Test Section"
#define KEY_NAME     "searchpath"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_StrGetTokens(HWND);

function ExFn_StrGetTokens(hMSI)
    LIST listID;
    STRING svSearchPath;
    STRING szTitle, szMsg;
begin

    // Get the value of the specified key from the specified section
    // of the initialization file.
    if GetProfString (FILE_NAME, SECTION_NAME, KEY_NAME, svSearchPath) < 0 then
        // Report the error.
        MessageBox ("Unable to retrieve key value from "+ FILE_NAME + ".", INFORMATION);
    else
        // Create a list to hold the paths that make up the search path
        // that was returned by GetProfString.
        listID = ListCreate (STRINGLIST);

        // Get each path from the search path into the list.
        if (StrGetTokens (listID, svSearchPath, ";") < 0) then
            // Report the error.
            MessageBox ("StrGetTokens failed.", SEVERE);
        else
            // Display the individual paths from the list.
            Sprintf (szMsg, "Paths found in %s %s", SECTION_NAME, KEY_NAME);
            SdShowInfoList ("Search Path", szMsg, listID);
        endif;
    endif;
end;

```

```
// Remove the list from memory.
ListDestroy (listID);

end;
```

## StrLength

The **StrLength** function returns the number of characters in a given string variable (that is, the number of code units in the UTF-16-encoded string) up to the first null character.

**StrLength** and [StrLengthChars](#) return the same results.

To obtain the number of characters in a string with embedded nulls, use [SizeOf](#).

### Syntax

```
StrLength ( szString );
```

### Parameters

**Table 263** • StrLength Parameters

Parameter	Description
<b>szString</b>	Specifies the string whose size is to be determined.

### Return Values

**Table 264** • StrLength Return Values

Return Value	Description
<b>X</b>	Where X is the number of characters in the string.
<b>&lt; 0</b>	Indicates that the function was unable to determine the number of characters in the string.

## StrLength Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the StrLength function.
 *
 * StrLength is called to retrieve the number of characters in a
 * string that an end user enters. The number of characters is
```

```

    * displayed in a message box.
    *
    \*-----*/

#define TITLE_TEXT "StrLength Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_StrLength(HWND);

function ExFn_StrLength(hMSI)
    STRING svString, szTitle, szMsg;
    NUMBER nLength;
begin
    // Get a line of text from the end user.
    AskText ("Enter a line of text. ", "", svString);

    // Get the number of characters in the string.
    nLength = StrLength (svString);

    if (nLength < 0) then
        MessageBox ("StrLength failed.", SEVERE);
    else
        // Display the string length.
        szMsg = "String '%s' is %d characters long.";
        sprintfBox (INFORMATION, TITLE_TEXT, szMsg, svString, nLength);
    endif;

end;

```

## StrLengthChars

The **StrLengthChars** function returns the number of characters in a given string variable (that is, the number of code units in the UTF-16-encoded string) up to the first null character.

**StrLengthChars** and [StrLength](#) return the same results.

To obtain the number of characters in a string with embedded nulls, use [SizeOf](#).

### Syntax

```
StrLengthChars ( szString );
```

## Parameters

**Table 265** • StrLengthChars Parameters

Parameter	Description
<b>szString</b>	Specifies the string whose size is to be determined.

## Return Values

**Table 266** • StrLengthChars Return Values

Return Value	Description
<b>X</b>	Where X is the number of characters in the string.
<b>&lt; 0</b>	Indicates that the function was unable to determine the number of characters in the string.

## StrLengthChars Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the StrLengthChars function.
 *
 * StrLengthChars is called to retrieve the number of characters
 * in a string that an end user enters. The number of characters is
 * displayed in a message box.
 *
 \*-----*/

#define TITLE_TEXT "StrLengthChars Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_StrLengthChars(HWND);

function ExFn_StrLengthChars(hMSI)
    STRING svString, szTitle, szMsg;
    NUMBER nLength;
begin

    // Get a line of text from the end user.
    AskText ("Enter a line of text. ", "", svString);

    // Get the number of characters in the string.
    nLength = StrLengthChars (svString);

    if (nLength < 0) then
        MessageBox ("StrLengthChars failed.", SEVERE);
    else

```

```

        // Display the number of code units in the string.
        szMsg  = "String '%s' is %d code units long.";
        sprintfBox (INFORMATION, TITLE_TEXT, szMsg, svString, nLength);
    endif;

end;

```

## StrPutTokens

The **StrPutTokens** function extracts list items from the string list specified by listID and places them into the string specified by svString. The substrings placed into svString from listID are separated by szSeparator in the resulting string. Note that instances of szSeparator are *not* placed at the beginning and end of the string, only between list elements added to the string.

For example, if you call StrPutTokens with the list containing the items "One", "Two", "Three", "Four", and "Five" as the first parameter and ";" as the third parameter, svString will contain "One;Two;Three;Four;Five" after the function is called.

The value of bNull determines whether null characters are used to separate the substrings in the resulting string.

### Syntax

```
StrPutTokens( listID, svString, szSeparator, bNull );
```

## Parameters

**Table 267** • StrPutTokens Parameters

Parameter	Description
<b>listID</b>	Specifies the string list to be processed. The string list identified by listID must already have been initialized by a call to ListCreate. Note that this function does not work with number lists.
<b>svString</b>	Specifies the resulting string.
<b>szSeparator</b>	Specifies a string to be used to separate each list item in the resulting string. Note that unlike the StrGetTokens function, passing a null string in this parameter will <i>not</i> result in a null separated string; it will result in the substrings being placed one after the other with no separators. Set bNULL to TRUE to create a null separated string.
<b>bNull</b>	If bNull is set to TRUE, null characters will be used to separate the substrings in the resulting string, and szSeparator will be ignored. Note that the resulting string will be double null terminated in this case. If bNull is set to FALSE, the substrings in the resulting string will be separated by szSeparator.

## Return Values

**Table 268** • StrPutTokens Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully created the string from the specified string list.
<b>&lt; 0</b>	Indicates that the function was unable to create the specified string from the specified string list.

# StrRemoveLastSlash

The **StrRemoveLastSlash** function removes the trailing backslash from a path specification. Because its purpose is to produce a valid path, StrRemoveLastSlash does not remove the backslash from a root directory specification, such as "A:\" or "C:\".

## Syntax

```
StrRemoveLastSlash ( svPath );
```



## Parameters

**Table 269** • StrRemoveLastSlash Parameters

Parameter	Description
<b>svPath</b>	Specifies a string whose value must be a path specification; returns the path without the trailing backslash. Note that if the path does not include a trailing backslash, it is returned unchanged.

## Return Values

**Table 270** • StrRemoveLastSlash Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully removed the trailing backslash or that the path does not contain a trailing backslash.
<b>&lt; 0</b>	Indicates that the function was unable to remove the trailing backslash.

## Additional Information

StrRemoveLastSlash provides a convenient way to remove the trailing backslash from a path returned by AskPath or ParsePath. Because its purpose is to produce a valid path, StrRemoveLastSlash does not remove the backslash from a root directory specification, such as "A:\" or "C:\"; doing so would turn a valid path into a drive specification. If you need to remove the trailing backslash from a path in all cases, use the following script segment as a guide.

```
AskPath("", "", svPath);

if (StrLength(svPath) = 3)
    && (svPath[1] = ":")
    && (svPath[2] = "\\") then

    svTempString = svPath;
    StrSub(svPath, svTempString, 0, 2);
else
    StrRemoveLastSlash(svPath);
endif;
```

## StrRemoveLastSlash Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the StrRemoveLastSlash function.
*
```

```

* The AskPath dialog is displayed, prompting the user to
* specify a path. StrRemoveLastSlash is then called to remove
* the trailing backslash from the path returned by AskPath.
*
\*-----*/

#define TITLE_TEXT "StrRemoveLastSlash Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_StrRemoveLastSlash(HWND);

function ExFn_StrRemoveLastSlash(hMSI)
    STRING szDefaultPath, svString, szMsg, szTitle;
begin

    // Get a path from the user. AskPath returns the
    // path with a trailing backslash.
    AskPath ("Specify a path:", INSTALLDIR, svString);

    // Remove the last slash from the path.
    if (StrRemoveLastSlash (svString) < 0) then
        // Report an error.
        MessageBox ("StrRemoveLastSlash failed.", SEVERE);
    else
        // Display the folder name after the last backslash
        // has been removed.
        MessageBox ("Specified path: " + svString, INFORMATION);
    endif;

end;

```

## StrReplace

The **StrReplace** function searches svResult, beginning at the location specified by nStart, and replaces all found instances of szFind with szReplace.

### Syntax

```
StrReplace ( svResult, szFind, szReplace, nStart );
```

## Parameters

**Table 271** • StrReplace Parameters

Parameter	Description
<b>svResult</b>	Specifies the string to search. Returns the modified string.
<b>szFind</b>	Specifies the string to find in svResult.
<b>szReplace</b>	Specifies the string with which to replace found instances of szFind.
<b>nStart</b>	Specifies an offset into svResult that identifies the character at which to begin the search for szFind. Note that the position of the first character in szString is 0 (zero). If you want to replace all instances of szFind in svResult, specify 0 for nStart.

## Return Values

**Table 272** • StrReplace Return Values

Return Value	Description
<b>X</b>	The total number of replacements of szFind by szReplace.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function failed. A value less than ISERR_SUCCESS is returned if szFind is a null string ("") or if nStart is greater than the length of svResult.

# StrSub

The **StrSub** function copies part of the string specified by szString, beginning at the location specified by nStart. The parameter nLength specifies the number of characters to copy.

## Syntax

```
StrSub ( svSubStr, szString, nStart, nLength );
```

## Parameters

**Table 273 • StrSub Parameters**

Parameter	Description
<b>svSubStr</b>	Returns the substring copied from szString.
<b>szString</b>	Specifies the string from which the substring is to be copied.
<b>nStart</b>	Specifies an offset into szString that identifies the first character to be copied. Note that the position of the first character in szString is 0 (zero). If the value passed in nStart is equal to or greater than the length of szString, a null string ("" ) is returned in svSubStr.
<b>nLength</b>	Specifies the number of characters to copy from szString. If this value specifies more characters than exist between nStart and the end of szString, all characters from nStart to the end of the string are returned in svSubStr.

## Return Values

**Table 274 • StrSub Return Values**

Return Value	Description
<b>X</b>	Where X equals the number of characters in svSubStr.

## StrSub Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the StrSub function.
*
* This script gets a date from the user and then calls StrSub
* to extract the four-digit year. Finally, the year is
* displayed in a message box.
*
\*-----*/

#define TITLE_TEXT "StrSub Example"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_StrSub(HWND);
```

```

function ExFn_StrSub(hMSI)
    STRING svYear, svDate, szQuestion, szTitle, szMsg;
    NUMBER nYear;
    BOOL bDateOk;
begin

    // Set up message parameter for call to AskText.
    szQuestion = "Please enter the date in this form MM/DD/YYYY:";

    repeat
        // Get a date from the user.
        AskText (szQuestion, "09/28/1998", svDate);

        // Check the format of the date entered by the user.
        bDateOk = (StrLength(svDate) = 10) &&
            (svDate[2] = "/" ) &&
            (svDate[5] = "/" );

        // If the date format is incorrect, report an error.
        if !bDateOk then
            MessageBox(svDate + " is not a date in the specified format.", WARNING);
        endif;
    until bDateOk ;

    // Retrieve the four-digit year, which starts at byte six.
    if (StrSub (svYear, svDate, 6, 4) < 0) then
        MessageBox ("StrSub failed.", SEVERE);
    endif;

    // Validate the year field.
    if StrToNum(nYear, svYear) = 0 then
        // Display the edited string.
        szMsg = "You specified the year %s";
        sprintfBox (INFORMATION, TITLE_TEXT, szMsg, svYear);
    else
        MessageBox(svDate + " is not a date in the specified format.", WARNING);
    endif;

end;

```

## STRTOCHAR

The **STRTOCHAR** function returns the first character of szString as data of type CHAR.

### Syntax

```
STRTOCHAR ( szString );
```

## Parameters

**Table 275 • STRTOCHAR Parameters**

Parameter	Description
<b>szString</b>	Specifies the string whose first character will be returned as data of type CHAR.

## Return Values

**Table 276 • STRTOCHAR Return Values**

Return Value	Description
<b>char</b>	The first character of szString expressed as data of type CHAR.

## Additional Information

The STRTOCHAR function is useful when passing string literals to functions. Since InstallScript interprets double and single quotes as string delimiters, attempting to pass a literal character, for example, 'a', to a function that expects a character will cause a compiler error. To avoid this problem, pass the character literal to STRTOCHAR and pass the result of calling this function as the argument; for example:

```
Sprintf( szString, "%c", STRTOCHAR('a') );
```

# StrToLower

The **StrToLower** function converts all the letters in a string to lowercase. This function does not affect non-alphabetic characters.

## Syntax

```
StrToLower ( svTarget, szSource );
```

## Parameters

**Table 277** • StrToLower Parameters

Parameter	Description
<b>svTarget</b>	Returns the string in szSource with all characters converted to lowercase.
<b>szSource</b>	Specifies the string to convert to all lowercase characters.

## Return Values

**Table 278** • StrToLower Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully changed the case of the string.
<b>&lt; 0</b>	Indicates that the function was unable to change the case of the string.

## StrToLower Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the StrToUpper and StrToLower functions.
 *
 * StrToUpper is called to convert the characters in szSource
 * from lowercase to uppercase.
 *
 * StrToLower is then called to convert the uppercase
 * characters to lowercase.
 *
 \*-----*/

#define TITLE_TEXT "StrToUpper & StrToLower"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_StrToLower(HWND);

function ExFn_StrToLower(hMSI)

```

```

    STRING szSource, svTarget, szTitle, szMsg;
    NUMBER nReturn;
begin

    // Set up parameter for call to StrToUpper.
    szSource = "aBcDeF";

    // Convert all characters to upper case.
    nReturn = StrToUpper (svTarget, szSource);

    if (nReturn < 0) then
        MessageBox ("StrToUpper failed.", SEVERE);
    endif;

    szMsg = "Original: %s\n\nModified: %s";

    // Display the modified string.
    sprintfBox (INFORMATION, szTitle, szMsg, szSource, svTarget);

    // Set up parameter for call to StrToLower.
    szSource = "ABC123*&?d";

    // Convert all characters to lower case.
    nReturn = StrToLower (svTarget, szSource);

    if (nReturn < 0) then
        MessageBox ("StrToLower failed.", SEVERE);
    endif;

    // Display the modified string.
    szMsg = "Original: %s\n\nModified: %s\n(Note--Non-alphabetic chars " +
        "are not changed)";
    sprintfBox (INFORMATION, TITLE_TEXT, szMsg, szSource, svTarget);

end;

```

## StrToNum

The **StrToNum** function converts a string to a number, much like the C function `atol()`. It inspects `svString`, starting with the character at position 0 and continuing through the string until it reaches the end of the string or encounters a character that is not in the range "0"..."9". (The first character in the string can be a plus or minus sign.)

Then one of the following processes occurs:

- If all of the characters in the string are in the range "0"..."9", the number represented by the string is assigned to `nvVar`.
- If the string begins with one or more characters in the range "0"..."9" but also contains one or more non-numeric characters, a number based on the characters to the left of the first non-numeric character is assigned to `nvVar`. For example, if `szString` is "-123ABC456", `nvResult` will be -123.
- If the first character in the string is not in the range "0"..."9" and is not a plus or minus sign, the function fails.
- If the first character in the string is a plus or minus sign and the second character is not in the range "0"..."9", the function fails.



## Syntax

```
StrToNum ( nvVar, szString );
```

## Parameters

**Table 279** • StrToNum Parameters

Parameter	Description
<b>nvVar</b>	Returns the number created from the string in szString.
<b>szString</b>	Specifies the string to convert to a number. If the string corresponds to a number that is outside the range of allowed values for a number variable, this function gives unexpected results.

## Return Values

**Table 280** • StrToNum Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully converted the string into a numeric value.
<b>&lt; 0</b>	Indicates that the function was unable to convert the string into a numeric value.

## StrToNum Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the StrToNum function.
*
* StrToNum is called to convert "1222240" to 1222240 and
* "1222ABC40" to 1222.
*
\*-----*/

#define TITLE_TEXT "StrToNum Example"
#define MSG_TEXT   "String: %s\n\nNumber: %d"

// Include Ifx.h for built-in InstallScript function prototypes.
```

```

#include "Ifx.h"

export prototype ExFn_StrToNum(HWND);

function ExFn_StrToNum(hMSI)
    STRING szString, szMsg;
    NUMBER nVar;
begin

    // Convert a string with numeric characters to a number.
    szString = "1222240";

    if (StrToNum (nVar, szString) < 0) then
        MessageBox ("StrToNum failed.", SEVERE);
    else
        sprintfBox (INFORMATION, TITLE_TEXT, MSG_TEXT, szString, nVar);
    endif;

    // Convert string with non-numeric characters to a number.
    szString = "1222ABC40";

    if (StrToNum (nVar, szString) < 0) then
        MessageBox ("StrToNum failed.", SEVERE);
    else
        sprintfBox (INFORMATION, TITLE_TEXT, MSG_TEXT, szString, nVar);
    endif;

end;

```

## StrToNumHex

The **StrToNumHex** function converts a string to a number. It inspects `szString`, starting with the character at position 0 and continuing through the string until it reaches the end of the string or encounters a character that is not in the range "0"..."9", "a"..."f", "A"..."F". (The first two characters in the string may be "0x" or "0X".) Then one of the following processes occurs:

- If all of the characters in the string are in the range "0".."9", the hexadecimal number represented by the string is assigned to `nvVar`.
- If the string begins with one or more characters in the hexadecimal range but also contains one or more non-hexadecimal characters, a number based on the characters to the left of the first non-hexadecimal character is assigned to `nvVar`. For example, if `szString` is "0x1A2GHI456", `nvResult` will be 418 (0x1A2).
- If the first character in the string is not in the hexadecimal range, the function fails.
- If the first two characters in the string are "0x" or "0X" and the third character is not in the hexadecimal range, the function fails.

### Syntax

```
StrToNumHex ( nvVar, szString );
```

## Parameters

**Table 281** • StrToNumHex Parameters

Parameter	Description
<b>nvVar</b>	Returns the number created from the string in szString.
<b>szString</b>	Specifies the string to convert to a number.

## Return Values

**Table 282** • StrToNumHex Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the function successfully converted the string into a numeric value.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was unable to convert the string into a numeric value.

# StrToUpper

The **StrToUpper** function converts all the letters in a string to uppercase. This function does not affect non-alphabetic characters.

## Syntax

```
StrToUpper ( svTarget, szSource );
```

## Parameters

**Table 283** • StrToUpper Parameters

Parameter	Description
<b>svTarget</b>	Returns the string in szSource with all characters converted to uppercase.
<b>szSource</b>	Specifies the string to convert to all uppercase characters.

## Return Values

**Table 284** • StrToUpper Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully changed the case of the string.
<b>&lt; 0</b>	Indicates that the function was unable to change the case of the string.

## StrToUpper Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the StrToUpper and StrToLower functions.
 *
 * StrToUpper is called to convert the characters in szSource
 * from lowercase to uppercase.
 *
 * StrToLower is then called to convert the uppercase
 * characters to lowercase.
 *
 \*-----*/

#define TITLE_TEXT "StrToUpper & StrToLower"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_StrToUpper(HWND);

function ExFn_StrToUpper(hMSI)
```

```

    STRING szSource, svTarget, szTitle, szMsg;
    NUMBER nReturn;
begin

    // Set up parameter for call to StrToUpper.
    szSource = "aBcDeF";

    // Convert all characters to upper case.
    nReturn = StrToUpper (svTarget, szSource);

    if (nReturn < 0) then
        MessageBox ("StrToUpper failed.", SEVERE);
    endif;

    szMsg = "Original: %s\n\nModified: %s";

    // Display the modified string.
    sprintfBox (INFORMATION, szTitle, szMsg, szSource, svTarget);

    // Set up parameter for call to StrToLower.
    szSource = "ABC123*&?d";

    // Convert all characters to lower case.
    nReturn = StrToLower (svTarget, szSource);

    if (nReturn < 0) then
        MessageBox ("StrToLower failed.", SEVERE);
    endif;

    // Display the modified string.
    szMsg = "Original: %s\n\nModified: %s\n(Note--Non-alphabetic chars " +
        "are not changed)";
    sprintfBox (INFORMATION, TITLE_TEXT, szMsg, szSource, svTarget);

end;

```

## StrTrim

The **StrTrim** function removes the leading and trailing spaces and tabs from a string.

### Syntax

```
StrTrim (byref string svString);
```

## Parameters

**Table 285** • StrTrim Parameters

Parameter	Description
<b>svString</b>	Specify the string to be trimmed.

## Return Values

**Table 286** • StrTrim Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	The function successfully trimmed the string.
<b>&lt; ISERR_SUCCESS</b>	The function failed to trim the string.

# SuiteFormatString



**Project** • This information applies to the following project types:

- InstallScript
- Suite/Advanced UI



**Note** • This function is available for InstallScript installations that may be included as InstallScript packages in an Advanced UI or Suite/Advanced UI installation. For more information, see *Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project*.

This function is also available for InstallScript actions that are included in Suite/Advanced UI installations. For more information, see *Working with an Action that Runs InstallScript Code in a Suite/Advanced UI Installation*.

This function returns an error in the following scenarios:

- The function is called in an InstallScript installation that is launched directly (that is, not from an Advanced UI or Suite/Advanced UI installation).
- The function is called in an InstallScript installation that is included in an Advanced UI or Suite/Advanced UI installation as an executable package.

The **SuiteFormatString** function resolves formatted expressions that contain property names, environment variable references, and other special strings. At run time, the Advanced UI or Suite/Advanced UI installation expands the values of these expressions. The function stores the resulting string in a string variable.

## Syntax

```
SuiteFormatString (string szValue, string svFormattedValue);
```

## Parameters

**Table 287** • SuiteFormatString Parameters

Parameter	Description
<b>szValue</b>	Specify a string that includes one or more formatted expressions that contain property names, environment variable references, or other special strings. To learn about the syntax that is available for these expressions, see Using Formatted Expressions that Advanced UI and Suite/Advanced UI Installations Resolve at Run Time.
<b>svFormattedValue</b>	Specify the string variable that you want to contain the formatted string or expression. At run time, the installation expands the values of valid formatted expressions. Invalid expressions resolve to empty strings ("").

## Return Values

**Table 288** • SuiteFormatString Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	The function successfully resolved the formatted expression.
<b>ISERR_ISERVICE_NOT_ENABLED</b>	The function was not called in an InstallScript package that was launched from an Advanced UI or Suite/Advanced UI installation, or from an InstallScript action in a Suite/Advanced UI installation. It was called in an InstallScript installation that was launched directly, or from an InstallScript installation that was included as an executable package in an Advanced UI or Suite/Advanced UI installation.

## SuiteFormatString Example

```
//-----
//
// InstallScript Example Script
//
// Demonstrates the following functions:
//   *SuiteSetProperty
//   *SuiteGetProperty
//   *SuiteResolveString
//   *SuiteFormatString
//   *SuiteLogInfo
//
// This sample can be called in an InstallScript package that is launched
// from an Advanced UI or Suite/Advanced UI installation. This code logs
// the following line in the Advanced UI or Suite/Advanced UI debug log:
// SetGet: MyPropertyValue; Resolved: English (United States); Formatted: fmt MyPropertyValue;
//
//-----

function OnBegin()
```

```

STRING szValue;
STRING szResolved;
STRING szFormatted;

begin

    if SUITE_HOSTED then
        SuiteSetProperty ("MyPropertyName", "MyPropertyValue");
        SuiteGetProperty ("MyPropertyName", szValue);
        SuiteResolveString ("IDS_LANGUAGE_1033", szResolved);
        SuiteFormatString ("fmt [MyPropertyName]", szFormatted);
        SuiteLogInfo ("SetGet: %s; Resolved: %s; Formatted: %s;",
            szValue, szResolved, szFormatted);
    endif;

end;

```

## SuiteGetProperty



**Project** • This information applies to the following project types:

- InstallScript
- Suite/Advanced UI



**Note** • This function is available for InstallScript installations that may be included as InstallScript packages in an Advanced UI or Suite/Advanced UI installation. For more information, see [Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project](#).

This function is also available for InstallScript actions that are included in Suite/Advanced UI installations. For more information, see [Working with an Action that Runs InstallScript Code in a Suite/Advanced UI Installation](#).

This function returns an error in the following scenarios:

- The function is called in an InstallScript installation that is launched directly (that is, not from an Advanced UI or Suite/Advanced UI installation).
- The function is called in an InstallScript installation that is included in an Advanced UI or Suite/Advanced UI installation as an executable package.

The **SuiteGetProperty** function retrieves the value of an Advanced UI or Suite/Advanced UI property from the Advanced UI or Suite/Advanced UI installation that is running the InstallScript package, or from the Suite/Advanced UI installation that is running the InstallScript action. The function stores the property value in a string variable.

### Syntax

```
SuiteGetProperty (string szName, string svValue);
```



## Parameters

**Table 289** • SuiteGetProperty Parameters

Parameter	Description
<b>szName</b>	Specify the name of the Advanced UI or Suite/Advanced UI property whose value you want to retrieve.
<b>svValue</b>	Specify the string variable that you want to contain the value of the Advanced UI or Suite/Advanced UI property.

## Return Values

**Table 290** • SuiteGetProperty Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	The function successfully retrieved the property value.
<b>ISERR_ISERVICE_NOT_ENABLED</b>	The function was not called in an InstallScript package that was launched from an Advanced UI or Suite/Advanced UI installation, or from an InstallScript action in a Suite/Advanced UI installation. It was called in an InstallScript installation that was launched directly, or from an InstallScript installation that was included as an executable package in an Advanced UI or Suite/Advanced UI installation.

## SuiteGetProperty Example

```
//-----
//
//  InstallScript Example Script
//
//  Demonstrates the following functions:
//      *SuiteSetProperty
//      *SuiteGetProperty
//      *SuiteResolveString
//      *SuiteFormatString
//      *SuiteLogInfo
//
//  This sample can be called in an InstallScript package that is launched
//  from an Advanced UI or Suite/Advanced UI installation. This code logs
//  the following line in the Advanced UI or Suite/Advanced UI debug log:
//  SetGet: MyPropertyValue; Resolved: English (United States); Formatted: fmt MyPropertyValue;
//
//-----

function OnBegin()

    STRING szValue;
    STRING szResolved;
    STRING szFormatted;
```

```

begin

    if SUITE_HOSTED then
        SuiteSetProperty ("MyPropertyName", "MyPropertyValue");
        SuiteGetProperty ("MyPropertyName", szValue);
        SuiteResolveString ("IDS_LANGUAGE_1033", szResolved);
        SuiteFormatString ("fmt [MyPropertyName]", szFormatted);
        SuiteLogInfo ("SetGet: %s; Resolved: %s; Formatted: %s;",
            szValue, szResolved, szFormatted);
    endif;

end;

```

## SuiteLogInfo



**Project** • This information applies to the following project types:

- *InstallScript*
- *Suite/Advanced UI*



**Note** • This function is available for InstallScript installations that may be included as InstallScript packages in an Advanced UI or Suite/Advanced UI installation. For more information, see *Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project*.

This function is also available for InstallScript actions that are included in Suite/Advanced UI installations. For more information, see *Working with an Action that Runs InstallScript Code in a Suite/Advanced UI Installation*.

This function returns an error in the following scenarios:

- The function is called in an InstallScript installation that is launched directly (that is, not from an Advanced UI or Suite/Advanced UI installation).
- The function is called in an InstallScript installation that is included in an Advanced UI or Suite/Advanced UI installation as an executable package.

The **SuiteLogInfo** function logs information about the InstallScript package that is running in an Advanced UI or Suite/Advanced UI installation to the Advanced UI or Suite/Advanced UI debug log. It also logs information about the InstallScript action that is running in a Suite/Advanced UI installation to the Suite/Advanced UI debug log.

For information about generating an Advanced UI or Suite/Advanced UI debug log, see *Troubleshooting Issues with an Advanced UI or Suite/Advanced UI Installation*.

### Syntax

```
SuiteLogInfo (string szFormat, arglist);
```

## Parameters

**Table 291** • SuiteLogInfo Parameters

Parameter	Description
<b>szFormat</b>	Specify a standard printf-style format string to be rendered and written to the Advanced UI or Suite/Advanced UI installation's debug log.
<b>arglist</b>	Specify one or more arguments that correspond with the same number of format specifiers in szFormat.

## Return Values

**Table 292** • SuiteLogInfo Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	The function successfully logged information.
<b>ISERR_ISERVICE_NOT_ENABLED</b>	The function was not called in an InstallScript package that was launched from an Advanced UI or Suite/Advanced UI installation, or from an InstallScript action in a Suite/Advanced UI installation. It was called in an InstallScript installation that was launched directly, or from an InstallScript installation that was included as an executable package in an Advanced UI or Suite/Advanced UI installation.

## SuiteLogInfo Example

```
//-----
//
//  InstallScript Example Script
//
//  Demonstrates the following functions:
//      *SuiteSetProperty
//      *SuiteGetProperty
//      *SuiteResolveString
//      *SuiteFormatString
//      *SuiteLogInfo
//
//  This sample can be called in an InstallScript package that is launched
//  from an Advanced UI or Suite/Advanced UI installation. This code logs
//  the following line in the Advanced UI or Suite/Advanced UI debug log:
//  SetGet: MyPropertyValue; Resolved: English (United States); Formatted: fmt MyPropertyValue;
//
//-----

function OnBegin()

STRING szValue;
STRING szResolved;
STRING szFormatted;
```

```

begin

    if SUITE_HOSTED then
        SuiteSetProperty ("MyPropertyName", "MyPropertyValue");
        SuiteGetProperty ("MyPropertyName", szValue);
        SuiteResolveString ("IDS_LANGUAGE_1033", szResolved);
        SuiteFormatString ("fmt [MyPropertyName]", szFormatted);
        SuiteLogInfo ("SetGet: %s; Resolved: %s; Formatted: %s;",
            szValue, szResolved, szFormatted);
    endif;

end;

```

## SuiteReportError



**Project** • This information applies to the following project types:

- InstallScript
- Suite/Advanced UI



**Note** • This function is available for InstallScript installations that may be included as InstallScript packages in an Advanced UI or Suite/Advanced UI installation. For more information, see [Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project](#).

This function returns an error in the following scenarios:

- The function is called in an InstallScript installation that is launched directly (that is, not from an Advanced UI or Suite/Advanced UI installation).
- The function is called in an InstallScript installation that is included in an Advanced UI or Suite/Advanced UI installation as an executable package.

The **SuiteReportError** function displays a message box in the Advanced UI or Suite/Advanced UI user interface to report an error that occurred as the Advanced UI or Suite/Advanced UI installation ran the InstallScript package.

### Syntax

```
SuiteReportError (string szMessage, nFlags);
```

## Parameters

**Table 293** • SuiteReportError Parameters

Parameter	Description
<b>szMessage</b>	Specify the error message that you want to display for end users.
<b>nFlags</b>	Specify any valid MessageBox flags. For valid flags, see <a href="#">MessageBox Function</a> on MSDN.

## Return Values

**Table 294** • SuiteReportError Return Values

Return Value	Description
<b>ISERR_ISERVICE_NOT_ENABLED</b>	The function was not called in an InstallScript package that was launched from an Advanced UI or Suite/Advanced UI installation. It was called in an InstallScript installation that was launched directly, or from an InstallScript installation that was included as an executable package in an Advanced UI or Suite/Advanced UI installation.
<b>MessageBox Return Values</b>	For possible return values, see <a href="#">MessageBox Function</a> on MSDN.

# SuiteResolveString



**Project** • This information applies to the following project types:

- *InstallScript*
- *Suite/Advanced UI*



**Note** • This function is available for InstallScript installations that may be included as InstallScript packages in an Advanced UI or Suite/Advanced UI installation. For more information, see *Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project*.

This function is also available for InstallScript actions that are included in Suite/Advanced UI installations. For more information, see *Working with an Action that Runs InstallScript Code in a Suite/Advanced UI Installation*.

This function returns an error in the following scenarios:

- The function is called in an InstallScript installation that is launched directly (that is, not from an Advanced UI or Suite/Advanced UI installation).
- The function is called in an InstallScript installation that is included in an Advanced UI or Suite/Advanced UI installation as an executable package.

The **SuiteResolveString** function replaces an Advanced UI or Suite/Advanced UI string identifier with its corresponding string value in the InstallScript package that is running in an Advanced UI or Suite/Advanced UI installation, or in the InstallScript action that is running in a Suite/Advanced UI installation.

## Syntax

```
SuiteResolveString (string szStringID, string svResolvedString);
```

## Parameters

**Table 295** • SuiteResolveString Parameters

Parameter	Description
<b>szStringID</b>	Specify the Advanced UI or Suite/Advanced UI string identifier that you want to be replaced.
<b>svResolvedString</b>	Specify the string variable that you want to store the value of the Advanced UI or Suite/Advanced UI string identifier; the value is based on the Advanced UI or Suite/Advanced UI installation's selected language. An unknown string identifier returns an empty string ("").

## Return Values

**Table 296** • SuiteResolveString Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	The function successfully replaced the string identifier with the corresponding string value.
<b>ISERR_ISERVICE_NOT_ENABLED</b>	The function was not called in an InstallScript package that was launched from an Advanced UI or Suite/Advanced UI installation, or from an InstallScript action in a Suite/Advanced UI installation. It was called in an InstallScript installation that was launched directly, or from an InstallScript installation that was included as an executable package in an Advanced UI or Suite/Advanced UI installation.

## SuiteResolveString Example

```
//-----
//
// InstallScript Example Script
//
// Demonstrates the following functions:
//   *SuiteSetProperty
//   *SuiteGetProperty
//   *SuiteResolveString
//   *SuiteFormatString
//   *SuiteLogInfo
//
// This sample can be called in an InstallScript package that is launched
```

```
// from an Advanced UI or Suite/Advanced UI installation. This code logs
// the following line in the Advanced UI or Suite/Advanced UI debug log:
// SetGet: MyPropertyValue; Resolved: English (United States); Formatted: fmt MyPropertyValue;
//
//-----

function OnBegin()

STRING szValue;
STRING szResolved;
STRING szFormatted;

begin

    if SUITE_HOSTED then
        SuiteSetProperty ("MyPropertyName", "MyPropertyValue");
        SuiteGetProperty ("MyPropertyName", szValue);
        SuiteResolveString ("IDS_LANGUAGE_1033", szResolved);
        SuiteFormatString ("fmt [MyPropertyName]", szFormatted);
        SuiteLogInfo ("SetGet: %s; Resolved: %s; Formatted: %s;",
            szValue, szResolved, szFormatted);
    endif;

end;
```

## SuiteSetProperty



**Project** • This information applies to the following project types:

- *InstallScript*
- *Suite/Advanced UI*



**Note** • This function is available for InstallScript installations that may be included as InstallScript packages in an Advanced UI or Suite/Advanced UI installation. For more information, see *Adding an InstallScript Package to an Advanced UI or Suite/Advanced UI Project*.

This function is also available for InstallScript actions that are included in Suite/Advanced UI installations. For more information, see *Working with an Action that Runs InstallScript Code in a Suite/Advanced UI Installation*.

This function returns an error in the following scenarios:

- The function is called in an InstallScript installation that is launched directly (that is, not from an Advanced UI or Suite/Advanced UI installation).
- The function is called in an InstallScript installation that is included in an Advanced UI or Suite/Advanced UI installation as an executable package.

The **SuiteSetProperty** function sets the value of an Advanced UI or Suite/Advanced UI property in the Advanced UI or Suite/Advanced UI installation that launched the running InstallScript package, or the value of a Suite/Advanced UI property in the Suite/Advanced UI installation that launched the running InstallScript action.

## Syntax

```
SuiteSetProperty (string szName, string szValue);
```

## Parameters

**Table 297** • SuiteSetProperty Parameters

Parameter	Description
<b>szName</b>	Specify the name of the Advanced UI or Suite/Advanced UI property whose value you want to set.
<b>szValue</b>	Specify the value of the Advanced UI or Suite/Advanced UI property that you want to set.

## Return Values

**Table 298** • SuiteSetProperty Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	The function successfully set the property value.
<b>ISERR_ISERVICE_NOT_ENABLED</b>	The function was not called in an InstallScript package that was launched from an Advanced UI or Suite/Advanced UI installation, or from an InstallScript action in a Suite/Advanced UI installation. It was called in an InstallScript installation that was launched directly, or from an InstallScript installation that was included as an executable package in an Advanced UI or Suite/Advanced UI installation.

## SuiteSetProperty Example

```
//-----
//
// InstallScript Example Script
//
// Demonstrates the following functions:
//   *SuiteSetProperty
//   *SuiteGetProperty
//   *SuiteResolveString
//   *SuiteFormatString
//   *SuiteLogInfo
//
// This sample can be called in an InstallScript package that is launched
// from an Advanced UI or Suite/Advanced UI installation. This code logs
// the following line in the Advanced UI or Suite/Advanced UI debug log:
// SetGet: MyPropertyValue; Resolved: English (United States); Formatted: fmt MyPropertyValue;
//
//-----

function OnBegin()
```



```

STRING szValue;
STRING szResolved;
STRING szFormatted;

begin

    if SUITE_HOSTED then
        SuiteSetProperty ("MyPropertyName", "MyPropertyValue");
        SuiteGetProperty ("MyPropertyName", szValue);
        SuiteResolveString ("IDS_LANGUAGE_1033", szResolved);
        SuiteFormatString ("fmt [MyPropertyName]", szFormatted);
        SuiteLogInfo ("SetGet: %s; Resolved: %s; Formatted: %s;",
            szValue, szResolved, szFormatted);
    endif;

end;

```

## System

The **System** function is documented for backward compatibility with InstallShield Professional. With newer versions of InstallShield, RebootDialog and SdFinishReboot are better functions to use to restart Windows or reboot the system. Of these two, SdFinishReboot provides the most functionality. Refer to the individual function descriptions to see which one best meets your needs.

Use the System function to restart Windows or reboot the system after an installation has completed. System does not perform an aborted setup—that is, it does not remove installed files. However, InstallShield does remove any temporary directories and temporary files it placed on the system to carry out the setup. To reboot the system immediately after calling the System function, use the exit keyword immediately after calling the System function.

Some systems may not restart or may hang when this function is called. Many setup routines (including installation for system software such as MS-DOS) display a warning message to the user before they restart the system. This warning message indicates what is happening and instructs the user to reboot the system manually if the command fails.




---

**Note** • This function calls the Windows API *ExitWindows*. Due to the wide variety of BIOS types, this function is highly dependent on the BIOS interaction with the system.

### Syntax

```
System ( nOp );
```

## Parameters

**Table 299** • System Parameters

Parameter	Description
nOp	Specifies which action to perform after terminating the setup. Pass the following predefined constant in this parameter: <ul style="list-style-type: none"> <li><b>SYS_BOOTMACHINE</b>—Reboots the system.</li> </ul>

## System Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the System function.
*
* This script reboots the computer.
*
\*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_System(HWND);

function ExFn_System(hMSI)
begin

    System (SYS_BOOTMACHINE);

end;
```

## TextSubGetValue

The **TextSubGetValue** function retrieves in svValue the text substitution string that is associated with szTextSub. The bGlobalOnly parameter specifies whether the function searches for local text substitutions.

### Syntax

```
TextSubGetValue ( szTextSub, svValue, bGlobalOnly, bResolveEmbedded );
```

## Parameters

**Table 300** • TextSubGetValue Parameters

Parameter	Description
<b>szTextSub</b>	Specifies the string that is replaced by svValue when text substitution is performed. You can optionally enclose the string value in angle brackets, for example, "<MYTEXTSUB>". If you enclose the string value in angle brackets, you can include additional text in the string outside the angle brackets (for example, "additional<MYTEXTSUB>text") that is automatically removed before text substitution is performed.
<b>svValue</b>	Returns the string that replaces occurrences of szTextSub when text substitution is performed. This string value can itself contain embedded text substitutions. For more information, see <a href="#">Text Substitutions</a> .
<b>bGlobalOnly</b>	Specifies whether TextSubGetValue searches among only global text substitutions (TRUE) or both global and local text substitutions (FALSE). For more information, see <a href="#">Text Substitutions</a> .
<b>bResolveEmbedded</b>	Specifies whether to perform any embedded text substitutions. For more information, see <a href="#">Text Substitutions</a> . See the following example code and comment lines: <pre>TextSubSetValue ( "&lt;MYTEXTSUB1&gt;", "First Text Sub", FALSE ); TextSubSetValue ( "&lt;MYTEXTSUB2&gt;", "Second Text Sub &lt;MYTEXTSUB1&gt;", FALSE ); TextSubGetValue ( "&lt;MYTEXTSUB2&gt;", svValue, FALSE, TRUE ); // svValue is "Second Text Sub First Text Sub" TextSubGetValue ( "&lt;MYTEXTSUB2&gt;", svValue, FALSE, FALSE ); // svValue is "Second Text Sub &lt;MYTEXTSUB1&gt;"</pre>

## Return Values

**Table 301** • TextSubGetValue Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	TextSubGetValue successfully retrieved the associated string.
<b>&lt; ISERR_SUCCESS</b>	TextSubGetValue failed to retrieve the associated string.

## TextSubGetValue Example

```
/*-----*\
```

```

*
* InstallShield Example Script
*
* Demonstrates the TextSub* functions: TextSubSetValue,
* TextSubSubstitute, TextSubGetValue, and TextSubParseTextSub.
*
\*-----*/

function OnBegin( )
    string svString, svValue;
begin
    TextSubSetValue ( "<MYTEXTSUB1>", "First Text Sub", FALSE );
    svString = "Text <MYTEXTSUB1> String";
    TextSubSubstitute ( svString, FALSE );
    MessageBox( svString, INFORMATION );
    // svString is "Text First Text Sub String"

    TextSubSetValue ( "<MYTEXTSUB2>", "Second Text Sub <MYTEXTSUB1>", FALSE );
    TextSubGetValue ( "<MYTEXTSUB2>", svValue, FALSE, TRUE );
    MessageBox( svValue, INFORMATION );
    // svValue is "Second Text Sub First Text Sub"
    TextSubGetValue ( "<MYTEXTSUB2>", svValue, FALSE, FALSE );
    MessageBox( svValue, INFORMATION );
    // svValue is "Second Text Sub <MYTEXTSUB1>"

    svString = "Text <MYTEXTSUB1> String";
    TextSubParseTextSub ( svString );
    MessageBox( svString, INFORMATION );
    // svString is "MYTEXTSUB1"
end;

```

## TextSubParseTextSub

The **TextSubParseTextSub** function searches svTextSub for a substring that is enclosed in angle brackets (which is a standard way of denoting a text substitution string). If TextSubParseTextSub finds such a substring, it returns the first such substring, without the enclosing angle brackets, in svTextSub; otherwise, TextSubParseTextSub does not change svTextSub.

### Syntax

```
TextSubParseTextSub ( svTextSub );
```

## Parameters

**Table 302** • TextSubParseTextSub Parameters

Parameter	Description
<b>svTextSub</b>	Specifies the string that is to be searched.

## Return Values

**Table 303** • TextSubParseTextSub Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	This function always returns ISERR_SUCCESS.

## TextSubParseTextSub Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the TextSub* functions: TextSubSetValue,
 * TextSubSubstitute, TextSubGetValue, and TextSubParseTextSub.
 *
 \*-----*/

function OnBegin( )
    string svString, svValue;
begin
    TextSubSetValue ( "<MYTEXTSUB1>", "First Text Sub", FALSE );
    svString = "Text <MYTEXTSUB1> String";
    TextSubSubstitute ( svString, FALSE );
    MessageBox( svString, INFORMATION );
    // svString is "Text First Text Sub String"

    TextSubSetValue ( "<MYTEXTSUB2>", "Second Text Sub <MYTEXTSUB1>", FALSE );
    TextSubGetValue ( "<MYTEXTSUB2>", svValue, FALSE, TRUE );
    MessageBox( svValue, INFORMATION );
    // svValue is "Second Text Sub First Text Sub"
    TextSubGetValue ( "<MYTEXTSUB2>", svValue, FALSE, FALSE );
    MessageBox( svValue, INFORMATION );
    // svValue is "Second Text Sub <MYTEXTSUB1>"

    svString = "Text <MYTEXTSUB1> String";
    TextSubParseTextSub ( svString );
    MessageBox( svString, INFORMATION );
    // svString is "MYTEXTSUB1"
end;

```

# TextSubSetValue


The **TextSubSetValue** function creates a text substitution association between szTextSub and szValue. The bGlobal parameter specifies whether the association is global or local.

## Syntax

```
TextSubSetValue ( szTextSub, szValue, bGlobal );
```

## Parameters

**Table 304** • TextSubSetValue Parameters

Parameter	Description
<b>szTextSub</b>	Specifies the string that is replaced by szValue when text substitution is performed. You can optionally enclose the string value in angle brackets, for example, "<MYTEXTSUB>". If you enclose the string value in angle brackets, you can include additional text in the string outside the angle brackets (for example, "additional<MYTEXTSUB>text") that is automatically removed before text substitution is performed.
	 <p><b>Note</b> • In order to avoid conflicts with existing text substitutions that are created internally or by objects used by the installation, it is recommended that any values of szTextSub that you specify in TextSubSetValue use a prefix that is specific to the installation.</p>
<b>szValue</b>	Specifies the string that replaces occurrences of szTextSub when text substitution is performed. This string value can itself contain embedded text substitutions. For more information, see <a href="#">Text Substitutions</a> .
<b>bGlobal</b>	Specifies whether the association between szTextSub and szValue is global or local; for more information, see <a href="#">Text Substitutions</a> .

## Return Values

**Table 305** • TextSubSetValue Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	TextSubSetValue successfully associated the specified strings.
<b>&lt; ISERR_SUCCESS</b>	TextSubSetValue failed to associate the specified strings.

## TextSubSetValue Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the TextSub* functions: TextSubSetValue,
 * TextSubSubstitute, TextSubGetValue, and TextSubParseTextSub.
 *
 \*-----*/

function OnBegin( )

```

```

    string svString, svValue;
begin
    TextSubSetValue ( "<MYTEXTSUB1>", "First Text Sub", FALSE );
    svString = "Text <MYTEXTSUB1> String";
    TextSubSubstitute ( svString, FALSE );
    MessageBox( svString, INFORMATION );
    // svString is "Text First Text Sub String"

    TextSubSetValue ( "<MYTEXTSUB2>", "Second Text Sub <MYTEXTSUB1>", FALSE );
    TextSubGetValue ( "<MYTEXTSUB2>", svValue, FALSE, TRUE );
    MessageBox( svValue, INFORMATION );
    // svValue is "Second Text Sub First Text Sub"
    TextSubGetValue ( "<MYTEXTSUB2>", svValue, FALSE, FALSE );
    MessageBox( svValue, INFORMATION );
    // svValue is "Second Text Sub <MYTEXTSUB1>"

    svString = "Text <MYTEXTSUB1> String";
    TextSubParseTextSub ( svString );
    MessageBox( svString, INFORMATION );
    // svString is "MYTEXTSUB1"
end;

```

## TextSubSubstitute

The **TextSubSubstitute** function performs text substitution on svString. The bGlobalOnly parameter specifies whether the function performs local text substitutions.

### Syntax

```
TextSubSubstitute ( svString, bGlobalOnly );
```



## Parameters

**Table 306** • TextSubSubstitute Parameters

Parameter	Description
<b>svString</b>	Specifies the string on which text substitution is performed.
<b>bGlobalOnly</b>	Specifies whether TextSubSubstitute performs only global text substitutions (TRUE) or both global and local text substitutions (FALSE); for more information, see <a href="#">Text Substitutions</a> .

## Return Values

**Table 307** • TextSubSubstitute Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	TextSubSubstitute successfully performed text substitution.
<b>&lt; ISERR_SUCCESS</b>	TextSubSubstitute failed to perform text substitution.

## TextSubSubstitute Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the TextSub* functions: TextSubSetValue,
 * TextSubSubstitute, TextSubGetValue, and TextSubParseTextSub.
 *
 \*-----*/

function OnBegin( )
    string svString, svValue;
begin
    TextSubSetValue ( "<MYTEXTSUB1>", "First Text Sub", FALSE );
    svString = "Text <MYTEXTSUB1> String";
    TextSubSubstitute ( svString, FALSE );
    MessageBox( svString, INFORMATION );
    // svString is "Text First Text Sub String"

    TextSubSetValue ( "<MYTEXTSUB2>", "Second Text Sub <MYTEXTSUB1>", FALSE );
    TextSubGetValue ( "<MYTEXTSUB2>", svValue, FALSE, TRUE );
    MessageBox( svValue, INFORMATION );
    // svValue is "Second Text Sub First Text Sub"
    TextSubGetValue ( "<MYTEXTSUB2>", svValue, FALSE, FALSE );
    MessageBox( svValue, INFORMATION );
    // svValue is "Second Text Sub <MYTEXTSUB1>"

```

```
svString = "Text <MYTEXTSUB1> String";  
TextSubParseTextSub ( svString );  
MessageBox( svString, INFORMATION );  
// svString is "MYTEXTSUB1"  
end;
```

# Built-In Functions (U-Z)

For a list of functions by category, see [Built-In Functions by Category](#).

## UninstallApplication



---

**Project** • *This information applies to InstallScript projects.*

The **UninstallApplication** function launches the uninstallation that is specified by szUninstallKey.

### Syntax

```
UninstallApplication ( szUninstallKey, szAdditionalCmdLine, nOptions );
```

## Parameters

**Table 1 • UninstallApplication Parameters**

Parameter	Description
<b>szUninstallKey</b>	Specifies the name of a subkey under the target registry's <root key>\Software\Microsoft\Windows\CurrentVersion\Uninstall key; the function first checks for the subkey under the root key HKEY_CURRENT_USER, and if it does not find the subkey there, it checks under HKEY_LOCAL_MACHINE. For setups created with InstallShield Professional 5.53 or earlier, this is typically the name of the application; for setups created with InstallShield Professional 6.0 or later, this is the application's product GUID including the surrounding braces ({}). Do not enter the product GUID of the current setup.
<b>szAdditionalCmdLine</b>	Specifies any additional command line arguments that you want to pass to the uninstallation. You do not need to specify command line arguments that are already specified in szUninstallKey's UninstallString value's data.
<b>nOptions</b>	Specifies additional options. You can specify any option that is supported by <a href="#">LaunchApplication</a> .

## Return Values

**Table 2 • UninstallApplication Return Values**

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the function successfully launched the uninstallation.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function was unable to launch the uninstallation.  You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a> .

# UnUseDLL

The **UnUseDLL** function unloads a .dll file from memory. **UnUseDLL** decrements the lock count of the .dll file by one. When the lock count equals zero, the InstallScript engine unloads the .dll file. Every call to **UseDLL** should have a matching call to **UnUseDLL** so that .dll files are not left in memory after they are no longer needed, using system resources. After you unload a .dll file, you cannot call the functions in that .dll file.

If the script exits or terminates before properly unloading the .dll file with **UnUseDLL**, the .dll file is locked in memory. If you attempt to access the .dll file again, your script might fail. You must remove the .dll file from memory by restarting Windows.



**Caution** • Microsoft Windows system .dll files, such as **User32.dll**, **Gdi32.dll**, and **Kernel32.dll**, are loaded and unloaded automatically by Windows. Do not call **UseDLL** and **UnUseDLL** to load and unload these .dll files.

## Syntax

```
UnUseDLL ( szDLLName );
```

## Parameters

**Table 3** • UnUseDLL Parameters

Parameter	Description
<b>szDLLName</b>	Specifies the file name of the .dll file. Do not include the path in this parameter.

## Return Values

**Table 4** • UnUseDLL Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully unlocked and possibly unloaded the .dll file from memory.
<b>&lt; 0</b>	Indicates that the function was unable to unlock or unload the .dll file.

## UnUseDLL Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
```

```

* Demonstrates the UseDLL and the UnUseDLL functions.
*
* UseDLL is called to load an example .dll file into memory. A
* function in this .dll file is then called to demonstrate how
* .dll functions can be used in an installation. Finally,
* UnUseDLL is called to unload the example .dll file from memory.
*
* Note: This script requires that the constant DLL_FILE be set
*       to the fully qualified name of a .dll file that contains
*       a function called TheExportedFunction. The format of that
*       function must match the prototype declaration below.
*
\*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

#define DLL_FILE SUPPORTDIR ^ "TheDLL.dll"

// Prototype TheExportedFunction in TheDLL.dll.
prototype TheDLL.TheExportedFunction (INT, WPOINTER);

export prototype ExFn_UnUseDLL(HWND);

function ExFn_UnUseDLL(hMSI)
    STRING    svString;
    INT       nValue;
    WPOINTER  psvString;
    NUMBER    nResult;
    BOOL      bDone;
begin

    // Load the .dll file into memory.
    nResult = UseDLL (DLL_FILE);

    if (nResult = 0) then
        MessageBox ("UseDLL successful \n\n.dll file loaded.", INFORMATION);
    else
        MessageBox ("UseDLL failed.\n\nCouldn't load .dll file.", INFORMATION);
        abort;
    endif;

    // bDone controls the following while loop.
    bDone = FALSE;

    // Loop while bDone is FALSE.
    while (bDone = FALSE)
        // Disable the Back button in installation dialogs.
        Disable (BACKBUTTON);

        // Get a string from the user.
        AskText ("Enter an example string.", "Example string.", svString);

        // Get a pointer to the string to pass to TheExportedFunction.
        psvString = &svString;

        // Get the string length to pass to TheExportedFunction.
        nValue = StrLength (svString);

```

```

// Call TheExportedFunction.
TheExportedFunction (nValue, psvString);

// Display the string to observe how it was altered by TheExportedFunction.
sprintfBox (INFORMATION, "UseDLL", "TheExportedFunction() changed the string " +
    "to: %s", svString);

// Give the user a chance to do another example.
if (AskYesNo ("Do another example?", YES) = NO) then
    bDone = TRUE;
endif;
endwhile;

// Remove the .dll file from memory.
if (UnUseDLL (DLL_FILE) < 0) then
    MessageBox ("UnUseDLL failed.\n\n.dll file still in memory.", SEVERE);
else
    MessageBox ("UnUseDLL successful.\n\n.dll file removed from memory.",
        INFORMATION);
endif;

end;

```

## UpdateServiceCheckForUpdates



**Project** • This information applies to InstallScript projects.

This function is obsolete. If this function is called, it returns ISERR\_NOT\_IMPLEMENTED.

For information on adding FlexNet Connect support to an InstallScript project, consult the [Knowledge Base](#).

### Syntax

```
UpdateServiceCheckForUpdates ( szProductCode, bWait );
```

## UpdateServiceCreateShortcut



**Project** • This information applies to InstallScript projects.

This function is obsolete. If this function is called, it returns ISERR\_NOT\_IMPLEMENTED.

For information on adding FlexNet Connect support to an InstallScript project, consult the [Knowledge Base](#).

### Syntax

```
UpdateServiceCreateShortcut ( szProductCode, szFolder, szItemName );
```

# UpdateServiceEnableUpdateManagerInstall



**Project** • This information applies to InstallScript projects.

This function is obsolete. If this function is called, it returns ISERR\_NOT\_IMPLEMENTED.

For information on adding FlexNet Connect support to an InstallScript project, consult the [Knowledge Base](#).

## Syntax

```
UpdateServiceEnableUpdateManagerInstall (BOOL bEnable);
```

# UpdateServiceGetAgentTarget



**Project** • This information applies to InstallScript projects.

This function is obsolete. If this function is called, it returns a null string ("").

For information on adding FlexNet Connect support to an InstallScript project, consult the [Knowledge Base](#).

## Syntax

```
UpdateServiceGetAgentTarget ( );
```

# UpdateServiceOnEnabledStateChange



**Project** • This information applies to InstallScript projects.

This function is obsolete. If this function is called, it returns ISERR\_NOT\_IMPLEMENTED.

For information on adding FlexNet Connect support to an InstallScript project, consult the [Knowledge Base](#).

## Syntax

```
UpdateServiceOnEnabledStateChange ( );
```

# UpdateServiceRegisterProduct



**Project** • This information applies to InstallScript projects.

This function is obsolete. If this function is called, it returns ISERR\_NOT\_IMPLEMENTED.

For information on adding FlexNet Connect support to an InstallScript project, consult the [Knowledge Base](#).



## Syntax

```
UpdateServiceRegisterProduct ( szProductCode, szProductVersion, bRegister, nInterval );
```

# UpdateServiceRegisterProductEx



**Project** • This information applies to InstallScript projects.

This function is obsolete. If this function is called, it returns ISERR\_NOT\_IMPLEMENTED.

For information on adding FlexNet Connect support to an InstallScript project, consult the [Knowledge Base](#).

## Syntax

```
UpdateServiceRegisterProductEx (BOOL bRegister);
```

# UpdateServiceSetHost



**Project** • This information applies to InstallScript projects.

This function is obsolete. If this function is called, it returns ISERR\_NOT\_IMPLEMENTED.

For information on adding FlexNet Connect support to an InstallScript project, consult the [Knowledge Base](#).

## Syntax

```
UpdateServiceSetHost ( szProductCode, szHostURL );
```

# UpdateServiceSetLanguage



**Project** • This information applies to InstallScript projects.

This function is obsolete. If this function is called, it returns ISERR\_NOT\_IMPLEMENTED.

For information on adding FlexNet Connect support to an InstallScript project, consult the [Knowledge Base](#).

## Syntax

```
UpdateServiceSetLanguage ( szProductCode, nLanguageID );
```

# UseDLL

The **UseDLL** function loads a .dll file into memory. After the .dll file has been loaded into memory, your installation script can call a function from that .dll file. Note that if the .dll file that is specified by szDLLName requires other .dll files, those other .dll files must reside in the folder from which the .dll file attempts to load them. Normally this will

be the current folder. To ensure that those .dll files can be located, call **ChangeDirectory** to change the current folder to the location of those .dll files before calling **UseDLL**. Failure to do so may prevent the .dll file from loading properly.

Each time that you load a .dll file into memory, the .dll file's lock count is incremented. The lock count identifies the number of applications that are using the .dll file. You should call **UnUseDLL** to unload a .dll file as soon as you are done using it. If you do not unload a .dll file when you are done with it, the .dll file will remain in memory when no applications need it, thereby wasting system resources. Every call to **UseDLL** should have a matching call to **UnUseDLL** in the script.



---


**Caution** • Microsoft Windows system .dll files, such as **User32.dll**, **Gdi32.dll**, and **Kernel32.dll**, are loaded and unloaded automatically by Windows. Do not call **UseDLL** and **UnUseDLL** to load and unload these .dll files.

## Syntax

```
UseDLL ( szDLLName );
```

## Parameters

**Table 5 • UseDLL Parameters**

Parameter	Description
<b>szDLLName</b>	<p>Specifies the name of the .dll file to be loaded. If you do not specify an extension, the InstallScript engine assumes that the file has the extension .dll or .exe. Including a path in this parameter is recommended but optional. If the path for the .dll file is not specified in this parameter, the InstallScript engine searches for the .dll file using the same search order that the Windows API function <b>LoadLibrary</b> uses. For more information regarding the search order, see the description of this Windows API function on MSDN.</p> <p>To include the .dll file in your installation, add it to the appropriate subfolder of the Language Independent folder in the Support Files view. When the installation is running on a target system, the .dll file is available in the temporary directory that is specified by SUPPORTDIR. You can then append the .dll file name to SUPPORTDIR as follows in order to reference the .dll file:</p> <pre>szDLLName = SUPPORTDIR ^ "TheDLL.dll";    UseDLL (szDLLName);</pre> <p>If you do not place your .dll file into your installation (by inserting it into the appropriate folder in the Support Files view), you can instead distribute the file along with the files of your application and then load it from the target system. However, if you do so, you must specify the location to which you install the .dll file so that your installation can reference it. You must also ensure that your installation does not attempt to load the .dll file before it has been transferred to the target system.</p> <p> <b>Note •</b> The <i>szDLLName</i> parameter does not support uniform resource locators (URLs).</p>

## Return Values

**Table 6 • UseDLL Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully loaded the .dll file into memory.
<b>&lt; 0</b>	<p>Indicates that the function was unable to load the .dll file into memory.</p> <p>If <b>UseDLL</b> fails, the most likely cause is that the .dll file was not found. If this happens, make sure that the correct path is specified in the parameter <i>szDLLName</i>.</p> <p>Another common cause of failure that is associated with using .dll files is related to .dll file dependencies—.dll files that are accessed by the .dll file that you load. If the .dll files that your .dll file must access are not loaded or found, your .dll file call may fail. If this occurs, make sure that the other .dll files are on the system and that they are accessible.</p> <p>If the script exits or terminates before properly unloading the .dll file with <b>UnUseDLL</b>, the .dll file will be locked in memory. If you attempt to access the .dll file again, your script may fail. You must remove the .dll file from memory by restarting Windows.</p>

## UseDLL Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the UseDLL and the UnUseDLL functions.
 *
 * UseDLL is called to load an example .dll file into memory. A
 * function in this .dll file is then called to demonstrate how
 * .dll functions can be used in an installation. Finally,
 * UnUseDLL is called to unload the example .dll file from memory.
 *
 * Note: This script requires that the constant DLL_FILE be set
 *       to the fully qualified name of a .dll file that contains
 *       a function called TheExportedFunction. The format of that
 *       function must match the prototype declaration below.
 *
\*-----*/

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

#define DLL_FILE SUPPORTDIR ^ "TheDLL.dll"

// Prototype TheExportedFunction in TheDLL.dll.
prototype TheDLL.TheExportedFunction (INT, WPOINTER);

export prototype ExFn_UseDLL(HWND);

function ExFn_UseDLL(hMSI)
    STRING    svString;
    INT       nValue;
    WPOINTER  psvString;
    NUMBER    nResult;
    BOOL      bDone;
begin

    // Load the .dll file into memory.
    nResult = UseDLL (DLL_FILE);

    if (nResult = 0) then
        MessageBox ("UseDLL successful \n\n.dll file loaded.", INFORMATION);
    else
        MessageBox ("UseDLL failed.\n\nCouldn't load .dll file.", INFORMATION);
        abort;
    endif;

    // bDone controls the following while loop.
    bDone = FALSE;

    // Loop while bDone is FALSE.
    while (bDone = FALSE)

```

```

// Disable the Back button in installation dialogs.
Disable (BACKBUTTON);

// Get a string from the user.
AskText ("Enter an example string.", "Example string.", svString);

// Get a pointer to the string to pass to TheExportedFunction.
psvString = &svString;

// Get the string length to pass to TheExportedFunction.
nValue = StrLength (svString);

// Call TheExportedFunction.
TheExportedFunction (nValue, psvString);

// Display the string to observe how it was altered by TheExportedFunction.
sprintfBox (INFORMATION, "UseDLL", "TheExportedFunction() changed the string " +
            "to: %s", svString);

// Give the user a chance to do another example.
if (AskYesNo ("Do another example?", YES) = NO) then
    bDone = TRUE;
endif;
endwhile;

// Remove the .dll file from memory.
if (UnUseDLL (DLL_FILE) < 0) then
    MessageBox ("UnUseDLL failed.\n\n.dll file still in memory.", SEVERE);
else
    MessageBox ("UnUseDLL successful.\n\n.dll file removed from memory.",
                INFORMATION);
endif;

end;

```

## VarInit




The **VarInit** function initializes or reinitializes internal lists used by the [VarSave](#) and [VarRestore](#) functions. Calling this function effectively clears any information stored by previous **VarSave** calls and not yet used by subsequent **VarRestore** functions.

### Syntax

```
VarInit (nType);
```

## Parameters

**Table 7 •** VarInit Parameters

Parameter	Description
<b>nType</b>	<p>Specifies which information to reset. Pass one of the following predefined constants in this parameter.</p> <p></p> <p><b>Project</b> • These constants apply to InstallScript projects:</p> <ul style="list-style-type: none"> <li>• <b>VAR_LOGGING</b>—Resets any stored uninstall logging entries.</li> <li>• <b>VAR_CURRENTDIR</b>—Resets any stored current directory entries.</li> <li>• <b>VAR_ALLSUPPORTED</b>—Resets all stored entries.</li> <li>• <b>VAR_HKEYCURRENTROOTKEY</b>—Resets any stored current root key entries.</li> <li>• <b>CURRENTROOTKEY</b>—This constant is obsolete. Use VAR_HKEYCURRENTROOTKEY instead.</li> </ul> <p></p> <p><b>Project</b> • This constant applies to Basic MSI, InstallScript, and InstallScript MSI projects:</p> <ul style="list-style-type: none"> <li>• <b>VAR_SRCTARGETDIR</b>—Resets any stored current entries for TARGETDIR (in InstallScript installations), and INSTALLDIR (in Basic MSI and InstallScript MSI installations), and SRCDIR.</li> </ul> <p></p> <p><b>Project</b> • This constant applies to Basic MSI and InstallScript MSI projects:</p> <ul style="list-style-type: none"> <li>• <b>SRCINSTALLDIR</b>—This constant is obsolete. Use VAR_SRCTARGETDIR instead.</li> </ul>

## Return Values

**Table 8 •** VarInit Return Values

Return Value	Description
<b>ISERR_SUCCESS</b>	Indicates that the current values of the system variables were reset.

## VarRestore




The **VarRestore** function reassigns the values that were saved by an earlier call to the **VarSave** function to the system variables TARGETDIR (in InstallScript installations), INSTALLDIR (in Basic MSI and InstallScript MSI installations), SRCDIR, or HKEYCURRENTROOTKEY. Call **VarSave** whenever you need to change the value of these system variables temporarily, for example, to set source and target directories before a call to **XCOPYFile**. Afterwards, set those variables back to their previous values by calling **VarRestore**.

## Syntax

```
VarRestore ( nType );
```

## Parameters

**Table 9 •** VarRestore Parameters

Parameter	Description
<b>nType</b>	<p>Specifies which system variables to restore. Pass one or more of the following predefined constants in this parameter. Note that you can pass multiple constants separated by the OR operator (  ).</p> <p></p> <p><b>Project •</b> <i>These constants apply to InstallScript projects:</i></p> <ul style="list-style-type: none"> <li>• <b>VAR_LOGGING</b>—Resets any stored uninstallation logging entries.</li> <li>• <b>VAR_CURRENTDIR</b>—Resets any stored current directory entries.</li> <li>• <b>VAR_ALLSUPPORTED</b>—Resets all stored entries.</li> <li>• <b>VAR_HKEYCURRENTROOTKEY</b>—Resets any stored current root key entries.</li> <li>• <b>VAR_REGOPTIONS</b>—Resets any stored current registry options.</li> <li>• <b>CURRENTROOTKEY</b>—This constant is obsolete. Use VAR_HKEYCURRENTROOTKEY instead.</li> </ul> <p></p> <p><b>Project •</b> <i>This constant applies to Basic MSI, InstallScript, and InstallScript MSI projects:</i></p> <ul style="list-style-type: none"> <li>• <b>VAR_SRCTARGETDIR or SRCTARGETDIR</b>—Resets any stored current TARGETDIR (in InstallScript installations), INSTALLDIR (in Basic MSI and InstallScript MSI installations), and SRCDIR entries.</li> </ul> <p></p> <p><b>Project •</b> <i>This constant applies to Basic MSI and InstallScript MSI projects:</i></p> <ul style="list-style-type: none"> <li>• <b>SRCINSTALLDIR</b>—This constant is obsolete. Use VAR_SRCTARGETDIR instead.</li> </ul>

## Return Values

**Table 10 •** VarRestore Return Values

Return Value	Description
<b>0</b>	Indicates that the previously saved values of the system variables were restored.
<b>&lt; 0</b>	Indicates that there are no values in storage to be restored. This error occurs if you call VarRestore without calling VarSave first, or if you call VarRestore more times than you call VarSave.

## Additional Information



Each time you call **VarSave**, the InstallScript engine pushes the current values of the system variables onto an internal stack, which operates as a last in, first out storage area. This method allows you to stack a series of values in memory. You can then make calls to **VarRestore** to retrieve those values from the stack in the opposite order that you stored them.

For example, if you call **VarSave** three times with SRCTARGETDIR as its argument (without calling **VarRestore** in between those calls), there will be three sets of SRCDIR and TARGETDIR values on the stack. The first call to **VarRestore** with SRCTARGETDIR as its argument restores the values from the third call to **VarSave**. The next call to **VarRestore** restores the values from the second call to **VarSave**. The third call to **VarRestore** restores the values from the first call to **VarSave**. At that point, the stack is empty. To see a script fragment that illustrates this process, refer to [VarSave Stack Example](#).

## VarRestore Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the VarSave and VarRestore functions.
*
* This script starts by displaying the starting values of the
* system variables SRCDIR and INSTALLDIR. It then calls VarSave
* to save those values. Next, it assigns new values to SRCDIR
* and INSTALLDIR, and it displays those values. Finally, it
* calls VarRestore to restore the original values, which are
* then displayed.
*
\*-----*/

#define NEW_SOURCE_DIR "C:\\Source"
#define NEW_INSTALL_DIR "C:\\Target"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_VarRestore(HWND);

function ExFn_VarRestore(hMSI)
begin
    // Display the values of SRCDIR and INSTALLDIR.
    sprintfBox (INFORMATION, "Starting Source and Target Folders",
        "Source:\n\n%s\n\n Target:\n\n%s ",
        SRCDIR, INSTALLDIR);

    // Save the current values of SRCDIR and INSTALLDIR.
    VarSave (SRCTARGETDIR);

    // Assign new values to SRCDIR and INSTALLDIR.
    SRCDIR = NEW_SOURCE_DIR;
    INSTALLDIR = NEW_INSTALL_DIR;
```

```

    // Display the values of SRCDIR and INSTALLDIR.
    sprintfBox (INFORMATION, "New Source and Target Folders",
               "New Source:\n\n%s\n\n New Target:\n\n%s",
               SRCDIR, INSTALLDIR);

    // Restore the old values.
    VarRestore (SRCTARGETDIR);

    // Display the values of SRCDIR and INSTALLDIR.
    sprintfBox (INFORMATION, "Restored Source and Target Folders",
               "Source:\n\n%s\n\n Target:\n\n%s ",
               SRCDIR, INSTALLDIR);

end;

```

## VarSave




The **VarSave** function saves the current values of the system variables TARGETDIR (in InstallScript installations), INSTALLDIR (in Basic MSI and InstallScript MSI installations), SRCDIR, or HKEYCURRENTROOTKEY, which are used by many other InstallScript functions. Call **VarSave** whenever you need to change the value of these system variables temporarily, for example, to set source and target directories before a call to **XCOPYFile**. Afterwards, set those variables back to their previous values by calling **VarRestore**.

### Syntax

```
VarSave (nType);
```

## Parameters

**Table 11 • VarSave Parameters**

Parameter	Description
<b>nType</b>	<p>Specifies which system variables to save. Pass one or more of the following predefined constants in this parameter. Note that you can pass multiple constants separated by the OR operator (  ).</p> <p></p> <p><b>Project</b> • This information applies to InstallScript projects:</p> <ul style="list-style-type: none"> <li>• <b>VAR_LOGGING</b>—Resets any stored uninstallation logging entries.</li> <li>• <b>VAR_CURRENTDIR</b>—Resets any stored current directory entries.</li> <li>• <b>VAR_ALLSUPPORTED</b>—Resets all stored entries.</li> <li>• <b>VAR_HKEYCURRENTROOTKEY</b>—Resets any stored current root key entries.</li> <li>• <b>VAR_REGOPTIONS</b>—Resets any stored current registry options.</li> <li>• <b>CURRENTROOTKEY</b>—This constant is obsolete. Use VAR_HKEYCURRENTROOTKEY instead.</li> </ul> <p></p> <p><b>Project</b> • This constant applies to Basic MSI, InstallScript, and InstallScript MSI projects:</p> <ul style="list-style-type: none"> <li>• <b>VAR_SRCTARGETDIR or SRCTARGETDIR</b>—Resets any stored current TARGETDIR (in InstallScript installations), INSTALLDIR (in Basic MSI and InstallScript MSI installations), and SRCDIR entries.</li> </ul> <p></p> <p><b>Project</b> • This constant applies to Basic MSI and InstallScript MSI projects:</p> <ul style="list-style-type: none"> <li>• <b>SRCINSTALLDIR</b>—This constant is obsolete. Use VAR_SRCTARGETDIR instead.</li> </ul>

## Return Values

**Table 12 • VarSave Return Values**

Return Value	Description
<b>0</b>	Indicates that the current values of the system variables were saved.
<b>&lt; 0</b>	Indicates that the values were not saved due to an internal error. This error should occur only if the system is low on available memory.

## Additional Information

Each time you call **VarSave**, the InstallScript engine pushes the current values of the system variables onto an

internal stack, which operates as a last in, first out storage area. This method allows you to stack a series of values in memory. You can then make calls to **VarRestore** to retrieve those values from the stack in the opposite order that you stored them.

For example, if you call **VarSave** three times with SRCTARGETDIR as its argument (without calling **VarRestore** in between those calls), there will be three sets of SRCDIR and TARGETDIR values on the stack. The first call to **VarRestore** with SRCTARGETDIR as its argument restores the values from the third call to **VarSave**. The next call to **VarRestore** restores the values from the second call to **VarSave**. The third call to **VarRestore** restores the values from the first call to **VarSave**. At that point, the stack is empty. To see a script fragment that illustrates this process, refer to [VarSave Stack Example](#).

## VarSave Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the VarSave and VarRestore functions.
*
* This script starts by displaying the starting values of the
* system variables SRCDIR and INSTALLDIR. It then calls VarSave
* to save those values. Next, it assigns new values to SRCDIR
* and INSTALLDIR, and it displays those values. Finally, it
* calls VarRestore to restore the original values, which are
* then displayed.
*
\*-----*/

#define NEW_SOURCE_DIR "C:\\Source"
#define NEW_INSTALL_DIR "C:\\Target"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_VarSave(HWND);

function ExFn_VarSave(hMSI)
begin
    // Display the values of SRCDIR and INSTALLDIR.
    sprintfBox (INFORMATION, "Starting Source and Target Folders",
        "Source:\n\n%s\n\n Target:\n\n%s ",
        SRCDIR, INSTALLDIR);

    // Save the current values of SRCDIR and INSTALLDIR.
    VarSave (SRCTARGETDIR);

    // Assign new values to SRCDIR and INSTALLDIR.
    SRCDIR = NEW_SOURCE_DIR;
    INSTALLDIR = NEW_INSTALL_DIR;
```

```

// Display the values of SRCDIR and INSTALLDIR.
sprintfBox (INFORMATION, "New Source and Target Folders",
            "New Source:\n\n%s\n\n New Target:\n\n%s",
            SRCDIR, INSTALLDIR);

// Restore the old values.
VarRestore (SRCTARGETDIR);

// Display the values of SRCDIR and INSTALLDIR.
sprintfBox (INFORMATION, "Restored Source and Target Folders",
            "Source:\n\n%s\n\n Target:\n\n%s ",
            SRCDIR, INSTALLDIR);

end;

```

## VarSave Stack Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

// This script fragment illustrates how VarSave
// and VarRestore work with an internal stack
// to save and retrieve the values assigned to
// SRCDIR and INSTALLDIR

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_VarSave Stack(HWND);

function ExFn_VarSave Stack(hMSI)
begin
    // Store the starting values of SRCDIR and INSTALLDIR
    VarSave(SRCTARGETDIR);

    // Assign new values to SRCDIR and INSTALLDIR
    SRCDIR    = "E:\\";
    INSTALLDIR = "C:\\Program Files";

    // . . .

    // Store those values ("E:\\\" and "C:\\Program Files")
    VarSave(SRCTARGETDIR);

    // Assign new values to SRCDIR and INSTALLDIR
    SRCDIR    = "A:\\";
    INSTALLDIR = "C:\\Windows";

    // . . .

    // Store those new values ("A:\\\" and "C:\\Windows")
    VarSave(SRCTARGETDIR);

    // . . .

```

```

// Restore the values from the third call to VarSave
VarRestore(SRCTARGETDIR);

// SRCDIR is now "A:\\
// INSTALLDIR is now "C:\\Windows"

// . . .

// Restore the values from the second call to VarSave
VarRestore(SRCTARGETDIR);

// SRCDIR is now "E:\\
// INSTALLDIR is now "C:\\Program Files"

// . . .

// Restore the values from the first call to VarSave
VarRestore(SRCTARGETDIR);

// SRCDIR and INSTALLDIR now have their starting values.

end;

```

## VerCompare

The **VerCompare** function compares two strings that contain version information and returns whether the first one is less than, greater than, or equal to the second one.



**Important** • The format of the versions that you pass through the `szVersionInfo1` and `szVersionInfo2` parameters must be in the format **w.x.y.z**, where **w**, **x**, **y**, and **z** represent numbers. All four fields must be present; otherwise, **VerCompare** cannot successfully compare the two version strings.

For example, the following entries are valid `szVersionInfo1` and `szVersionInfo2` parameters:

- "1.0.0.0"
- "10.10.20.10"
- "3.21.01.2"

The following entries are not valid parameters:

- "1.20"
- "1.12.3"
- "2"

If one of the version strings has fewer than four fields, consider using the concatenate string operator (+) to add a decimal point and the number 0 as needed.

### Syntax

```
VerCompare ( szVersionInfo1, szVersionInfo2, nCompareFlag );
```

## Parameters

**Table 13 • VerCompare Parameters**

Parameter	Description
<b>szVersionInfo1</b>	Specify the first version string in the format <b>w.x.y.z</b> , where <b>w</b> , <b>x</b> , <b>y</b> , and <b>z</b> represent numbers.
<b>szVersionInfo2</b>	Specify the second version string in the format <b>w.x.y.z</b> , where <b>w</b> , <b>x</b> , <b>y</b> , and <b>z</b> represent numbers.
<b>nCompareFlag</b>	Specify the predefined constant VERSION to indicate that you are comparing version numbers. No other values are allowed in this parameter.

## Return Values

**Table 14 • VerCompare Return Values**

Return Value	Description
<b>EQUALS (2)</b>	Indicates that the two strings are of equal value.
<b>LESS_THAN (1)</b>	Indicates that the first string contains a value less than the second.
<b>GREATER_THAN (0)</b>	Indicates that the first string contains a value greater than the second.
<b>ISERR_GEN_FAILURE (-1)</b>	Indicates that the function could not compare the two strings.

## VerCompare Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the VerFindFileVersion and VerCompare functions.
*
* This script calls VerFindFileVersion to find the target file
* and retrieve its version information. If the specified
* target file is not found, the source file is copied to
* INSTALLDIR. If found, VerCompare is called to compare the
* version number of the EXAMPLE file found on the target
* system (the target file) to the version number of the file
* in SRCDIR (the source file). If the source file version
* number is newer than the target file, the target file is
* overwritten with the source file.
*
\*-----*/
```

```

#define EXAMPLE    "Stirinfcdll"
#define SOURCE_VER "2.0.1.0"
#define TITLE      "VerCompare and VerFindFileVersion"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_VerFindFileVersion(HWND);

function ExFn_VerFindFileVersion(hMSI)
    STRING szFileName, svPath, svVersionNumber, szExistingVersion, szUpdateVersion;
    STRING szTitle, szMsg;
    NUMBER nResult, nCompareFlag;
begin
    // Set the file to be updated.
    szFileName = EXAMPLE;

    // Find szFileName on the target system and retrieve its version number.
    nResult = VerFindFileVersion(szFileName, svPath, svVersionNumber);

    if (nResult = FILE_NOT_FOUND) then
        // If szFileName not found, copy source file to INSTALLDIR.
        szMsg = "Unable to locate %s. Copying %s to %s.";
        sprintfBox (INFORMATION, TITLE, szMsg, szFileName, szFileName,
                    INSTALLDIR);

        CopyFile (szFileName, szFileName);
        abort;
    elseif (nResult = FILE_NO_VERSION) then

        // If no version number found, copy source file to svPath and exit.
        szMsg = "%s version number not found. Copying %s to %s.";
        sprintfBox (INFORMATION, TITLE, szMsg, szFileName, szFileName,
                    INSTALLDIR);

        CopyFile (szFileName, szFileName);
        abort;
    elseif (nResult < 0) then
        MessageBox ("VerFindFileVersion failed.", SEVERE);
        abort;
    endif;

    // Compare the versions of the two files. It is assumed
    // that the source version number is known.
    szExistingVersion = svVersionNumber;
    MessageBox (szExistingVersion, INFORMATION);

    szUpdateVersion = SOURCE_VER;
    MessageBox (szUpdateVersion, INFORMATION);

    nCompareFlag = VERSION;

    nResult = VerCompare (szUpdateVersion, szExistingVersion, nCompareFlag);

    // If the source file is a more current version, install it.
    if (nResult = GREATER_THAN) then
        szMsg      = "%s updated into the %s directory.";

```



```

    SprintfBox (INFORMATION, TITLE, szMsg, szFileName, INSTALLDIR);

    CopyFile (szFileName, szFileName);

    // If the target file is a more current version, do not install.
    elseif (nResult = LESS_THAN) then
        szMsg = "No need for an upgrade, the most current version of %s is " +
            "installed.";
        SprintfBox (INFORMATION, TITLE, szMsg, szFileName);

    // If both the target and source versions are the same, do not install.
    elseif (nResult = EQUALS) then
        MessageBox ("Versions are equal. No update needed.", INFORMATION);
    endif;

end;

```

## VerFindFileVersion

The **VerFindFileVersion** function searches for a specified file and retrieves the file version and location.

**VerFindFileVersion** uses the following search algorithm to find the file (searches the folders in the following order):

1. Windows folder
2. Windows system folder
3. The folder specified by the TARGETDIR system variable (in InstallScript installations) or the INSTALLDIR system variable (in Basic MSI and InstallScript MSI installations)
4. The folders specified by the PATH environment variable
5. The folder from which **Setup.exe** is running

For information about the Windows system folder, see the documentation for the InstallScript system variable [WINSYSDIR](#).



**Note** • When using **VerFindFileVersion**, you might need to set a value for TARGETDIR (in InstallScript installations) or INSTALLDIR (in Basic MSI and InstallScript installations) other than the value that is automatically set by the InstallScript engine. Since the function looks for the file in the TARGETDIR or INSTALLDIR folder, you might need to reset the value of the system variable temporarily to ensure that **VerFindFileVersion** finds the file. If you need to do this, use **VarSave** to save the value of TARGETDIR or INSTALLDIR before temporarily setting it to another folder. After the **VerFindFileVersion** function call, reset TARGETDIR or INSTALLDIR using **VarRestore**.

### Syntax

```
VerFindFileVersion ( szFileName, svPath, svVersionNumber );
```

## Parameters

**Table 15 • VerFindFileVersion Parameters**

Parameter	Description
<b>szFileName</b>	Specifies the unqualified name of the file whose version is to be obtained. Do not specify a drive designation or path in this parameter.
<b>svPath</b>	Returns the complete path (including the drive designation) to the folder in which the file resides.
<b>svVersionNumber</b>	Returns the version number of the file in the following format:  <div style="text-align: center;">&lt;major version&gt;.&lt;minor version&gt;</div> For example, if svVersionNumber returns 2.1.2.0, the major version number is 2.1 and the minor version number is 2.0.

## Return Values

**Table 16 • VerFindFileVersion Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully returned the version information.
<b>FILE_NO_VERSION (-8)</b>	Indicates that the file was found but did not contain version information.
<b>FILE_NOT_FOUND (-2)</b>	Indicates that the file was not found.

## VerFindFileVersion Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the VerFindFileVersion and VerCompare functions.
*
* This script calls VerFindFileVersion to find the target file
* and retrieve its version information. If the specified
* target file is not found, the source file is copied to
* INSTALLDIR. If found, VerCompare is called to compare the
* version number of the EXAMPLE file found on the target
* system (the target file) to the version number of the file
* in SRCDIR (the source file). If the source file version
* number is newer than the target file, the target file is
* overwritten with the source file.
*
```

```

\*-----*/

#define EXAMPLE    "Stirinfcdll"
#define SOURCE_VER "2.0.1.0"
#define TITLE      "VerCompare and VerFindFileVersion"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_VerFindFileVersion(HWND);

function ExFn_VerFindFileVersion(hMSI)
    STRING szFileName, svPath, svVersionNumber, szExistingVersion, szUpdateVersion;
    STRING szTitle, szMsg;
    NUMBER nResult, nCompareFlag;
begin

    // Set the file to be updated.
    szFileName = EXAMPLE;

    // Find szFileName on the target system and retrieve its version number.
    nResult = VerFindFileVersion(szFileName, svPath, svVersionNumber);

    if (nResult = FILE_NOT_FOUND) then
        // If szFileName not found, copy source file to INSTALLDIR.
        szMsg = "Unable to locate %s. Copying %s to %s.";
        sprintfBox (INFORMATION, TITLE, szMsg, szFileName, szFileName,
                    INSTALLDIR);

        CopyFile (szFileName, szFileName);
        abort;
    elseif (nResult = FILE_NO_VERSION) then

        // If no version number found, copy source file to svPath and exit.
        szMsg = "%s version number not found. Copying %s to %s.";
        sprintfBox (INFORMATION, TITLE, szMsg, szFileName, szFileName,
                    INSTALLDIR);

        CopyFile (szFileName, szFileName);
        abort;
    elseif (nResult < 0) then
        MessageBox ("VerFindFileVersion failed.", SEVERE);
        abort;
    endif;

    // Compare the versions of the two files. It is assumed
    // that the source version number is known.
    szExistingVersion = svVersionNumber;
    MessageBox (szExistingVersion, INFORMATION);

    szUpdateVersion = SOURCE_VER;
    MessageBox (szUpdateVersion, INFORMATION);

    nCompareFlag = VERSION;

    nResult = VerCompare (szUpdateVersion, szExistingVersion, nCompareFlag);

    // If the source file is a more current version, install it.
    if (nResult = GREATER_THAN) then

```

```

    szMsg      = "%s updated into the %s directory.";
    sprintfBox (INFORMATION, TITLE, szMsg, szFileName, INSTALLDIR);

    CopyFile (szFileName, szFileName);

    // If the target file is a more current version, do not install.
    elseif (nResult = LESS_THAN) then
        szMsg = "No need for an upgrade, the most current version of %s is " +
            "installed.";
        sprintfBox (INFORMATION, TITLE, szMsg, szFileName);

    // If both the target and source versions are the same, do not install.
    elseif (nResult = EQUALS) then
        MessageBox ("Versions are equal. No update needed.", INFORMATION);
    endif;

end;

```

## VerGetFileLanguages

The **VerGetFileLanguages** function retrieves the list of languages supported by the file that is specified by szFile.

### Syntax

```
VerGetFileLanguages ( szFileName, listLanguages );
```

## Parameters

**Table 17** • VerGetFileLanguages Parameters

Parameter	Description
<b>szFileName</b>	Specifies the fully qualified name of the file whose list of supported languages is to be retrieved.
<b>listLanguages</b>	<p>Returns the list of numeric language codes and code page IDs of the supported languages. Each list element is a 32-bit integer whose low-order word contains the supported language code and whose high-order word contains the corresponding code page ID, which can be extracted with the functions <a href="#">LOWORD</a> and <a href="#">HIWORD</a>.</p> <p>The number list identified by listLanguages must already have been initialized by a call to <a href="#">ListCreate(NUMBERLIST)</a>.</p>

## Return Values

**Table 18** • VerGetFileLanguages Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	The function successfully retrieved the list of supported languages.
<b>&lt; ISERR_SUCCESS</b>	The function failed to retrieve the list of supported languages.

## VerGetFileLanguages Example

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the VerGetFileLanguages function.
 *
 \*-----*/

function OnBegin()
    number nListGetItem, nvLanguageInfo, nLanguageCode, nCodePage;
    string szFileName;
    LIST listLanguages, listLanguageCodes, listCodePages;
begin
    // Get file language information.
    szFileName = "C:\\Program Files\\Internet Explorer\\Iexplore.exe";

```

```

listLanguages = ListCreate( NUMBERLIST );
VerGetFileLanguages( szFileName, listLanguages );

// Extract language codes and code page IDs
// from list items and add to new lists.
listLanguageCodes = ListCreate( NUMBERLIST );
listCodePages = ListCreate( NUMBERLIST );
nListItem = ListGetFirstItem( listLanguages ,nvLanguageInfo);
while nListItem!=0
    nLanguageCode = LOWORD( nvLanguageInfo );
    ListAddItem( listLanguageCodes, nLanguageCode, AFTER );
    nCodePage = HIWORD( nvLanguageInfo );
    ListAddItem( listCodePages, nCodePage, AFTER );
    nListItem = ListGetNextItem( listLanguages ,nvLanguageInfo);
endwhile;
end;

```

## VerGetFileVersion

The **VerGetFileVersion** function retrieves the numeric version information of the specified file.



**Note** • Although InstallScript's file version functions take version information in string format, the version information they reference in a file is the numeric version information. A file's string version information is not inspected or returned by InstallScript functions. Further, when Windows Explorer displays a file's properties, it shows the string version information, which might not be equivalent to the file's numeric version information. For that reason, the value returned in `svVersionNumber` by `VerGetFileVersion` might not match the version information displayed by Windows Explorer.

For more about file version information, consult the Windows documentation.

### Syntax

```
VerGetFileVersion ( szFileName, svVersionNumber );
```

## Parameters

**Table 19 • VerGetFileVersion Parameters**

Parameter	Description
<b>szFileName</b>	Specifies the fully qualified name of the file whose numeric version information is to be retrieved.
<b>svVersionNumber</b>	<p>Returns the numeric version information as a string in the following format:</p> <p style="text-align: center;">&lt;major version&gt;.&lt;minor version&gt;</p> <p>For example, if svVersionNumber returns 2.1.2.0, the major version number is 2.1 and the minor version number is 2.0.</p>

## Return Values

**Table 20 • VerGetFileVersion Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully returned the version information.
<b>FILE_NOT_FOUND (-2)</b>	Indicates that the specified file was not found.
<b>FILE_NO_VERSION (-8)</b>	Indicates that the file was found but did not contain version information.

## VerGetFileVersion Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the VerGetFileVersion function.
*
* The script below calls VerGetFileVersion to retrieve the
* version number of the Windows Notepad. The information is
* displayed in a message box.
*
\*-----*/

#define EXAMPLE_FILE WINDIR ^ "Notepad.exe"
#define TITLE_TEXT "VerGetFileVersion Example"

// Include Ifx.h for built-in InstallScript function prototypes.
```

```

#include "Ifx.h"

export prototype ExFn_VerGetFileVersion(HWND);

function ExFn_VerGetFileVersion(hMSI)
    NUMBER nResult;
    STRING szFile, szPath, szMsg, svVersionNumber;
begin

    // Get the version number of the specified file.
    nResult = VerGetFileVersion(EXAMPLE_FILE, svVersionNumber);

    // Report the results of VerGetFileVersion.
    if (nResult = FILE_NO_VERSION) then
        szMsg = EXAMPLE_FILE + " does not contain version information.";
        MessageBox (szMsg, INFORMATION);
    elseif (nResult = FILE_NOT_FOUND) then
        szMsg = EXAMPLE_FILE + " could not be found.";
        MessageBox (szMsg, INFORMATION);
    else
        szMsg = "The version number of %s is %s";
        sprintfBox (INFORMATION, TITLE_TEXT, szMsg,
            EXAMPLE_FILE, svVersionNumber);
    endif;

end;

```

## VerProductCompareVersions



**Project** • This information applies to InstallScript projects.

The **VerProductCompareVersions** function compares version information and returns a value indicating the result of the comparison.

### Syntax

```
VerProductCompareVersions ( );
```

### Parameters

None.



## Return Values

**Table 21** • VerProductCompareVersions Return Values

Return Value	Description
<b>VERSION_COMPARE_RESULT_NOT_INSTALLED</b>	No version of the product is installed or the installed version was not found (IFX_INSTALLED_VERSION is a null string).
<b>VERSION_COMPARE_RESULT_SAME</b>	The update setup's version is the same as the version currently installed on the target system.
<b>VERSION_COMPARE_RESULT_OLDER</b>	The update setup's version is older than the version currently installed on the target system.
<b>VERSION_COMPARE_RESULT_NEWER_NOT_SUPPORTED</b>	The update setup's version is newer than the version currently installed on the target system, but the currently installed version is not supported by the update setup.
<b>VERSION_COMPARE_RESULT_NEWER</b>	The update setup's version is newer than the version currently installed on the target system, and the currently installed version is supported by the update setup.
<b>&lt; ISERR_SUCCESS</b>	The function failed.

## Comments

The function compares the values of the system variables [IFX\\_INSTALLED\\_VERSION](#) and [IFX\\_PRODUCT\\_VERSION](#) to determine if the version currently installed on the target system is older than the update setup's version; if so, the function compares the values of the system variables [IFX\\_INSTALLED\\_VERSION](#) and [IFX\\_SUPPORTED\\_VERSIONS](#) to determine if the update setup applies to the currently installed version.

VerProductCompareVersions is called by the default code for the OnSetUpdateMode and OnUpdateUIBefore event handler functions.

# VerProductGetInstalledVersion



**Project** • This information applies to InstallScript projects.

The **VerProductGetInstalledVersion** function returns in svVersionInstalled the string equivalent of the data in the Version value of the application uninstallation registry key if that data is a packed DWORD.

## Syntax

```
VerProductGetInstalledVersion ( svVersionInstalled );
```

## Parameters

**Table 22** • VerProductGetInstalledVersion Parameters

Parameter	Description
<b>svVersionInstalled</b>	Returns the string equivalent of the data in the application uninstallation registry key's Version value. If the function fails, the value of svVersionInstalled is not changed.

## Return Values

**Table 23** • VerProductGetInstalledVersion Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	The function successfully retrieved the installed version information.
<b>&lt; ISERR_SUCCESS</b>	The function failed to retrieve the installed version information; the registry key or value does not exist or the data is not a packed DWORD.

# VerProductIsVersionSupported



**Project** • This information applies to InstallScript projects.

The **VerProductIsVersionSupported** function checks whether the version string in szVersionCheck is one of the versions in szVersionSupported.

## Syntax

```
VerProductIsVersionSupported ( szVersionCheck, szVersionSupported );
```

## Parameters

**Table 24** • VerProductIsVersionSupported Parameters

Parameter	Description
<b>szVersionCheck</b>	Specifies a version string in packed DWORD format (for example, the initial value of the system variable <a href="#">IFX_PRODUCT_VERSION</a> ).
<b>szVersionSupported</b>	Specifies a pipe( )-delimited list of version strings in packed DWORD format (for example, the initial value of the system variable <a href="#">IFX_SUPPORTED_VERSIONS</a> ).

## Return Values

**Table 25** • VerProductIsVersionSupported Return Values

Return Value	Description
<b>TRUE</b>	szVersionCheck is one of the versions in szVersionSupported, or szVersionSupported is a null string ("").
<b>FALSE</b>	szVersionCheck is not one of the versions in szVersionSupported.

# VerProductNumToStr



**Project** • This information applies to InstallScript projects.

The **VerProductNumToStr** function returns in svVersion the version string corresponding to the packed DWORD specified in nVersion.

## Syntax

```
VerProductNumToStr ( svVersion, nVersion );
```

## Parameters

**Table 26** • VerProductNumToStr Parameters

Parameter	Description
<b>svVersion</b>	Returns the string equivalent, in packed DWORD format, of nVersion.
<b>nVersion</b>	Specifies version information as a four-byte value, that is, a number in the range 0 to 4294967295.

## Return Values

**Table 27** • VerProductNumToStr Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	The function succeeded.
<b>&lt; ISERR_SUCCESS</b>	The function failed.

# VerProductStrToNum



**Project** • This information applies to InstallScript projects.

The **VerProductStrToNum** function returns in nvVersionResult the packed DWORD version information corresponding to the string specified in svVersion.

## Syntax

```
VerProductStrToNum ( nvVersionResult, szVersion );
```

## Parameters

**Table 28** • VerProductStrToNum Parameters

Parameter	Description
<b>nvVersionResult</b>	Returns the packed DWORD version information.
<b>szVersion</b>	Specifies version information in string form. The function fails unless szVersion meets all of the following conditions: <ul style="list-style-type: none"> <li>• szVersion is in the format major.minor.build or major.minor.build.release (release is ignored when determining nvVersionResult).</li> <li>• major and minor are the string equivalents of numbers in the range 0 to 255.</li> <li>• build is the string equivalent of a number in the range 0 to 65535.</li> </ul>

## Return Values

**Table 29** • VerProductStrToNum Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	The function succeeded.
<b>&lt; ISERR_SUCCESS</b>	The function failed.

# VerProductVerFromVerParts



**Project** • This information applies to InstallScript projects.

The **VerProductVerFromVerParts** function retrieves the packed DWORD that corresponds to the version parts specified by nVersionMajor, nVersionMinor, and nVersionBuild.

## Syntax

```
VerProductVerFromVerParts ( nvVersion, nVersionMajor, nVersionMinor, nVersionBuild );
```

## Parameters

**Table 30** • VerProductVerFromVerParts Parameters

Parameter	Description
<b>nvVersion</b>	Returns the four-byte value that corresponds to the version parts specified by nVersionMajor, nVersionMinor, and nVersionBuild.
<b>nVersionMajor</b>	Specifies the first byte of version information. VerProductVerFromVerParts fails if this value is less than zero (0) or greater than 255.
<b>nVersionMinor</b>	Specifies the second byte of version information. VerProductVerFromVerParts fails if this value is less than zero (0) or greater than 255.
<b>nVersionBuild</b>	Specifies the last two bytes of version information. VerProductVerFromVerParts fails if this value is less than zero (0) or greater than 65535.

## Return Values

**Table 31** • VerProductVerFromVerParts Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	Indicates that the function succeeded.
<b>&lt; ISERR_SUCCESS</b>	Indicates that the function failed.

# VerProductVerPartsFromVer



**Project** • This information applies to InstallScript projects.

The **VerProductVerPartsFromVer** function retrieves as separate numeric values the version parts of the packed DWORD specified by nVersion.

## Syntax

```
VerProductVerPartsFromVer ( nVersion, nvVersionMajor, nvVersionMinor, nvVersionBuild );
```

## Parameters

**Table 32 • VerProductVerPartsFromVer Parameters**

Parameter	Description
<b>nVersion</b>	Specifies version information as a four-byte value, that is, a number in the range 0 to 4294967295.
<b>nvVersionMajor</b>	Returns the first byte of nVersion.
<b>nvVersionMinor</b>	Returns the second byte of nVersion.
<b>nvVersionBuild</b>	Returns the last two bytes of nVersion.

## Return Values

**Table 33 • VerProductVerPartsFromVer Return Values**

Return Value	Description
<b>ISERR_SUCCESS</b>	This function always returns ISERR_SUCCESS.

# VerSearchAndUpdateFile

The **VerSearchAndUpdateFile** function searches for the specified file and installs a newer version of the file if necessary. If the function finds the file, it compares the version number of the existing file to the version number of the new file. If the existing file is older, it is replaced with the new file. The new file must be in the directory specified by the system variable SRCDIR. If the function does not find an existing file, it copies the new file to the target system. Windows decides where the file is installed depending on the type of the file. For example, DLLs and system drivers are installed in the Windows system folder. For information about the Windows system folder, see the documentation for the InstallScript system variable [WINSYSDIR](#).

**VerFindFileVersion** uses the following search algorithm to find the file (searches the folders in the following order):

1. Windows folder
2. Windows system folder
3. The folder specified by the TARGETDIR system variable (in InstallScript installations) or the INSTALLEDIR system variable (in Basic MSI or InstallScript MSI installations)
4. The folders specified by the PATH environment variable
5. The folder from which **Setup.exe** is running



**Note •** For file transfer, an alternative to **VerSearchAndUpdateFile** is **XCOPYFile**—which can do check version, mark locked .dll and .exe files for update after system reboot, and increment registry reference counters for shared .dll and .exe files.

## Syntax

```
VerSearchAndUpdateFile ( szFileName, nUpdateFlag, svInstalledFile );
```



## Parameters

**Table 34 • VerSearchAndUpdateFile Parameters**

Parameter	Description
<b>szFileName</b>	Specifies the unqualified name of the file to install. Do not specify a drive designation or path in this parameter.
<b>nUpdateFlag</b>	Specifies whether the file should be updated unconditionally or only if the version of the file found on the target system is older than the version of the file that you have shipped. Pass one of the following predefined constants in this parameter: <ul style="list-style-type: none"> <li>• <b>VER_UPDATE_COND</b>—Updates the existing file only if it is an older version.</li> <li>• <b>VER_UPDATE_ALWAYS</b>—Updates the existing file even if it is a newer version.</li> </ul>
<b>svInstalledFile</b>	Returns the fully qualified name of the file installed by the function. If the file you want to replace is in use, the file is installed to the same directory with a slightly different name. The file is renamed with a tilde (~) character in the first character of the extension. For example, if you are installing the file Shell.dll and the file is locked, the file is copied as Shell.~ll. The name is returned in this variable.

## Return Values

**Table 35 • VerSearchAndUpdateFile Return Values**

Return Value	Description
<b>FILE_INSTALLED (0)</b>	Function was successful in installing the file.
<b>FILE_IS_LOCKED (-4)</b>	Indicates that the file is in use by Windows and cannot be replaced. The new file is copied to the same directory with a new name.
<b>FILE_NO_VERSION (-8)</b>	Indicates that the file was found, but it did not contain version information. File update is not performed.
<b>FILE_RD_ONLY (-5)</b>	Indicates that the existing file is write protected. The script should reset the read-only flag of the destination file before proceeding with the setup and then attempt to install the file again.
<b>FILE_SRC_OLD (-7)</b>	Indicates that the file to install has the same date or is older than the preexisting file.
<b>OUT_OF_DISK_SPACE (-6)</b>	Indicates that the function cannot create the file due to insufficient disk space on the destination drive. File update is not performed.
<b>VER_DLL_NOT_FOUND (-3)</b>	Indicates Ver.dll was not found. File update is not performed.
<b>OTHER_FAILURE (-1)</b>	Indicates an unspecified error occurred. File update is not performed.

## VerSearchAndUpdateFile Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the VerSearchAndUpdateFile function.
 *
 * The first call to VerSearchAndUpdateFile replaces the file
 * specified by the constant UPDATE_FILE1 regardless of the
 * version on the target system.
 *
 * The second call to VerSearchAndUpdateFile replaces the file
 * specified by the constant UPDATE_FILE2 only if the version
 * in the source directory is newer than the version in the
 * target directory.
 *
 \*-----*/

#define UPDATE_FILE1 "Example.txt"
#define UPDATE_FILE2 "Readme.txt"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_VerSearchAndUpdateFile(HWND);

function ExFn_VerSearchAndUpdateFile(hMSI)
    STRING szFileName, svInstalledFile, szTitle, szMsg;
    NUMBER nUpdateFlag, nResult;
    BOOL    bDone;
begin

    // Set up title and message parameters for call
    // to VerSearchAndUpdateFile.
    szTitle = "VerSearchAndUpdateFile Example";
    szMsg    = " was successfully updated.";

    // Update UPDATE_FILE1 regardless of version number.
    if (VerSearchAndUpdateFile (UPDATE_FILE1, VER_UPDATE_ALWAYS,
                               svInstalledFile) = 0) then
        SprintfBox (INFORMATION, szTitle, UPDATE_FILE1 + szMsg);
    endif;

    // Set indicator to control loop exit.
    bDone = FALSE;

    // Begin the while loop.
    while (bDone = FALSE)
        // Update UPDATE_FILE2 only if existing file is older.
        nResult = VerSearchAndUpdateFile (UPDATE_FILE2, VER_UPDATE_COND,
                                           svInstalledFile);

        switch (nResult)
```

```

case 0:
    // VerSearchAndUpdate successful.
    sprintfBox (INFORMATION, szTitle, UPDATE_FILE2 + szMsg);
    bDone = TRUE;
// The target file does not have a version number.
case FILE_NO_VERSION:
    // Ask the user if the file should be updated regardless.
    if (AskYesNo ("Version number was not found.\nDo you still wish " +
        "to update " + UPDATE_FILE2 + "?", YES) = YES) then
        // Update the file UPDATE_FILE2 regardless of version number.
        VerSearchAndUpdateFile (UPDATE_FILE2, VER_UPDATE_ALWAYS,
            svInstalledFile);

        bDone = TRUE;
    else
        bDone = TRUE;
    endif;

// The target file is locked.
case FILE_IS_LOCKED:
    MessageBox ("The target file is locked.\n\nPlease close all " +
        "programs and run Setup again.", INFORMATION);

    bDone = TRUE;
// The target file is read-only.
case FILE_RD_ONLY:
    // Ask the user if Setup should remove the read-only attribute.
    if (AskYesNo ("File is read-only.\nShould Setup remove the " +
        "write-protection of " + UPDATE_FILE2 + "?", YES) = YES) then
        // Change the attribute of the target file to normal.
        SetFileInfo (svInstalledFile, FILE_ATTRIBUTE, FILE_ATTR_NORMAL, "");
        bDone = FALSE;
    else
        bDone = TRUE;
    endif;
// The target disk does not have enough space.
case OUT_OF_DISK_SPACE:
    MessageBox ("You need more free space for this update.", SEVERE);
    bDone = TRUE;
// The required VER.DLL file was not found.
case VER_DLL_NOT_FOUND:
    MessageBox ("VER.DLL was not found.", SEVERE);
    bDone = TRUE;
// Some other error occurred.
case OTHER_FAILURE:
    MessageBox ("Update has failed.", SEVERE);
    bDone = TRUE;
default:
    bDone = TRUE;
endswitch;

endwhile;

end;

```

## VerUpdateFile

The **VerUpdateFile** function uses the version information of a specified file to determine whether or not to install the file on the target directory. VerUpdateFile gets the file name specified in szFileName.

VerFindFileVersion uses the following search algorithm to find the file (searches the folders in the following order):

1. Windows folder
2. Windows system folder
3. The folder specified by the TARGETDIR system variable (in InstallScript installations) or the INSTALLDIR system variable (in Basic MSI or InstallScript MSI installations)
4. The folders specified by the PATH environment variable
5. The folder from which **Setup.exe** is running

VerUpdateFile then compares the version of the file with the same name in SRCDIR (the source file), if it exists, with the version of the target file. If the source file has a more recent version number than the target file, the target file is replaced by the source file. If the target file does not exist, InstallShield copies the source file to the target location.

When the SHAREDFILE or LOCKEDFILE option is used in the parameter nUpdateFlag and .dll or .exe files to be updated are in use by the system, renamed copies of the source files are transferred to the target system and the system variable BATCH\_INSTALL is set to TRUE. Then, when **RebootDialog** or **SdFinishReboot** is called at the end of the setup and the system is restarted, the locked files are updated. For more information on updating locked files, see [RebootDialog](#) and [SdFinishReboot](#). The system variable BATCH\_INSTALL can be tested to determine if locked .dll or .exe files were encountered. You cannot use the SHAREDFILE and LOCKEDFILE options simultaneously-you must use one or the other.

For file transfer, preferable alternatives to VerUpdateFile may be XCopyFile, which can perform version checking, mark locked .dll and .exe files for update after system reboot, and increment registry reference counters for shared .dll and .exe files.

## Syntax

```
VerUpdateFile ( szFileName, nUpdateFlag, svInstalledFilePath );
```

## Parameters

**Table 36 • VerUpdateFile Parameters**

Parameter	Description
<b>szFileName</b>	<p>Specifies the qualified or unqualified name of the file to update. If the name is unqualified (that is, if it does not include a drive designation or path), InstallShield searches the Windows or Win95 directory, the System directory, the directories specified by the PATH environment variable, and then the path of the InstallShield executable file for a matching file. VerUpdateFile takes the file name portion of szFileName and uses it to identify the file in SRCDIR that is used as the source file.</p>
<b>nUpdateFlag</b>	<p>Specifies whether the file is updated unconditionally or updated only if the version of the target file found is older than the version of the source file. Pass one of the following predefined constants in this parameter. You can combine the constant SHAREDFILE with one of the other constants using the bitwise OR operator (   ). However, you cannot combine SHAREDFILE and LOCKEDFILE.</p> <ul style="list-style-type: none"> <li> <b>LOCKEDFILE</b>—Causes VerUpdateFile to record locked .dll and .exe files for update when Windows or the system is rebooted. A locked file is a file that is in use by an application or the system when InstallShield attempts to access or update the file. The LOCKEDFILE option works like SHAREDFILE except that LOCKEDFILE does not create registry entries or modify the registry reference counter. You cannot use the LOCKEDFILE option when using the SHAREDFILE option. There are some unshared files (such as shell extensions) for which the script writer does not want a registry entry and reference counter. These files should never be uninstalled, except by the application itself. LOCKEDFILE allows VerUpdateFile to handle locked files that are not shared. </li> <li> <b>SHAREDFILE</b>—Combines shared and locked file handling by causing VerUpdateFile to treat all files as shared, and to record locked .dll and .exe files for update when Windows or the system restarts. See RebootDialog and SdFinishReboot. </li> </ul> <p>The SHAREDFILE option causes VerUpdateFile to treat all files as shared files and increment the registry reference counter by one when the file exists in the target directory and it has a reference count greater than 0. If the shared file does not exist in the target directory and it has no reference counter, InstallShield creates the counter and sets it to 1. If the shared file already exists in the target directory but has no reference counter, InstallShield creates the counter and initializes it to 2 as a precaution against accidental removal during uninstallation.</p>

**Table 36 • VerUpdateFile Parameters (cont.)**

Parameter	Description
<b>nUpdateFlag (continued)</b>	<ul style="list-style-type: none"> <li> <b>SELFREGISTER</b>—Carries out the self-registration process immediately, when using the “non-batch method” of installing self-registering files.             If you have called <b>Enable(SELFREGISTERBATCH)</b>, this option queues up self-registering files for registration. The files are registered once <b>Do(SELFREGISTRATIONPROCESS)</b> is called, when using the “batch method” of installing self-registering files.             Always use SELFREGISTER together with the constant SHAREDFILE, using the bitwise OR operator (   ).         </li> <li> <b>VER_UPDATE_ALWAYS</b>—Updates the file regardless of the version numbers.         </li> <li> <b>VER_UPDATE_COND</b>—Updates the file only if the file being replaced is an older version.         </li> </ul>
<b>svInstalledFilePath</b>	<p>Returns the fully qualified name of the file that was installed. If the file you want to replace is in use, the file is installed to the same directory with a modified name. InstallShield uses a tilde (~) character to replace the first character of the file's extension.</p> <p>For example, if you are updating the file Shell.dll and the target file is locked, the source file is copied to the target directory as Shell.~ll. This name is returned in the parameter svInstalledFilePath.</p> <p>If the SHAREDFILE option is used in the parameter nUpdateFlag and locked files are properly committed for update when Windows or the system restarts, the ~ modified name versions of the files are deleted when the update takes place.</p>

## Return Values

**Table 37 • VerUpdateFile Return Values**

Return Value	Description
<b>FILE_INSTALLED (0)</b>	Indicates that the function successfully installed the file. This constant is equal to 0 (zero). All other return values are less than zero (< 0).
<b>FILE_IS_LOCKED (-4)</b>	Indicates that the existing file is in use by Windows and cannot be replaced. The new file is copied to the same directory with a new name, as described above.
<b>FILE_NO_VERSION (-8)</b>	Indicates that the file was found, but it did not contain version information in it. File update is not performed.
<b>FILE_RD_ONLY (-5)</b>	Indicates that the existing file is write-protected. You must reset the read-only bit in the destination file before proceeding with the setup, and then attempt to install the file again.
<b>FILE_SRC_EQUAL (-9)</b>	Indicates that the file you want to install has the same version as the existing file. File update is not performed if the VER_UPDATE_COND flag is set.
<b>FILE_SRC_OLD (-7)</b>	Indicates that the file you want to install is older than the existing file. File update is not performed if the VER_UPDATE_COND flag is set.
<b>OUT_OF_DISK_SPACE (-6)</b>	Indicates that the function cannot create the file due to insufficient disk space on the destination drive. File update is not performed.
<b>-51</b>	A self-registering file did not register successfully.
<b>OTHER_FAILURE (-1)</b>	Indicates an unspecified error occurred. File update is not performed.

## VerUpdateFile Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the VerUpdateFile function.
 *
 * This script calls VerUpdateFile twice to update a Windows
 * accessory.
 *
 \*-----*/

#define APPFILE "Notepad.exe"
#define TITLE  "VerUpdateFile Example"

```

```

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_VerUpdateFile(HWND);

function ExFn_VerUpdateFile(hMSI)
    STRING svInstalledFilePath, szTitle, szMsg;
    NUMBER nResult;
    BOOL    bDone;
begin
    // Update the file regardless of file version.
    nResult = VerUpdateFile (APPPFILE, VER_UPDATE_ALWAYS, svInstalledFilePath);

    if (nResult < 0) then
        MessageBox ("First call to VerUpdateFile failed.", SEVERE);
    else
        szMsg = "%s successfully updated.";
        sprintfBox (INFORMATION, TITLE, szMsg, APPFILE);
    endif;

    // Update the files only if the target files are more recent.
    nResult = VerUpdateFile (APPPFILE, VER_UPDATE_COND, svInstalledFilePath);

    if (nResult < 0) then
        MessageBox ("Second call to VerUpdateFile failed.", SEVERE);
    endif;

end;

```

## WaitForApplication

The **WaitForApplication** function waits for a running application to terminate before returning.

If the function fails to wait for the application to terminate before returning, verify that the application does not terminate until other sub-applications launched by the application terminate. The **WaitForApplication** function monitors the process handle of the specified application; if the application passes control to a secondary application or process and then terminates, the function returns immediately.

### Syntax

```

WaitForApplication( byval number hProcess, byval number dwProcessId, byval number nTimeOut, byval number
    nOptions );

```




## Parameters

**Table 38 • WaitForApplication Parameters**

Parameter	Description
<b>hProcess</b>	Specifies the process handle of the running process for which to wait. (If the application was launched with <b>LaunchAppAndWait</b> or <b>LaunchApp</b> , this handle is returned in the LAAW_PROCESS_INFORMATION.hProcess member.)
<b>dwProcessId</b>	<p>Specifies the process ID of the running process for which to wait. You need to specify a non-zero value for this parameter only if you specify LAAW_OPTION_WAIT_INCL_CHILD in nOptions.</p> <p>If the application was launched with <b>LaunchAppAndWait</b> or <b>LaunchApp</b> and the process ID can be determined (as described for the description of <a href="#">LAAW_OPTION_WAIT_INCL_CHILD</a>), this handle is returned in the LAAW_PROCESS_INFORMATION.dwProcessId member.</p>
<b>nTimeout</b>	Specifies the maximum amount of time (in milliseconds) to wait for the application to complete before returning. If this amount of time passes before the application terminates, the function ends the wait and returns failure. You can specify INFINITE to indicate that the function should wait indefinitely. (You can also pass the LAAW_PARAMETERS.nTimeout value to mimic the behavior of the <b>LaunchAppAndWait</b> function.)

**Table 38 •** WaitForApplication Parameters (cont.)

Parameter	Description
<b>nOptions</b>	<p>Pass one of the following predefined constants in this parameter. You can combine these constants by using the bitwise OR operator ( ), with the following exceptions: you cannot combine LAAW_OPTION_NOWAIT with LAAW_OPTION_WAIT.</p> <ul style="list-style-type: none"> <li>• <b>LAAW_OPTION_WAIT</b>—Specifies that the function should wait until the specified application has terminated.</li> </ul> <p></p> <p><b>Note</b> • If the specified applications fails to terminate and nOptions is set to LAAW_OPTION_WAIT (and nTimeOut is set to INFINITE), the installation will wait indefinitely for the launched application to complete.</p> <ul style="list-style-type: none"> <li>• <b>LAAW_OPTION_NOWAIT</b>—Specifies that the function should return immediately.</li> <li>• <b>LAAW_OPTION_USE_CALLBACK</b>—Causes the function to call the OnLaunchAppAndWaitCallback event every second, or whatever interval you set for the LAAW_PARAMETERS.nCallbackInterval member, while waiting for the application to terminate.</li> <li>• <b>LAAW_OPTION_WAIT_INCL_CHILD</b>—Indicates that the function should wait for the launched application and any of its direct child processes.</li> </ul> <p>When LAAW_OPTION_WAIT_INCL_CHILD is used, the function detects and waits for only direct child processes of the launched process—not for any additional child processes that are launched by child processes of the initially launched process.</p> <p>Note that to detect and wait for child processes with LAAW_OPTION_WAIT_INCL_CHILD, the function must receive or be able to determine the process ID of the launched process. Therefore, in order for this option to work, if dwProcessId is not specified, the Windows API <b>GetProcessId</b> must be available on the system so that the function can determine the process ID. According to the Windows API documentation, <b>GetProcessId</b> is available on Windows XP SP1 and later and Windows Server 2003 and later. If this API is not available, LAAW_OPTION_WAIT_INCL_CHILD behaves as LAAW_OPTION_WAIT.</p> <p>You can specify any <a href="#">LaunchApplication</a> option; however, only the aforementioned options will have any effect.</p>

## Return Values

**Table 39** • WaitForApplication Return Values

Return Value	Description
<b>&gt;= ISERR_SUCCESS</b>	The application terminated and the function returned as a result.
<b>&lt; ISERR_SUCCESS</b>	<p>The function returned as a result of something other than the application terminating. You can check the value of LAAW_PARAMETERS.nWaitResult to determine additional information:</p> <ul style="list-style-type: none"> <li>• <b>WAIT_OBJECT_0</b>—Indicates that the application terminated.</li> <li>• <b>WAIT_OBJECT_0 + 1</b>—Indicates that the installation received a message to terminate.</li> <li>• <b>WAIT_TIMEOUT</b>—Indicates that the timeout interval was reached or the callback function returned LAAW_CALLBACK_RETURN_END_WAIT. Check the value of LAAW_PARAMETERS.bCallbackEndedWait to determine which ended the wait.</li> </ul>

## WaitOnDialog

The **WaitOnDialog** function displays a custom dialog. You can write your script to handle different responses from the user based on the return value from this function.

### Syntax

```
WaitOnDialog ( szDlgName );
```

## Parameters

**Table 40** • WaitOnDialog Parameters

Parameter	Description
<b>szDlgName</b>	Specifies the ID of the dialog to display.

## Return Values

**Table 41** • WaitOnDialog Return Values

Return Value	Description
<b>dialog control ID</b>	The ID of the dialog control that received the WM_COMMAND message.
<b>IDCANCEL (2)</b>	This message is received as a signal that the dialog is about to close.
<b>DLG_ERR (-1)</b>	This message is received if any errors occurred.
<b>DLG_INIT (-100)</b>	This message is received immediately before the dialog is displayed.

## Additional Information

To enable end users to cancel the installation by clicking the close button in the upper-right corner of the InstallScript dialog, the dialog must have a button control whose Control Identifier property is set to 2. For more information, see [Using InstallScript to Implement Custom Dialogs](#).

## WaitOnDialog Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the DefineDialog, WaitOnDialog, EndDialog, and
* ReleaseDialog functions.
*
* This script opens a simple custom dialog that displays
* a bitmap. The dialog can be closed with any of three
* buttons: Back, Next, or Cancel.
*
* The "custom" dialog used in this script is actually the
* InstallShield Sd dialog that is displayed by the built-in
* function SdBitmap. Because this dialog is stored in
* the file _isres.dll, which is already compressed in
* the installation, it can be used in a script as a custom
* dialog.
*
```

```

* In order to use this dialog as a custom dialog, the
* script first defines it by calling DefineDialog. It then
* displays the dialog by calling WaitOnDialog. When an event
* ends dialog processing, EndDialog is called to close the
* dialog. Then the dialog is released from memory by
* a call to ReleaseDialog.
*
\*-----*/

// Dialog and control IDs.
#define RES_DIALOG_ID      12027 // ID of dialog itself
#define RES_PBUT_NEXT      1 // ID of Next button
#define RES_PBUT_CANCEL    9 // ID of Cancel button
#define RES_PBUT_BACK      12 // ID of Back button

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_WaitOnDialog(HWND);

function ExFn_WaitOnDialog(hMSI)
    STRING  szDialogName, szDLLName, szDialog;
    NUMBER  nDialog, nResult, nCmdValue;
    BOOL     bDone;
    HWND     hInstance, hwndParent, hwndDlg;
begin

    // Define the name of a dialog to pass as first
    // parameter to DefineDialog.
    szDialogName = "ExampleDialog";

    // DefineDialog's second parameter will be 0 because the
    // dll is in _isres.dll.
    hInstance = 0;

    // DefineDialog's third parameter will be null; installation
    // will search for the dialog in _isuser.dll and _isres.dll.
    szDLLName = "";

    // DefineDialog's fifth parameter will be null because the
    // dialog is identified by its ID in the fourth parameter.
    szDialog = "";

    // This value is reserved and must be 0.
    hwndParent = 0;

    // Define the dialog. The installation's main window will own
    // the dialog (indicated by HWND_INSTALL in parameter 7).
    nResult = DefineDialog (szDialogName, hInstance, szDLLName,
                           RES_DIALOG_ID, szDialog, hwndParent,
                           HWND_INSTALL, DLG_MSG_STANDARD|DLG_CENTERED);

    // Check for an error.
    if (nResult < 0) then
        MessageBox ("An error occurred while defining the dialog.", SEVERE);
        bDone = TRUE;
        abort;
    endif;

```

```

// Initialize the indicator used to control the while loop.
bDone = FALSE;

// Loop until done.
repeat

    // Display the dialog and return the next dialog event.
    nCmdValue = WaitOnDialog(szDialogName);

    // Respond to the event.
    switch (nCmdValue)
    case DLG_CLOSE:
        // The user clicked the window's Close button.
        Do (EXIT);
    case DLG_ERR:
        MessageBox ("Unable to display dialog. Setup canceled.", SEVERE);
        abort;
    case DLG_INIT:
        // Initialize the back, next, and cancel button enable/disable states
        // for this dialog and replace %P, %VS, %VI with
        // IFX_PRODUCT_DISPLAY_NAME, IFX_PRODUCT_DISPLAY_VERSION, and
        // IFX_INSTALLED_DISPLAY_VERSION, respectively, on control IDs 700-724
        // and 202.
        hwndDlg = CmdGetHwndDlg(szDialogName);
        SdGeneralInit(szDialogName, hwndDlg, 0, "");
    case RES_PBUT_CANCEL:
        // The user clicked the Cancel button.
        Do (EXIT);
    case RES_PBUT_NEXT:
        bDone = TRUE;
    case RES_PBUT_BACK:
        bDone = TRUE;
    endswitch;

until bDone;

// Close the dialog.
EndDialog (szDialogName);

// Free the dialog from memory.
ReleaseDialog (szDialogName);

end;

```

## Welcome



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

The **Welcome** function displays a dialog that welcomes the end user.

In a procedural script, you must call `SdProductName` before calling `Welcome` so that `InstallShield` can insert the product name into the first paragraph of message text in the `Welcome` dialog. In an event-based script, `SdProductName` is called automatically, with the `PRODUCT_NAME` string entry as its argument, before the `Begin` event. If you do not pass a product name using `SdProductName`, `InstallShield` cannot insert the product name but inserts extra spaces instead.

## Syntax

```
Welcome ( szTitle, nReserved );
```

## Parameters

**Table 42** • Welcome Parameters

Parameter	Description
<b>szTitle</b>	Specifies the title of this dialog. To display the default title <code>Welcome</code> , pass a null string ("" ) in this parameter.
<b>nReserved</b>	Pass 0 in this parameter.

## Return Values

**Table 43** • Welcome Return Values

Return Value	Description
<b>NEXT (1)</b>	Indicates that the end user clicked the <code>NEXT</code> button.
<b>BACK (12)</b>	Indicates that the end user clicked the <code>BACK</code> button.
<b>&lt; 0</b>	Indicates that <code>Welcome</code> failed to display the dialog.

## Additional Information

To view an example of this or other dialogs for your installation, use the `Dialog Sampler`. In `InstallShield`, on the `Tools` menu, point to `InstallScript`, then click `Standard Dialog Sampler` or `Skinned Dialog Sampler`.

## Welcome Example



**Project** • This information applies to the following project types:

- *InstallScript*
- *InstallScript MSI*

```
/*-----*\
 *
 * InstallShield Example Script
 *
 * Demonstrates the Welcome function.
```

```

*
* This script displays the installation Welcome dialog.
*
\*-----*/

#define PRODUCT "ExampleProduct"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

    export prototype ExFn_Welcome(HWND);

function ExFn_Welcome(hMSI)
    STRING svLogFile;
begin

    SdProductName ( PRODUCT );

    // Display the Welcome dialog.
    if (Welcome ("Welcome Dialog Example", 0) < 0) then
        MessageBox ("Welcome dialog failed.", SEVERE);
    endif;

end;

```

## WizardDirection



**Project** • This information applies to InstallScript projects.

The **WizardDirection** function is called in an object script to report the argument that was passed to the most recent call in the main setup's (or parent object's) script to the ShowObjWizardPages function or one of the object's ShowxxxxxUlyyyyy methods.

### Syntax

```
WizardDirection ( );
```

### Parameters

None.



## Return Values

**Table 44** • WizardDirection Return Values

Return Value	Description
<b>NEXT</b>	Indicates that NEXT was passed as the argument in the most recent call in the main setup's (or parent object's) script to the ShowObjWizardPages function or one of the object's ShowxxxxUlyyyy methods.
<b>BACK</b>	Indicates that BACK was passed as the argument in the most recent call in the main setup's (or parent object's) script to the ShowObjWizardPages function or one of the object's ShowxxxxUlyyyy methods.

## Additional Information

The purpose of WizardDirection is to report the direction in which the end user was moving through the dialog sequence at the most recent transition to the object's dialog sequence from the main setup's (or parent object's) dialog sequence. For more information, see [Creating the Object's Run-Time User Interface](#).

# WriteArrayProperty



**Project** • This information applies to *InstallScript* projects.

The **WriteArrayProperty** function is called in an object script to enter a value for a specified property whose value is an array.

## Syntax

```
WriteArrayProperty ( nPropertyBag, szPropertyName, ArrayPointer );
```

## Parameters

**Table 45** • WriteArrayProperty Parameters

Parameter	Description
<b>nPropertyBag</b>	Specifies a reference to the object's property bag object, in which property values are stored.
<b>szPropertyName</b>	Specifies the name of the property whose value you want to enter.
<b>ArrayPointer</b>	Specifies a pointer to the array property.

## Return Values

**Table 46** • WriteArrayProperty Return Values

Return Value	Description
<b>0</b>	This function always returns zero (0).

# WriteBoolProperty



**Project** • This information applies to *InstallScript* projects.

The **WriteBoolProperty** function is called in an object script to enter a value for a specified property whose value is a Boolean.

## Syntax

```
WriteBoolProperty ( nPropertyBag, szPropertyName, bPropertyValue );
```

## Parameters

**Table 47** • WriteBoolProperty Parameters

Parameter	Description
<b>nPropertyBag</b>	Specifies a reference to the object's property bag object, in which property values are stored.
<b>szPropertyName</b>	Specifies the name of the property whose value you want to enter.
<b>bPropertyValue</b>	Specifies the value of the property.

## Return Values

**Table 48** • WriteBoolProperty Return Values

Return Value	Description
<b>0</b>	This function always returns zero (0).

# WriteBytes

The **WriteBytes** function writes a specific number of bytes to a file opened in the binary mode. This function starts writing bytes at the current file pointer location.



**Note** • Before calling *WriteBytes*, you must open the file by calling *OpenFileMode(FILE\_MODE\_BINARY)* and then calling *OpenFile*.

The parameter *nIndex* is an index into *svString*; *nBytes* specifies how many bytes beyond the value of *nIndex* you want to write to the file. If *nIndex* plus *nBytes* exceeds the length of *svString*, InstallShield writes only the number of bytes from the index into the string to the end of the string. For example, if *svString* is 100 bytes long, *nIndex* is 50 and *nBytes* is 75, only the bytes between 51 and 100 (50 bytes instead of 75 bytes) are written to the file.

## Syntax

```
WriteBytes (nFile, svString, nIndex, nBytes);
```

## Parameters

**Table 49 • WriteBytes Parameters**

Parameter	Description
<b>nFile</b>	Specifies the file handle of a file that has been opened in binary mode.
<b>svString</b>	Specifies the string variable that contains the bytes to write to the output file.
<b>nIndex</b>	Specifies an index into svString. The bytes starting at this location are written to the output file. Note that the first byte is at index location 0.
<b>nBytes</b>	Specifies the number of bytes you want to write to the output file.

## Return Values

**Table 50 • WriteBytes Return Values**

Return Value	Description
<b>X</b>	Where X is the number of bytes actually written.
<b>&lt; 0</b>	Indicates that the function was unable to write the bytes.

## WriteBytes Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the WriteBytes function.
*
* WriteBytes is called to write a company name to a binary file.
*
* Note: Before running this script, set the preprocessor
*       constants so that they reference an existing directory
*       and file on the target system. Because the script will
*       write to this file -- overwriting any existing data --
*       you should create a file or make a copy of an existing
*       file for use with this example.
*
\*-----*/

#define EXAMPLE_DIR "C:\\\"
#define EXAMPLE_FILE "ISExempl.bin"

// Include Ifx.h for built-in InstallScript function prototypes.
```

```

#include "Ifx.h"

export prototype ExFn_WriteBytes(HWND);

function ExFn_WriteBytes(hMSI)
    STRING  szQuestion, svCompany[28];
    NUMBER  nFileHandle, nOffset, nIndex, nBytes;
begin

    // Set the file open mode.
    OpenFileMode (FILE_MODE_BINARY);

    // Open the file and get the file handle.
    if (OpenFile (nFileHandle, EXAMPLE_DIR, EXAMPLE_FILE) < 0) then
        MessageBox ("Unable to open the file.", SEVERE);
        abort;
    endif;

    // Ask user for his or her company name.
    szQuestion = "Please enter your company name. You may enter up to " +
        "twenty-seven characters.";
    AskText (szQuestion, "My Software Company", svCompany);

    // Move the file pointer 15 bytes from the start of the file.
    nOffset = 15;
    SeekBytes (nFileHandle, nOffset, FILE_BIN_START);

    // Write the company name to the file.
    nIndex = 0;
    nBytes = 27;
    if (WriteBytes (nFileHandle, svCompany, nIndex, nBytes) < 0) then
        MessageBox ("WriteBytes failed.", SEVERE);
    else
        MessageBox ("Bytes successfully written to file.", INFORMATION);
    endif;

    // Close the file.
    CloseFile (nFileHandle);

end;

```

## WriteLine

The **WriteLine** function writes a line of text to a text file opened in append mode. You must first set the file mode to append mode with **OpenFileMode**, and then either create the file with **CreateFile**, or open the file with **OpenFile**, before calling **WriteLine**. This function places the line at the end of the file.

**WriteLine** produces lines that have a carriage return and line feed character at the end of the line. To write to a binary file, use **WriteBytes**.



**Caution** • This function does not work with files opened in read-only mode.

## Syntax

```
WriteLine ( nvFileHandle, szLine );
```

## Parameters

**Table 51 • WriteLine Parameters**

Parameter	Description
<b>nvFileHandle</b>	Specifies the file handle of an open file. The handle is obtained from <b>OpenFile</b> or <b>CreateFile</b> .
<b>szLine</b>	Specifies a string that contains the text to write to the file.  <div data-bbox="628 657 665 703" data-label="Image"></div> <p><b>Note</b> • You can embed double quotation marks inside a string by delimiting the string with single quotation marks. For example, if you want to write the string "This string contains a double "quotation mark.", you can use the following code:</p> <pre>WriteLine(nvFileHandle, 'This string contains a double "quotation mark."');</pre> <div data-bbox="628 919 665 963" data-label="Image"></div> <p><b>Caution</b> • Do not attempt to write multiple lines by embedding newline characters in the string you pass in szLine. The following code writes the string as one line, with an unprintable character between "one" and "This":</p> <pre>szString = "This is line one\nThis is two"; WriteLine(nvFileHandle, szString);</pre> <p>To write two lines with one call to <b>WriteLine</b>, you must embed a return and a newline (in that order):</p> <pre>szString = "This is line one\r\nThis is two";</pre>

## Return Values

**Table 52 • WriteLine Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully wrote the line to the file.
<b>&lt; 0</b>	Indicates that the function was unable to write the line to the file.

## WriteLine Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```

/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the CreateFile and WriteLine functions.
*
* Createfile is called to create a file to store a string. The
* string is written into the file by the WriteLine function.
*
* Note: Before running this script, set the preprocessor
*       constant EXAMPLE_DIR so that it references an existing
*       directory on the target system. Note that if the file
*       specified by EXAMPLE_FILE already exists, it will be
*       overwritten.
*
\*-----*/

#define EXAMPLE_DIR "C:\\\"
#define EXAMPLE_FILE "ISExampl.txt"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_WriteLine(HWND);

function ExFn_WriteLine(hMSI)
    STRING  szTitle, szMsg;
    NUMBER  nvFileHandle;
begin

    // Set the file mode to append.
    OpenFileMode (FILE_MODE_APPEND);

    // Create a new file and leave it open.
    if (CreateFile (nvFileHandle, EXAMPLE_DIR, EXAMPLE_FILE) < 0) then
        // Report the error.
        MessageBox ("CreateFile failed.", SEVERE);
        abort;
    else
        // Set the message to write to the file.
        szMsg = "This line was appended by an example InstallShield script.";

        // Append the message to the file.
        if (WriteLine(nvFileHandle, szMsg) < 0) then
            // Report the error.
            MessageBox ("WriteLine failed.", SEVERE);
        else
            // Report success.
            szTitle = "CreateFile & WriteLine";
            szMsg = "Successfully created and wrote to %s.";
            sprintfBox (INFORMATION, szTitle, szMsg, EXAMPLE_FILE);
        endif;
    endif;

    // Close the file.
    CloseFile (nvFileHandle);

end;

```

# WriteNumberProperty



**Project** • This information applies to InstallScript projects.

The **WriteNumberProperty** function is called in an object script to enter a value for a specified property whose value is a number.

## Syntax

```
WriteNumberProperty ( nPropertyBag, szPropertyName, nPropertyValue );
```

## Parameters

**Table 53** • WriteNumberProperty Parameters

Parameter	Description
<b>nPropertyBag</b>	Specifies a reference to the object's property bag object, in which property values are stored.
<b>szPropertyName</b>	Specifies the name of the property whose value you want to enter.
<b>nPropertyValue</b>	Specifies the value of the property.

## Return Values

**Table 54** • WriteNumberProperty Return Values

Return Value	Description
<b>0</b>	This function always returns zero (0).

# WriteProfInt



**Tip** • All INI file changes should be created in the INI Files view of the IDE. Handling all of your INI file changes in this way allows for a clean uninstallation through the Windows Installer service.

The **WriteProfInt** function modifies an .ini file by inserting or updating a profile string that assigns an integer value to a key. Note the following important points:

- Because of the way in which Windows caches file changes, you should flush the cache buffer after calls to **WriteProfString**.
- Changes made to .ini files can be logged for uninstallation.
- To write a string value to an .ini file, call **WriteProfString** instead.
- Use the **AddProfString** and **ReplaceProfString** functions when you want to modify the **System.ini** file.




## Syntax

```
WriteProfInt ( szFileName, szSectionName, szKeyName, iValue );
```

## Parameters

**Table 55 • WriteProfInt Parameters**

Parameter	Description
<b>szFileName</b>	Specifies the name of the .ini file. If the file name is unqualified (that is, if a drive designation and path are <i>not</i> included), InstallShield searches for the file in the Windows folder. If the file does not exist, it is created in the specified folder; if a path is not included in file name, the file is created in the Windows folder. Note that if the file name is qualified with a path that does not exist, WriteProfInt will fail.
<b>szSectionName</b>	Specifies the name of the .ini file section in which szKeyName will be inserted or modified. The section name should <i>not</i> be enclosed within delimiting brackets ( [ ] ). The search for this name is <i>not</i> case sensitive.
<b>szKeyName</b>	Specifies the unique key to be updated. If a key that is to be updated does not exist in the specified section, it is created.   <b>Note</b> • To delete an entire section from an .ini file, pass a null string ("" ) in szKeyName. To delete one or more keys from a section in an .ini file, use <b>WriteProfString</b> .
<b>iValue</b>	Specifies an integer value to assign to the unique key identified by szKeyName.

## Return Values

**Table 56 • WriteProfInt Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully updated the specified .ini file.
<b>&lt; 0</b>	Indicates that the function was unable to update the specified .ini file.

## Additional Information

- The WriteProfInt function uses the Windows API WritePrivateProfileString to access the .ini file. Therefore, its functionality is limited by the functionality provided by the Windows API. Consult Microsoft documentation for more information on .ini files.
- Windows 95 and later cache .ini files, which can cause a delay in writing changes to the specified files. This in turn can interfere with subsequent file operations, such as calls to [CopyFile](#) and [XCopyFile](#). Therefore, you

should flush the cache buffer after using WriteProfInt if you are using file operations shortly afterward. Simply call WriteProfInt with null parameters to force Windows 95 or later to write the data to the .ini file immediately, as shown below:

```
WriteProfInt ("c:\\Test.ini", "Windows", "KeyboardDelay", 100);    WriteProfInt ("", "", "", 0); // null
string in first three parameters    //CopyFile should now have access to updated file.    CopyFile
("c:\\test.ini", "d:\\test.ini");
```

## WriteProfInt Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the WriteProfInt function.
*
* This script updates an integer value in an initialization file
* in the Windows directory.  If the file does not exist, it is
* created.
*
\*-----*/

// Define the initialization file name.
#define EXAMPLE_INI WINDIR^"ISExempl.ini"

// Define the initialization item and its new value.
#define SECTION "Windows"
#define KEYNAME "OurAppVal"
#define KEYVAL 0

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_WriteProfInt(HWND);

function ExFn_WriteProfInt(hMSI)
begin
    // Update a field in the initialization file.
    if (WriteProfInt (EXAMPLE_INI, SECTION, KEYNAME, KEYVAL) < 0 ) then
        // Report the error.
        sprintfBox (SEVERE, "WriteProfString",
                    "%s could not be updated", EXAMPLE_INI);
    else
        // Report success.
        sprintfBox (INFORMATION, "WriteProfString",
                    "%s was modified.", EXAMPLE_INI);
    endif;
end;
```

# WriteProfString

The **WriteProfString** function writes a profile string to an .ini file. Depending on the values passed to WriteProfString, it can create a section, delete an entire section, create a unique KEY=VALUE entry, delete a KEY=VALUE entry, or update a key's value.

Note the following important points:

- To write an integer value to an .ini file, call WriteProfInt instead.
- Use the AddProfString and ReplaceProfString functions when you want to modify the System.ini file.
- Changes made to .ini files can be logged for uninstallation. However, there are some important restrictions to be aware of. For more information, see Uninstalling Initialization (.ini) File Entries.
- WriteProfString uses the Windows API WritePrivateProfileString to access the .ini file. Therefore, its functionality is limited by the functionality provided by the Windows API. Consult Windows programming documentation for more information on .ini files.
- Windows caches .ini files, which can cause a delay in writing changes to the specified files. This in turn can interfere with subsequent file operations, such as calls to CopyFile and XCopyFile. Therefore, you should flush the cache buffer after using WriteProfString if you are using file operations shortly afterward. Simply call WriteProfString with null parameters to force Windows to write the data to the .ini file immediately:

```
WriteProfString ("C:\\Test.ini", "Windows", "KeyboardDelay", "100");
WriteProfString ("", "", "", ""); // null string for all four parameters
// CopyFile should now have access to updated file. CopyFile ("C:\\Test.ini", "C:\\Temp\\Test.ini");
```

## Syntax

```
WriteProfString ( szFileName, szSectionName, szKeyName, szValue );
```

## Parameters

**Table 57 • WriteProfString Parameters**

Parameter	Description
<b>szFileName</b>	Specifies the name of the .ini file. If the file name is unqualified (that is, if a drive designation and path are <i>not</i> included), InstallShield searches for the file in the Windows folder. If the file does not exist, it is created in the specified folder; if a path is not included in file name, the file is created in the Windows folder. Note that if the file name is qualified with a path that does not exist, WriteProfString will fail.
<b>szSectionName</b>	Specifies the name of the .ini file section to search for szKeyName. The section name should <i>not</i> be enclosed within delimiting brackets ( [ ] ). The search for this name is <i>not</i> case sensitive.
<b>szKeyName</b>	Specifies the unique key to be updated or deleted. If a key that is to be updated does not exist, it is created. To delete the key specified here, pass a null string ("" ) in szValue. To delete the entire section specified by szSectionName, including all entries within that, pass a null string ("" ) in this parameter.
<b>szValue</b>	Specifies the value to assign to the unique key identified by szKeyName. To delete the key, pass a null string ("" ) in this parameter. To delete the value of the key but retain the key itself, pass a blank space in this parameter.

## Return Values

**Table 58 • WriteProfString Return Values**

Return Value	Description
<b>0</b>	Indicates that the function successfully wrote the string to the specified .ini file.
<b>&lt; 0</b>	Indicates that the function was unable to write the string to the specified .ini file.

## WriteProfString Example



**Note •** To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
```

```

*
* Demonstrates the WriteProfString function.
*
* This script updates a field in an initialization file in the
* Windows directory. If the file does not exist, it is created.
*
\*-----*/

// Define the initialization file name.
#define EXAMPLE_INI WINDIR ^ "ISExempl.ini"

// Define the initialization item and its new value.
#define SECTION "Windows"
#define KEYNAME "Keyboard"
#define KEYVALUE "English"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_WriteProfString(HWND);

function ExFn_WriteProfString(hMSI)
begin
    // Update a field in the initialization file.
    if (WriteProfString (EXAMPLE_INI, SECTION, KEYNAME, KEYVALUE) < 0) then
        // Report the error.
        sprintfBox (SEVERE, "WriteProfString",
            "%s could not be updated", EXAMPLE_INI);
    else
        // Report success.
        sprintfBox (INFORMATION, "WriteProfString", "%s was modified.", EXAMPLE_INI);
    endif;
end;

```

## WriteStringProperty



**Project** • This information applies to InstallScript projects.

The **WriteStringProperty** function is called in an object script to enter a value for a specified property whose value is a string.

### Syntax

```
WriteStringProperty ( nPropertyBag, szPropertyName, szPropertyValue );
```

## Parameters

**Table 59** • WriteStringProperty Parameters

Parameter	Description
<b>nPropertyBag</b>	Specifies a reference to the object's property bag object, in which property values are stored.
<b>szPropertyName</b>	Specifies the name of the property whose value you want to enter.
<b>szPropertyValue</b>	Specifies the value of the property.

## Return Values

**Table 60** • WriteStringProperty Return Values

Return Value	Description
<b>0</b>	This function always returns zero (0).

# XCopyFile

The **XCopyFile** function copies one or more files from a source directory to a target directory. The function creates and logs the target directory if necessary. This function can copy subdirectories as well as files. **XCopyFile** creates subdirectories on the target directory if necessary when the constant `INCLUDE_SUBDIR` is passed in the parameter `nOp`.

If you use this function to transfer files to [WINSYSDIR64](#), you must first disable file system redirection using [WOW64FSREDIRECTION](#). Otherwise, files being transferred to `WINSYSDIR64` are incorrectly redirected to the 32-bit `SysWOW64` folder. Since some Windows functionality that could be used by the installation requires that redirection be enabled to work, Windows documentation recommends that you disable redirection only for as long as necessary. It is recommended that you then enable file system redirection as soon as you have completed transferring the necessary files to `WINSYSDIR64`. To learn more, see [Targeting 64-Bit Operating Systems with InstallScript Installations](#).

You cannot rename files using **XCopyFile**. To rename a file during a file copy operation, use the **CopyFile** function.



**Tip** • It is strongly recommended that you disable the Cancel button using the **Disable** function before calling the **XCopyFile** function if the status dialog is displayed during the copy. If you do not disable the Cancel button and the end user cancels during the copy file operation, the `OnCancelling` event handler is not called. Instead, the copy file operation returns a failure error code, which your script must handle by calling the appropriate event and then relaunching the copy file operation. You can enable and disable the Cancel button using the [Enable](#) and [Disable](#) functions.



---

**Note** • If you use unqualified file names and set values for `SRCDIR` and `TARGETDIR` when using **`XCopyFile`**, save the current values using [VarSave](#) before calling **`XCopyFile`** and then restore them using [VarRestore](#).

## Syntax

```
XCopyFile ( szSrcFile, szTargetPath, nOp );
```

## Parameters

**Table 61** • XCopyFile Parameters

Parameter	Description
<b>szSrcFile</b>	<p>Specify which files to copy. If the file specified is qualified—that is, if it includes a path—<b>XCopyFile</b> copies the file from the specified location. If szSrcFile contains an unqualified file name—without path information—<b>XCopyFile</b> copies from the folder that is identified by the system variable <a href="#">SRCDIR</a>. To copy multiple files, use wildcard characters in this parameter.</p> <p>You can specify a valid URL in this parameter. If you pass a CGI or ASP request (for example, "http://www.mydomain.net/login.asp?name=Me&amp;pwd=wow"), the response is sent to the file that is specified in the szTargetPath parameter. When passing a URL, do not include wildcard characters. To check the validity of a URL, call the following:</p> <pre>Is (VALID_PATH, szURL);</pre>
<b>szTargetPath</b>	<p>Specify the directory to which the files that are specified by szSrcFile should be copied. If this parameter contains a null string (""), the files are copied to the directory that is specified by the system variable TARGETDIR (in InstallScript installations) or INSTALLDIR (in Basic MSI and InstallScript MSI installations).</p> <p>You cannot specify a URL in this parameter. If you do, the function fails and returns ISERR_INVALID_ARG.</p>



**Table 61** • XCopyFile Parameters (cont.)

Parameter	Description
<b>nOp</b>	<p>Specify the type of copy operation to perform. Pass one of the following predefined constants in this parameter. To specify more than one option, combine constants with the OR ( ) operator.</p> <ul style="list-style-type: none"> <li>• <b>COMP_NORMAL</b>—Copies files to the target system, updating existing same-named files regardless of date, time, or version information.</li> <li>• <b>COMP_UPDATE_SAME</b>—Update the files even if the date, time, or version of the source file is identical to the target file. You must also specify either COMP_UPDATE_DATE or COMP_UPDATE_VERSION with this constant. Otherwise, this constant is ignored.</li> <li>• <b>COMP_UPDATE_DATE</b>—Updates the files based on the file date and time. This constant updates the file if the source file is newer than the target file.</li> <li>• <b>COMP_UPDATE_VERSION</b>—Updates the files based on the file version. This constant updates the file if the source file is newer than the target file. If the file version does not exist in both the source and the target files, date and time are used for comparison. If the file version does not exist for only one file, InstallShield treats the file containing version information as the newer file.</li> <li>• <b>SELFREGISTER</b>—Carries out the self-registration process immediately, when using the "non-batch method" of installing self-registering files.  If you have called <a href="#">Enable(SELFREGISTERBATCH)</a>, this option queues up self-registering files for registration. The files are registered once <a href="#">Do(SELFREGISTRATIONPROCESS)</a> is called, when using the "batch method" of installing self-registering files.  Always use SELFREGISTER together with the SHAREDFILE option, combining them with the bitwise OR operator ( ).</li> <li>• <b>SHAREDFILE</b>—Combines shared and locked file handling by causing <b>XCopyFile</b> to treat all files as shared, and to record locked .dll and .exe files for update when Windows or the system restarts. For more information, see <a href="#">RebootDialog</a> and <a href="#">SdFinishReboot</a>.  The SHAREDFILE option causes <b>XCopyFile</b> to treat all files as shared files and increment the registry reference counter by one when the file exists in the target directory and it has a reference count greater than 0. If the shared file does not exist in the target directory and it has no reference counter, the installation creates the counter and sets it to 1. If the shared file already exists in the target directory but has no reference counter, the installation creates the counter and initializes it to 2 as a precaution against accidental removal during uninstallation.</li> </ul>

**Table 61** • XCopyFile Parameters (cont.)

Parameter	Description
	<ul style="list-style-type: none"> <li>• <b>LOCKEDFILE</b>—Causes <b>XCopyFile</b> to record locked .dll and .exe files for update when Windows or the system is rebooted. A locked file is a file that is in use by an application or the system when the installation attempts to access or update the file. The LOCKEDFILE option works like SHAREDFILE except that LOCKEDFILE does not create registry entries or modify the registry reference counter. You cannot use the LOCKEDFILE option when using the SHAREDFILE option. There are some unshared files (such as shell extensions) for which the script writer does not want a registry entry and reference counter. These files should never be uninstalled, except by the application itself. LOCKEDFILE allows <b>XCopyFile</b> to handle locked files that are not shared.</li> <li>• <b>EXCLUDE_SUBDIR</b>—Specifies to exclude subdirectories contained in the source path. Note that empty subdirectories are not copied.</li> <li>• <b>INCLUDE_SUBDIR</b>—Specifies that subdirectories below the source path must also be copied.</li> <li>• <b>CLEAR_FILE_ATTR</b>—Specifies to clear the attributes of the copied file. When you use this parameter, the file in the target folder does not have any file attributes set by default. If you do not use this parameter, <b>XCopyFile</b> uses the attributes of the original file for the copied file.</li> </ul>

## Return Values

**Table 62** • XCopyFile Return Values

Return Value	Description
<b>0</b>	Indicates that the function successfully copied the files.
<b>ISERR_INVALID_ARG</b>	Indicates that an invalid argument was passed to the function.
<b>All other negative values</b>	<p>Indicates that the function was unable to copy the files.</p> <p>You can obtain the error message text associated with a large negative return value—for example, -2147024891 (0x80070005)—by calling <a href="#">FormatMessage</a>.</p>

## Additional Information

After you modify .ini files with **WriteProfString**, you must flush the cache buffer before using **XCopyFile**. All .ini files are cached; this behavior can cause a delay in writing changes to the specified files. This in turn can interfere with subsequent file operations. To avoid this problem, simply call **WriteProfString** with null parameters to force Windows to write the data to the .ini file immediately, as show below:

```
WriteProfString ("C:\\Test.ini", "Windows", "KeyboardDelay", "100");
// null string (") for all four parameters
WriteProfString ("", "", "", "");
// XCopyFile should now have access to updated file.
XCopyFile ("C:\\Test.ini", "C:\\Temp", EXCLUDE_SUBDIR);
```

## XCOPYFile Example



**Note** • To call this function in a Basic MSI setup, you must first create a custom action for the entry-point function, execute the custom action in a sequence or as the result of a dialog's control event, and then build the release.

```
/*-----*\
*
* InstallShield Example Script
*
* Demonstrates the XCOPYFile function.
*
* The first call to XCOPYFile copies all text files,
* regardless of date, time, or version.
*
* The second call copies program files and creates the
* subdirectories that these files need to be located in.
*
* The third call copies template files based upon the date,
* writing over target files that have the same or earlier date
* as the source files.
*
* The fourth call copies sample files based upon the version
* number, writing over target files that have an older version
* number.
*
* Note: In order for this script to run correctly, you must
*       set the preprocessor constants to a valid file name
*       and path on the target system.
*
\*-----*/

#define SDIR          "C:\\ISExmpl\\Source\\"
#define SDIR_PROGRAM "C:\\ISExmpl\\Source\\Program\\"
#define SDIR_TEMPLATE "C:\\ISExmpl\\Source\\Template\\"
#define SDIR_SAMPLES "C:\\ISExmpl\\Source\\Samples\\"
#define TDIR          "C:\\ISExmpl\\Target\\"

// Include Ifx.h for built-in InstallScript function prototypes.
#include "Ifx.h"

export prototype ExFn_XCOPYFile(HWND);

function ExFn_XCOPYFile(hMSI)
    STRING szSrcFile;
    NUMBER nResult;
begin

    // Set variable to filespec for source files.
    szSrcFile = "*.txt";

    // Copy all text files in the source directory
    // into the target directory.
    if (XCOPYFile (SDIR ^ szSrcFile, TDIR ^ ".*", COMP_NORMAL) < 0) then
        MessageBox ("XCOPYFile failed", SEVERE);
    else
        MessageBox ("Text files successfully copied.", INFORMATION);
    end if
end function
```

```

endif;

// Set new variables.
szSrcFile = "*..*";

// Copy all program files in a source subdirectory to
// a subdirectory of the target directory.

if (XCopyFile (SDIR_PROGRAM ^ szSrcFile, TDIR ^ "PROGRAM" ^ "*..*",
    INCLUDE_SUBDIR) < 0) then
    MessageBox ("XCopyFile failed", SEVERE);
else
    MessageBox ("Program files successfully copied.", INFORMATION);
endif;

// Copy all template files in a source subdirectory
// to a subdirectory of the target directory.
if (XCopyFile (SDIR_TEMPLATE ^ szSrcFile, TDIR ^ "TEMPLATE" ^ "*..*",
    COMP_UPDATE_SAME | COMP_UPDATE_DATE) < 0) then
    MessageBox ("XCopyFile failed", SEVERE);
else
    MessageBox ("Template files successfully copied.", INFORMATION);
endif;

// Copy all sample files within a source subdirectory
// to a subdirectory of the target directory.
if (XCopyFile (SDIR_SAMPLES ^ szSrcFile, TDIR ^ "SAMPLES" ^ "*..*",
    COMP_UPDATE_VERSION) < 0) then
    MessageBox ("XCopyFile failed", SEVERE);
else
    MessageBox ("Sample files successfully copied.", INFORMATION);
endif;

end;

```

# Index

## Symbols

`^` operator 498, 501, 508  
`__FILE__` 279  
`__LINE__` 279  
`_FONTFILEINFO` 289  
`_isres.dll` 336  
`_isuser.dll` 805  
`_MAX_PATH` 278  
`!` operator 504  
`!=` operator 506  
`.` operator 505  
`.swf` 1170  
`@` operator 294, 508  
`*` operator 498, 504  
`/` operator 498  
`&` operator 299, 501  
`&&` operator 504  
`#define` 366  
`#elif` 367  
`#error` 367  
`#if...#else...#endif` 367  
`#ifdef` 368  
`#ifndef` 368  
`#include` 369  
`#undef` 370  
`#warning` 370  
`%` operator 508  
`%` sign 62  
`+` operator 498, 503, 508  
`<` operator 506  
`<<` operator 501  
`<=` operator 506  
`-` operator 498  
`=` operator 506

`->` operator 509  
`>` operator 506  
`>=` operator 506  
`>>` operator 501  
`|` operator 501  
`||` operator 504  
`~` operator 501

## A

`abort` 65  
Accelerator keys 1007  
    `AskOptions` 534  
    `Handler` 1007  
`AddFolderIcon` 519  
`AddFolderIcon Example` 522, 523, 525  
Adding 527  
    configuration file statements 587  
    elements to a list 1039  
    environment variables 547  
    features 810  
    lines to initialization files 527  
    path to batch file 786  
    path to search path 1144  
    strings to batch files 789  
`AddProfString` 527  
`AddProfString Example` 529  
`ADDREMOVE` 312  
`ADDREMOVE_COMBINEDBUTTON` 312  
`ADDREMOVE_HIDECHANGEOPTION` 313  
`ADDREMOVE_HIDEREMOVEOPTION` 313  
`ADDREMOVE_STRING_REMOVEONLY` 313  
`ADDREMOVE_SYSTEMCOMPONENT` 313  
Address operator 497  
`AdminAskPath` 530

- AdminAskPath example 531
- ADMINUSER, InstallScript variable 319
- Advanced 439
  - batch file functions 437
  - configuration file functions 439
- Advanced Event Handlers 429
- Advanced UI interaction InstallScript functions 483
- AFTER 79, 1043
- After Data Move Handlers 400
- ALLCONTENTS 80, 722
- ALLCONTROLS 80, 690
- ALLUSERS
  - InstallScript variable 314
- And operator 504
- APPEND 80, 922
- Append to path operator 498
- Application 1026
  - launching 453
- Arithmetic operators 498
- Arrays 306
  - character 306
  - data type 294
- ASCII characters
  - displaying with escape sequences 60
- ASKDESTPATH 80
- AskDestPath 531
- AskDestPath Example 533
- ASKOPTIONS 80
- AskOptions 534
- AskOptions Example 537
- ASKPATH 81
- AskPath 539
- AskPath Example 541
- ASKTEXT 81
- AskText 542
- AskText Example 543
- AskYesNo 544
- AskYesNo Example 546
- Assignment operator 500
- Attributes 142
  - file 142
- Autoexec.bat 786
- Autosizing strings 306
- AVI files 1170

## B

- BACK 81, 531
- BACKBUTTON 82, 754
- BACKGROUND 82, 754
- BACKGROUNDCAPTION 83, 1484
- Backslash character 1584
- BASEMEMORY 83
- BASICMSI script variable 280
- Batch files 553
  - adding paths 786
  - adding strings 789
  - advanced batch file functions 437
  - backup copy 559
  - default batch file name 570
  - deleting lines 553
  - Ez batch file functions 437
  - finding reference keys 562
  - loading into memory 555
  - moving a line 567
  - replacing text 793
  - saving after edit 559
- BATCH\_INSTALL
  - testing 1659
- BatchAdd 547
- BatchAdd Example 550
- BatchDeleteEx 553
- BatchDeleteEx Example 554
- BatchFileLoad 555
- BatchFileLoad Example 557
- BatchFileSave 334
- BatchFileSave Example 561
- BatchFind 562
- BatchFind Example 564
- BatchGetFileName 565
- BatchGetFileName Example 566
- BatchMoveEx 567
- BatchMoveEx Example 569
- BatchSetFileName 570
- BatchSetFileName Example 571
- Beep 1126
- BEFORE 83, 1043
- Before Data Move Handlers 381
- BIF\_BROWSEFORCOMPUTER 84
- BIF\_BROWSEFORPRINTER 84
- BIF\_DONTGOBELOWDOMAIN 84
- BIF\_EDITBOX 84
- BIF\_RETURNFSANCESTORS 85
- BIF\_RETURNONLYFSDIRS 85
- BIF\_STATUSTEXT 85
- BILLBOARD 85, 1166
- billboards
  - in InstallScript and InstallScript MSI projects
    - disabling 754
    - moving 1166
    - resizing 1514
    - special effects (SetDisplayEffect) 1454
    - specifying size 1514
- Binary arithmetic operators 499
- BINARY data type 289
- Binary files 626
  - closing 579
  - creating 626

- file pointer 1426
- opening 1133
- random access 1426
- reading 1185
- seeking 1426
- writing 1675
- Bit operators 501
- BITMAPICON 86, 1160
- Bitmaps 1454
  - displaying in dialog 735
  - placing 1160
  - setting display effects 1454
- BK\_BLUE 86, 1448
- BK\_GREEN 86, 1448
- BK\_MAGENTA 86, 1448
- BK\_ORANGE 86, 1448
- BK\_PINK 87, 1448
- BK\_RED 87, 1448
- BK\_SMOOTH 87, 1448
- BK\_SOLIDBLACK 87, 1448
- BK\_SOLIDBLUE 87, 1448
- BK\_SOLIDGREEN 88, 1448
- BK\_SOLIDMAGENTA 88, 1448
- BK\_SOLIDORANGE 88, 1448
- BK\_SOLIDPINK 88, 1448
- BK\_SOLIDRED 88, 1448
- BK\_SOLIDWHITE 89, 1448
- BK\_SOLIDYELLOW 89, 1448
- BK\_YELLOW 89, 1448
- BLACK 89, 1484
- BLUE 89, 1484
- BOOL 65
  - data type 289
- Boolean operators 365
- Boot drive 988
- BOOTUPDRIVE 90, 988
- Built-in dialogs 441
- Built-in functions 433
  - by category 435
- BUTTON\_CHECKED 90, 668
- BUTTON\_UNCHECKED 90, 668
- Buttons 534
  - AskOptions 534
- BYREF operator 502
- BYTES 90, 952
- Bytes 1675
- BYTES, KBYTES, MBYTES 988
- Bytes, reading from file 1185
  - copying from strings 616
  - seeking in file 1426
  - writing to file 1675
- BYVAL Operator 503

## C

- CalculateAndAddFileCost 572
- CallDLLFx 573
- CallDLLFx Example 574
- Calling 573
- CANCEL 91, 1429
- CANCELBUTTON 91
- Carriage return 60
- Case 1590
- case 73
  - changing 1590
- cdecl keyword 65
- CD-ROM 1135
- CDROM 91, 988
- CDROM\_DRIVE 91, 1001
- CENTERED 91, 1160
- ChangeDirectory 575
- ChangeDirectory Example 576
- Changing 575
  - elements in a list 1088
  - folder 575
- char data type 289
- CharReplace 577
- CharReplace Example 578
- Check 988
- Check boxes 534
  - AskOptions 534
- CHECKBOX 92, 735, 741
- CHECKBOX95 92, 741
  - CD-ROM 988
  - CPU 988
  - date 988
  - drive type 988
  - feature selection 866
  - for disk drive 784
  - for folder 783
  - language 988
- CHECKBOX95 dialog style 742
- CHECKLINE 92, 741
- CHECKLINE dialog style 744
- CHECKMARK 92, 741
  - memory 988
  - operating system 988
  - ports 988
  - time 988
  - video 988
  - Windows version 988
- CHECKMARK dialog style 743
- Class object 1200
- CLEAR\_FILE\_ATTR 92
- CloseFile 579
- CloseFile Example 580
- Closing 579

- custom dialogs 774
  - files 579
- CmdGetHwndDlg 581
- CmdGetHwndDlg Example 582
- CMDLINE 320
- CoCreateObject 584
- CoCreateObjectDotNet 585
- CoGetObject 585
- CoGetObject Example 586
- Color 988
  - colors available 988
- COLORS 93, 988
  - setting background and status bar 1448
  - setting custom colors 1274
- COM objects 584, 585, 765
  - unloading application domains 767
- Combine strings 503
- COMMAND 93, 553
- Command-Line Compiler 53
- Comments 57
- COMMON 93, 1175
- COMMONFILES 321
- COMMONFILES64 322
- COMP\_NORMAL 94
- COMP\_UPDATE\_DATE 94, 1686
- COMP\_UPDATE\_SAME 95, 1686
- COMP\_UPDATE\_VERSION 95, 1686
- COMPACT 93, 1392
- Company name 1012
- COMPARE\_DATE 93, 912
- COMPARE\_MD5\_SIGNATURE 94
- COMPARE\_SIZE 94, 912
- COMPARE\_VERSION 94, 912
- Comparing 1638
  - files 912
  - strings 1571
  - versions 1638
- Compiler
  - IDE 52
  - InstallScript limits for 55
  - preprocessor directives for 365
- Component Event Handlers 378
- Component functions 438
- ComponentAddItem 438
- ComponentCompareSizeRequired 438
- ComponentDialog 438
- ComponentError 438
- ComponentErrorInfo 438
- ComponentFileEnum 438
- ComponentFileInfo 438
- ComponentFilterLanguage 438
- ComponentFilterOS 438
- ComponentGetData 438
- ComponentGetItemSize 438
- ComponentGetTotalCost 438
- ComponentInitialize 438
- ComponentIsItemSelected 438
- ComponentListItems 438
- ComponentLoadTarget 438
- ComponentMoveData 438
- ComponentReinstall 438
- ComponentRemoveAll 438
- ComponentRemoveAllInLogOnly 438
- ComponentRemoveAllInMedia 438
- ComponentRemoveAllInMediaAndLog 438
- ComponentSaveTarget 438
- ComponentSelectItem 438
- ComponentSelectNew 438
- ComponentSetData 438
- ComponentSetTarget 438
- ComponentSetupTypeEnum 438
- ComponentSetupTypeGetData 438
- ComponentSetupTypeSet 438
- ComponentTotalSize 438
- ComponentTransferData 438
- ComponentUpdate 438
- ComponentValidate 438
- Concatenate strings 503
- Config.sys 603
- ConfigAdd 587
- ConfigAdd Example 589
- ConfigDelete 590
- ConfigDelete Example 591
- ConfigFileLoad 592
- ConfigFileLoad Example 593
- ConfigFileSave 595
- ConfigFileSave Example 596
- ConfigFind 598
- ConfigFind Example 600
- ConfigGetFileName 601
- ConfigGetFileName Example 602
- ConfigGetInt 603
- ConfigGetInt Example 604
- ConfigMove 606
- ConfigMove Example 607
- ConfigSetFileName 609
- ConfigSetFileName Example 610
- ConfigSetInt 611
- ConfigSetInt Example 612
- Configuration files 795
  - adding a device driver 795
  - adding statements 587
  - adding strings 798
  - advanced configuration file functions 439
  - backup copy 595
  - deleting statements 590
  - Ez configuration file functions 439
  - finding a reference key 598



- getting default configuration file name 601
- getting integer values 603
- getting numeric parameter values 801
- loading into memory 592
- moving lines 606
- saving 595
- setting default file name 609
- setting integer values 611
- setting values 803
- Constants
  - defining 56
  - predefined 79
  - user defined 294
- CONTINUE 95, 1150
- Converting 1595
  - lower to upper case 1595
  - number to string 1131
  - string to number 1592
  - unit constant to UI string 1573
  - upper to lower case 1590
- ConvertSizeToUnits 614
- ConvertWinHighLowSizeToISHighLowSize 616
- COPY\_ERR\_CREATEDIR 95, 620
- COPY\_ERR\_MEMORY 96, 1686
- COPY\_ERR\_NODISKSPACE 96, 620
- COPY\_ERR\_OPENINPUT 96, 620
- COPY\_ERR\_OPENOUTPUT 96, 620
- COPY\_ERR\_TARGETREADONLY 96, 1686
- CopyBytes 616
- CopyBytes Example 618
- CopyCharArrayToSStringArray 619
- CopyFile 620
- CopyFile Example 623
- Copying 616
  - files 620
  - substrings 616
- CPU 97, 988
- CreateDir 624
- CreateDir Example 625
- CreateFile 626
- CreateFile Example 628
- CreateInstallationInfo 629
- CreateObject 630
- CreateProgramFolder 631
- CreateProgramFolder Example 632
- CreateRegistrySet 633
- CreateRegistrySet Example 634
- CreateShellObjects 635
- CreateShellObjects Example 636
- CreateShortcut 637
  - example 1 643
  - example 2 644
  - example 3 645
  - example scenarios 642
- CreateShortcutFolder 647
  - example 647
- Creating 624
  - files 626
  - folders 624
  - program folders 631
- Critical files 855
- CS\_OPTION\_FLAG\_NO\_NEW\_INSTALL\_HIGHLIGHT 97
- CS\_OPTION\_FLAG\_NO\_STARTSCREEN\_PIN 97
- CS\_OPTION\_FLAG\_PREVENT\_PINNING 97
- CS\_OPTION\_FLAG\_REPLACE\_EXISTING 98
- CS\_OPTION\_FLAG\_RUN\_MAXIMIZED 98
- CS\_OPTION\_FLAG\_RUN\_MINIMIZED 98
- CtrlClear 648
- CtrlClear Example 649
- CtrlDir 652
- CtrlDir Example 653
- CtrlGetCurSel 656
- CtrlGetCurSel Example 656
- CtrlGetDlgItem 659
- CtrlGetMLEText 660
- CtrlGetMLEText Example 661
- CtrlGetMultCurSel 664
- CtrlGetMultCurSel Example 665
- CtrlGetState 668
- CtrlGetState Example 669
- CtrlGetSubCommand 672
- CtrlGetSubCommand Example 672
- CtrlGetText 675
- CtrlGetText Example 676
- CtrlGetUrlForLinkClicked 678
- CtrlGetUrlForLinkClicked example 679
- CtrlPGroups 681
- CtrlPGroups Example 681
- CtrlSelectText 684
- CtrlSelectText Example 684
- CtrlSetCurSel 686
- CtrlSetCurSel Example 687
- CtrlSetFont 690
- CtrlSetFont Example 691
- CtrlSetList 694
- CtrlSetList Example 695
- CtrlSetMLEText 699
- CtrlSetMLEText Example 700
- CtrlSetMultCurSel 703
- CtrlSetMultCurSel Example 704
- CtrlSetState 707
- CtrlSetState Example 708
- CtrlSetText 711
- CtrlSetText Example 712
- CURRENTROOTKEY 98
- CUSTOM 98, 1392
- Custom dialogs 668
  - check box controls 668

- clearing contents 648
- closing 774
- creating section names 1365
- defining 805
- displaying 1667
- displaying file names 652
- displaying program folders 681
- fonts 690
- freeing from memory 1256
- getting commands 672
- getting selected items 656
- multi-line edit control 660
- multi-selection list box controls 703
- radio button controls 668
- registering in a script 714
- response file 1508
- retrieving window handles 581
- setting list controls 694
- text controls 711
- Custom handler 1007

## D

- Data 56
- Data structures 509
  - member operator 505
  - overview 295
  - pointer to 299
  - structure pointer operator 509
- DATA\_COMPONENT 99, 1508
- DATA\_LIST 99, 1508
- DATA\_NUMBER 99, 1502
- DATA\_STRING 99, 1502
  - declaring 56
- DATE 100, 988
- Declaring 56
  - functions 56
  - variables 56
- DEFAULT 100
- default 73
- Default batch file 570
- DefineDialog 714
- DefineDialog Example 718
- Defining 56
  - custom dialogs 714
- DEFWINDOWMODE 100, 769
- DeinstallSetReference 720
- DeinstallStart 720
- Delay 721
- Delay Example 721
- DELETE 100
- DELETE\_EOF 100, 916
- DeleteCharArray 722
- DeleteDir 722
- DeleteDir Example 723
- DeleteFile 724
- DeleteFile Example 725
- DeleteFolderIcon 726
- DeleteFolderIcon Example 727
- DeleteProgramFolder 728
- DeleteProgramFolder Example 729
- DeleteShortcut 730
  - example 731
- DeleteShortcutFolder 732
  - example 733
- DeleteWCharArray 734
- Deleting 722
  - directories 722
  - elements from a list 1058
  - files 724
  - folder item 726
  - folders 722
  - lines 916
  - path from search path 1147
  - program folders 728
  - shortcut folders through InstallScript 732
  - shortcut through InstallScript 730
- Desktop folder 324
- Destination folder 1277
- Device Drivers
  - InstallScript functions 440
- Device drivers 795
  - installing into Config.sys 795
- Dialog styles 741
- DIALOGCACHE 101, 769
- Dialogs
  - built-in 441
  - confirming folder selection 1291
  - confirming registration 1294
  - custom 707
  - custom dialog 1399
  - displaying bitmaps 1288
  - displaying file modifications 1409
  - displaying help topics 1307
  - feature 819
  - finishing setup 1331
  - general 1401
  - getting a destination path 1429
  - getting text 542
  - getting user options 534
  - getting yes/no input 544
  - InstallScript 441
    - functions for customizing 450
  - license agreement 1346, 1356
  - messages 1414
  - product name 1378
  - prompting for next distribution disk 777
  - prompting to reboot 1189

- registering the user 1379
  - selecting a destination folder 1277
  - selecting features 1312
  - selecting folders 1436
  - selecting program folders 1388
  - selecting setup types 1397
  - setting elements in SD dialogs 735
  - setting titles 1452
  - start file transfer 1416
  - styles 741
  - welcome message 1670
- DialogSetFont 734
- DialogSetInfo 735
- DialogSetInfo Example 740
- DIFXAPI\_ERROR 101
- DIFXAPI\_INFO 101
- DIFXAPI\_SUCCESS 101
- DIFXAPI\_WARNING 101
- DIFxDriverPackageGetPath 744
- DIFxDriverPackageInstall 745
- DIFxDriverPackagePreinstall 749
- DIFxDriverPackageUninstall 751
- DIR\_WRITEABLE 102, 1012
- Directives, compiler 365
- DIRECTORY 102, 1139
  - changing 575
  - checking 783
  - creating 624
  - deleting 722
  - finding 925
  - InstallScript functions 459
  - searching 927
- Directory 459
- Disable 754
- Disable Example 758
- DISABLE\_ALLUSERBTN 102
- DISABLE\_PERUSERBTN 102
- DISK 103, 1139
- Disk 950
- DISK\_INFO\_QUERY\_ALL 103
- DISK\_INFO\_QUERY\_BYTES\_PER\_CLUSTER 104
- DISK\_INFO\_QUERY\_DISK\_FREE\_SPACE 104
- DISK\_INFO\_QUERY\_DISK\_TOTAL\_SPACE 104
- DISK\_INFO\_QUERY\_DRIVE\_TYPE 104
- DISK\_TOTALSPACE 104, 988
- DISK\_TOTALSPACE\_EX 105, 988
- DISK1COMPONENT 103
- DISK1SETUPEXENAME 322
- DISK1TARGET 322
  - boot drive 988
  - drive 784
  - drive designation 945
  - drive type 988
  - free space 952
  - total space 988
  - valid drive list 1001
  - volume label 988
- Displaying 1378
  - ASCII characters with escape sequences 60
  - product name 1378
  - program folder 1497
- DLG\_ASK\_OPTIONS 105, 1452
- DLG\_ASK\_PATH 105, 1452
- DLG\_ASK\_TEXT 105, 1452
- DLG\_ASK\_YESNO 105, 1452
- DLG\_CENTERED 106, 714
- DLG\_CLOSE 106, 774
- DLG\_DIR\_DIRECTORY 106, 652
- DLG\_DIR\_DRIVE 106, 652
- DLG\_DIR\_FILE 106, 652
- DLG\_ENTER\_DISK 107, 1452
- DLG\_ERR 107, 1667
- DLG\_ERR\_ALREADY\_EXISTS 107, 714
- DLG\_ERR\_ENDDL 107, 1256
- DLG\_INFO\_ALTIMAGE 107, 735
- DLG\_INFO\_ALTIMAGE\_HIDPI 108
- DLG\_INFO\_ALTIMAGE\_REVERT\_IMAGE 108, 735
- DLG\_INFO\_ALTIMAGE\_VERIFY\_BMP 108, 735
- DLG\_INFO\_CHECKSELECTION 108, 735
- DLG\_INFO\_KUNITS 108, 735
- DLG\_INFO\_USEDECIMAL 109, 735
- DLG\_INIT 109, 690
- DLG\_INIT routine 581
- DLG\_MSG\_ALL 109, 714
- DLG\_MSG\_INFORMATION 109, 1452
- DLG\_MSG\_SEVERE 109, 1452
- DLG\_MSG\_STANDARD 110, 714
- DLG\_MSG\_WARNING 110, 1452
- DLG\_STATUS 110, 1452
- DLG\_USER\_CAPTION 110, 1452
- DLLs 714
  - calling functions 573
  - custom dialogs 714
  - InstallScript functions 453
  - loading into memory 1625
  - unloading from memory 1621
- Do 759
- Do Example 760
- DoInstall 761
- DoInstall Example 764
- DOINSTALL\_OPTION\_NOHIDEPROGRESS 110
- DOINSTALL\_OPTION\_NOHIDESPLASH 111
- DOINSTALL\_OPTION\_NOLANGSWITCH 111
- DOINSTALL\_OPTION\_NOSETBATCHINSTALL 111
- DotNetCoCreateObject 765
- DOTNETFRAMEWORKINSTALLED 111
- DOTNETSERVICEPACKINSTALLED 111
- DotNetUnloadAppDomain 767

[downto](#) 66  
[DRIVE](#) 112, 988  
[DRIVE\\_CDROM](#) 112  
[DRIVE\\_FIXED](#) 112  
[DRIVE\\_NO\\_ROOT\\_DIR](#) 112  
[DRIVE\\_RAMDISK](#) 112  
[DRIVE\\_REMOTE](#) 113  
[DRIVE\\_REMOVABLE](#) 113  
[DRIVE\\_UNKNOWN](#) 113  
[DRIVER\\_PACKAGE\\_DELETE\\_FILES](#) 113  
[DRIVER\\_PACKAGE\\_FORCE](#) 113  
[DRIVER\\_PACKAGE\\_LEGACY\\_MODE](#) 114  
[DRIVER\\_PACKAGE\\_ONLY\\_IF\\_DEVICE\\_PRESENT](#) 114  
[DRIVER\\_PACKAGE\\_REPAIR](#) 114  
[DRIVER\\_PACKAGE\\_SILENT](#) 114

## E

[EDITBOX\\_CHANGE](#) 114, 672  
[EFF\\_BOXSTRIPE](#) 115, 1454  
[EFF\\_FADE](#) 115, 1454  
[EFF\\_HORZREVEAL](#) 115, 1454  
[EFF\\_HORZSTRIPE](#) 115, 1454  
[EFF\\_NONE](#) 115, 1454  
[EFF\\_REVEAL](#) 116, 1454  
[EFF\\_VERTSTRIPE](#) 116, 1454  
[else](#) 69  
[elseif](#) 70  
[Enable](#) 769  
[Enable Example](#) 772  
[ENABLED\\_ISERVICES](#) 322  
[END\\_OF\\_FILE](#) 116, 919  
[END\\_OF\\_LIST](#) 116, 1088  
[EndCurrentDialog](#) 773  
[EndDialog](#) 774  
[EndDialog Example](#) 775  
[endfor](#) 66  
[endif](#) 70  
[endprogram](#) 56  
[endswitch](#) 73  
[endwhile](#) 75  
[ENGINECOMMONDIR](#) 322  
[ENGINEDIR](#) 323  
[ENTERDISK](#) 117  
[EnterDisk](#) 777  
[EnterDisk Example](#) 778  
[EnterDiskError](#) 779  
[EnterLoginInfo](#) 780  
[EnterPassword](#) 782  
[Enumerating setup types](#) 892  
[Environment space](#) 988  
[Environment variables](#) 1223
 

- [adding and modifying](#) 547
- [getting](#) 954
- [registry](#) 1223

[EQUALS](#) 117, 1638  
[Err Object](#) 511  
[ERR\\_ABORT](#) 123  
[ERR\\_BOX\\_BADPATH](#) 124, 1457  
[ERR\\_BOX\\_BADTAGFILE](#) 124, 1457  
[ERR\\_BOX\\_DISKID](#) 124, 1457  
[ERR\\_BOX\\_DRIVEOPEN](#) 124, 1457  
[ERR\\_IGNORE](#) 124  
[ERR\\_NO](#) 125  
[ERR\\_PERFORM\\_AFTER\\_REBOOT](#) 125  
[ERR\\_RETRY](#) 125  
[ERR\\_YES](#) 125  
[ERROR\\_ACCESS\\_DENIED](#) 117  
[ERROR\\_CIRCULAR\\_DEPENDENCY](#) 118  
[ERROR\\_DATABASE\\_DOES\\_NOT\\_EXIST](#) 118  
[ERROR\\_DEPENDENT\\_SERVICES\\_RUNNING](#) 118  
[ERROR\\_DUP\\_NAME](#) 118  
[ERROR\\_FILE\\_NOT\\_FOUND](#) 119  
[ERROR\\_INVALID\\_HANDLE](#) 119  
[ERROR\\_INVALID\\_PARAMETER](#) 119  
[ERROR\\_INVALID\\_SERVICE\\_ACCOUNT](#) 119  
[ERROR\\_INVALID\\_SERVICE\\_CONTROL](#) 120  
[ERROR\\_PATH\\_NOT\\_FOUND](#) 120  
[ERROR\\_SERVICE\\_ALREADY\\_RUNNING](#) 120  
[ERROR\\_SERVICE\\_CANNOT\\_ACCEPT\\_CTRL](#) 120  
[ERROR\\_SERVICE\\_DATABASE\\_LOCKED](#) 121  
[ERROR\\_SERVICE\\_DEPENDENCY\\_DELETED](#) 121  
[ERROR\\_SERVICE\\_DEPENDENCY\\_FAIL](#) 121  
[ERROR\\_SERVICE\\_DISABLED](#) 121  
[ERROR\\_SERVICE\\_DOES\\_NOT\\_EXIST](#) 122  
[ERROR\\_SERVICE\\_EXISTS](#) 122  
[ERROR\\_SERVICE\\_LOGON\\_FAILED](#) 122  
[ERROR\\_SERVICE\\_NO\\_THREAD](#) 123  
[ERROR\\_SERVICE\\_NOT\\_ACTIVE](#) 122  
[ERROR\\_SERVICE\\_REQUEST\\_TIMEOUT](#) 123  
[ERROR\\_TIMEOUT](#) 123  
[ERRORFILENAME](#) 323  
[ErrorInfo.Feature](#) 413  
[ErrorInfo.Feature.Description](#) 413  
[ErrorInfo.Feature.DisplayName](#) 413  
[ErrorInfo.Feature.Name](#) 413  
[ErrorInfo.FileError.Description](#) 413  
[ErrorInfo.FileError.File](#) 413  
[ErrorInfo.FileGroup](#) 413  
[ErrorInfo.LastError](#) 413  
[Errors](#) 823
 

- [getting information](#) 956
- [run-time](#) 823
- [setting information](#) 1461
- [user-defined](#) 367

[Escape sequences](#) 62  
[Event Handlers](#) 371, 373  
[Event handlers](#) 1007

EXCLUDE\_SUBDIR 125, 925  
 EXCLUSIVE 126, 1281  
 Executing a program from a setup script 1025  
 Existence, check for specific program item or subfolder 1177  
 EXISTS 126, 784  
 ExistsDir 783  
 ExistsDir Example 783  
 ExistsDisk 784  
     file 1012  
     program item 1177  
     program subfolder 1177  
     registry key 1228  
 ExistsDisk Example 785  
 EXIT 126, 1007  
 Exit handler 759  
 exit keyword 65  
 export keyword 66  
 EXTENDEDMEMORY 126, 988  
 Extensibility functions 453  
 EXTENSION\_ONLY 127, 1139  
 external keyword 66  
 Ez configuration file functions 439  
 EzBatchAddPath 786  
 EzBatchAddPath Example 787  
 EzBatchAddString 789  
 EzBatchAddString Example 791  
 EzBatchReplace 793  
 EzBatchReplace Example 794  
 EzConfigAddDriver 795  
 EzConfigAddDriver Example 796  
 EzConfigAddString 798  
 EzConfigAddString Example 799  
 EzConfigGetValue 801  
 EzConfigGetValue Example 802  
 EzConfigSetValue 803  
 EzConfigSetValue Example 804  
 EzDefineDialog 805  
 EzDefineDialog Example 807

## F

FALSE 127, 1114  
 Feature event handlers 404  
 Feature Functions 454  
 FEATURE\_FIELD\_CDROM\_FOLDER 128  
 FEATURE\_FIELD\_DESCRIPTION 128  
 FEATURE\_FIELD\_DISPLAYNAME 128  
 FEATURE\_FIELD\_ENCRYPT 129  
 FEATURE\_FIELD\_FILENEED 129  
 FEATURE\_FIELD\_FTPLOCATION 129  
 FEATURE\_FIELD\_GUID 130  
 FEATURE\_FIELD\_HANDLER\_ONINSTALLED 130  
 FEATURE\_FIELD\_HANDLER\_ONINSTALLING 130  
 FEATURE\_FIELD\_HANDLER\_ONUNINSTALLED 131  
 FEATURE\_FIELD\_HANDLER\_ONUNINSTALLING 131  
 FEATURE\_FIELD\_HTTPLOCATION 131  
 FEATURE\_FIELD\_IMAGE 132  
 FEATURE\_FIELD\_MISC 132  
 FEATURE\_FIELD\_PASSWORD 132  
 FEATURE\_FIELD\_SELECTED 133  
 FEATURE\_FIELD\_SIZE 133  
 FEATURE\_FIELD\_STATUS 133  
 FEATURE\_FIELD\_VISIBLE 134  
 FEATURE\_INFO\_ATTRIBUTE 134  
 FEATURE\_INFO\_COMPONENT\_FLAGS 134  
 FEATURE\_INFO\_COMPSIZE\_HIGH 134  
 FEATURE\_INFO\_COMPSIZE\_LOW 135  
 FEATURE\_INFO\_DATE 135  
 FEATURE\_INFO\_DATE\_EX 135  
 FEATURE\_INFO\_DESTINATION 136  
 FEATURE\_INFO\_HTTPLOCATION 136  
 FEATURE\_INFO\_LANGUAGE 137  
 FEATURE\_INFO\_MD5\_SIGNATURE 137  
 FEATURE\_INFO\_MISC 137  
 FEATURE\_INFO\_ORIGSIZE\_HIGH 138  
 FEATURE\_INFO\_ORIGSIZE\_LOW 138  
 FEATURE\_INFO\_OS 138  
 FEATURE\_INFO\_OVERWRITE 138  
 FEATURE\_INFO\_PLATFORM\_SUITE 139  
 FEATURE\_INFO\_TIME 139  
 FEATURE\_INFO\_VERSIONLS 139  
 FEATURE\_INFO\_VERSIONMS 140  
 FEATURE\_INFO\_VERSIONSTR 140  
 FEATURE\_OPCOST\_UNINSTALL\_FILE 140  
 FEATURE\_OPCOST\_UNINSTALL\_REGORINI 141  
 FEATURE\_OPCOST\_UNINSTALL\_UNREGFILE 141  
 FEATURE\_VALUE\_CRITICAL 141  
 FEATURE\_VALUE\_HIGHLYRECOMMENDED 141  
 FEATURE\_VALUE\_STANDARD 142  
 FeatureAddCost 809  
 FeatureAddItem 810  
 FeatureAddItem Example 812  
 FeatureAddUninstallCost 813  
 FeatureCompareSizeRequired 814  
 FeatureCompareSizeRequired Example 816  
 FeatureConfigureFeaturesFromSuite 818  
 FeatureDialog 819  
 FeatureDialog Example 822  
 FeatureError 823  
 FeatureError Example 826  
 FeatureErrorInfo 827  
 FeatureErrorInfo Example 828  
 FeatureFileEnum 830  
 FeatureFileEnum Example 832  
 FeatureFileInfo 834  
 FeatureFileInfo Example 840  
 FeatureFilterLanguage 843  
 FeatureFilterLanguage Example 845

- FeatureFilterOS 846
- FeatureFilterOS Example 850
- FeatureGetCost 852
- FeatureGetCostEx 854
- FeatureGetData 855
- FeatureGetData Example 859
- FeatureGetItemSize 860
- FeatureGetItemSize Example 861
- FeatureGetTotalCost 862
- FeatureInitialize 863
- FeatureInitialize Example 865
- FeatureIsItemSelected 866
- FeatureIsItemSelected Example 867
- FeatureListItems 868
- FeatureListItems Example 869
- FeatureLoadTarget 870
- FeatureMoveData 871
- FeatureMoveData Example 872
- FeaturePatch 876
- FeatureReinstall 877
- FeatureRemoveAll 878
- FeatureRemoveAllInLogOnly 879
- FeatureRemoveAllInMedia 880
- FeatureRemoveAllInMediaAndLog 881
- Features
  - adding items 810
- FeatureSaveTarget 882
  - destination location 819
  - dialog 819
- FeatureSelectItem 882
- FeatureSelectItem Example 884
- FeatureSelectNew 885
  - enumerating setup types 892
  - errors 823
- FeatureSetData 886
- FeatureSetData Example 889
- FeatureSetTarget 890
- FeatureSetTarget Example 891
- FeatureSetupTypeEnum 892
- FeatureSetupTypeEnum Example 892
- FeatureSetupTypeGetData 893
- FeatureSetupTypeGetData Example 895
- FeatureSetupTypeSet 898
  - fields 886
  - filtering 843
  - listing items 868
  - retrieving feature information 855
  - retrieving setup type data 893
  - selecting 819
  - selection status 866
  - setting data 886
  - setting the setup type 898
  - validating 909
- FeatureSetupTypeSet Example 899

- FeatureSpendCost 900
- FeatureSpendUninstallCost 901
- FeatureStandardSetupTypeSet 902
- FeatureTotalSize 904
- FeatureTotalSize Example 905
- FeatureTransferData 907
- FeatureUpdate 908
- FeatureValidate 909
- FeatureValidate Example 910
- FF\_FLAGS 129
- FI\_FTPLOCATION 136
- File attributes 142
- File media library 459
- File transfer 620
  - CopyFile 620
  - features 871
  - XCopyFile 1686
- FILE\_ADD\_FILE 142
- FILE\_ADD\_SUBDIRECTORY 142
- FILE\_ALL\_ACCESS 143
- FILE\_APPEND\_DATA 143
- FILE\_ATTR\_ARCHIVED 143, 1462
- FILE\_ATTR\_HIDDEN 143, 1462
- FILE\_ATTR\_NORMAL 143, 1462
- FILE\_ATTR\_READONLY 144, 1462
- FILE\_ATTR\_SYSTEM 144, 1462
- FILE\_ATTRIBUTE 144, 1462
- FILE\_BIN\_CUR 144, 1426
- FILE\_BIN\_END 144, 1426
- FILE\_BIN\_START 145, 1426
- FILE\_DATE 145, 959
- FILE\_DELETE\_CHILD 145
- FILE\_EXECUTE 145
- FILE\_EXISTS 145, 1012
- FILE\_INSTALLED 146, 1655
- FILE\_IS\_LOCKED 146, 1659
- FILE\_LINE\_LENGTH 146, 922
- FILE\_LIST\_DIRECTORY 146
- FILE\_LOCKED 146, 1012
- FILE\_MD5\_SIGNATURE 147
- FILE\_MODE\_APPEND 147, 626
- FILE\_MODE\_APPEND\_UNICODE 147
- FILE\_MODE\_BINARY 147, 1135
- FILE\_MODE\_BINARYREADONLY 147, 1135
- FILE\_MODE\_NORMAL 148, 1135
- FILE\_NO\_VERSION 148, 1646
- FILE\_NOT\_FOUND 148, 1646
- FILE\_RD\_ONLY 148, 916
- FILE\_READ\_ATTRIBUTES 149
- FILE\_READ\_DATA 149
- FILE\_READ\_EA 149
- FILE\_SHARED\_COUNT 149
- FILE\_SIZE 150, 959
- FILE\_SIZE\_HIGH 150

- FILE\_SIZE\_LOW 150
- FILE\_SRC\_OLD 150, 1655
- FILE\_TIME 150, 959
- FILE\_TRAVERSE 151
- FILE\_WRITE\_ATTRIBUTES 151
- FILE\_WRITE\_DATA 151
- FILE\_WRITE\_EA 151
- FILE\_WRITEABLE 151, 1012
- FileCompare 912
- FileCompare Example 914
- FileDeleteLine 916
- FileDeleteLine Example 917
- FileGrep 919
- FileGrep Example 920
- FileInsertLine 922
- FileInsertLine Example 923
- FILENAME 152, 1139
- FILENAME\_ONLY 152, 1139
- Files
  - attributes 959
  - closing 579
  - comparing 912
  - copying 1686
  - creating 626
  - critical 855
  - date and time 1462
  - deleting 724
  - deleting lines 916
  - exist 1012
  - File attributes 142
  - file pointer 1426
  - finding 931
  - hidden 142
  - inserting lines 922
  - InstallScript functions 459
  - InstallShield Silent 1365
  - locked 472
  - mode 1135
  - opening 1133
  - random access 1426
  - reading 1185
  - read-only 142
  - renaming 1259
  - searching 919
  - seeking 1426
  - self-registering files 759
  - setting the file mode 1135
  - shared files 472
  - supported languages 1644
  - system 142
  - writing 1675
- Fill with blanks or zeros 62
- Filtering 843
  - languages 843
  - operating systems 846
- FindAllDirs 925
- FindAllDirs Example 926
- FindAllFiles 927
- FindAllFiles Example 929
- FindFile 931
- FindFile Example 932
- Finding 925
  - directories 925
  - files 927
  - folders 925
  - path in search path 1150
  - substring 509
- FindWindow 933
- FindWindow Example 934
- FIXED\_DRIVE 152, 1001
- FlexNet Connect 461
  - GetUpdateStatus 1000
  - GetUpdateStatusReboot 1001
  - SdFinishUpdate 1338
  - SdFinishUpdateEx 1338
  - SdFinishUpdateReboot 1340
  - SetUpdateStatus 1488
  - SetUpdateStatusReboot 1489
  - UPDATE\_SERVICE\_INSTALL constant 270
  - UpdateServiceCheckForUpdates 1623
  - UPDATESERVICECOMPONENT constant 270
  - UpdateServiceCreateShortcut 1623
  - UpdateServiceEnableUpdateManagerInstall 1624
  - UpdateServiceGetAgentTarget 1624
  - UpdateServiceOnEnabledStateChange 1624
  - UpdateServiceRegisterProduct 1624
  - UpdateServiceRegisterProductEx 1625
  - UpdateServiceSetHost 1625
  - UpdateServiceSetLanguage 1625
- Flow control 77
- FOLDER\_APPDATA 323
- FOLDER\_APPLICATIONS 324
- FOLDER\_APPLICATIONS64 324
- FOLDER\_COMMON\_APPDATA 324
- FOLDER\_DESKTOP 964
- FOLDER\_DOTNET\_10 325
- FOLDER\_DOTNET\_11 325
- FOLDER\_DOTNET\_20 325
- FOLDER\_DOTNET\_30 325
- FOLDER\_DOTNET\_35 325
- FOLDER\_DOTNET\_40 325
- FOLDER\_FONTS 326
- FOLDER\_LOCAL\_APPDATA 326
- FOLDER\_PERSONAL 326
- FOLDER\_PROGRAMS 964
- FOLDER\_STARTMENU 964
- FOLDER\_STARTUP 327
- FOLDER\_TEMP 327



Folders [927](#)  
     adding icons [519](#)  
     adding shortcut folders through InstallScript [637](#)  
     changing [575](#)  
     checking [783](#)  
     creating [624](#)  
     deleting [722](#)  
     deleting items [726](#)  
     finding [925](#)  
     InstallScript functions [459](#)  
     replacing icons [1263](#)  
     retrieving shortcuts and subfolder names [964](#)  
     searching [927](#)  
     selecting [1436](#)  
     shared support [349](#)  
     support [350](#)  
 FONT\_AVAILABLE [152](#)  
 Fonts [690](#)  
     dialogs [967](#)  
 for [66](#)  
 Format specifiers [1518](#)  
 FormatMessage [935](#)  
 FormatMessage Example [936](#)  
 Free memory [973](#)  
 FTP location [855](#)  
 FULL [152](#), [1150](#)  
 \_FULLSCREEN [153](#), [1160](#)  
 \_FULLSCREENIZE [153](#), [1160](#)  
 FULLWINDOWMODE [153](#), [769](#)  
 Function block [56](#)  
 function keyword [433](#)  
     declaring [56](#)  
     overview [433](#)  
 FUNCTION\_EXPORTED [153](#)

## G

GBYTES [154](#), [952](#)  
 GENERIC\_ALL [154](#)  
 GENERIC\_EXECUTE [154](#)  
 GENERIC\_READ [154](#)  
 GENERIC\_WRITE [154](#)  
 GetAndAddAllFilesCost [937](#)  
 GetAndAddFileCost [938](#)  
 GetArrayFromISArray [939](#)  
 GetCharArrayFromISStringArray [940](#)  
 GetCurrentDialogName [941](#)  
 GetCurrentDir [942](#)  
 GetDir [943](#)  
 GetDir Example [944](#)  
 GetDisk [945](#)  
 GetDisk Example [946](#)  
 GetDiskInfo [947](#)  
 GetDiskInfo example [949](#)  
 GetDiskSpace [950](#)  
 GetDiskSpace Example [950](#)  
 GetDiskSpaceEx [952](#)  
 GetDiskSpaceEx Example [953](#)  
 GetEnvVar [954](#)  
 GetEnvVar Example [955](#)  
 GetExtendedErrInfo [956](#)  
 GetExtents [957](#)  
 GetExtents Example [958](#)  
 GetFileInfo [959](#)  
 GetFileInfo Example [962](#)  
 GetFolderNameList [964](#)  
 GetFolderNameList Example [966](#)  
 GetFont [967](#)  
 GetFont Example [968](#)  
 GetLine [971](#)  
 GetLine Example [972](#)  
 GetMemFree [973](#)  
 GetObject [973](#)  
 GetObjectByIndex [974](#)  
 GetObjectCount [975](#)  
 GetProfilnt [976](#)  
 GetProfilnt Example [977](#)  
 GetProfSectionKeyCount [977](#)  
 GetProfString [978](#)  
 GetProfString Example [980](#), [982](#)  
 GetProfStringList [981](#)  
 GetShortcutInfo [984](#)  
     example [986](#)  
 GetStatus [988](#)  
 GetSystemInfo [988](#)  
 GetSystemInfo Example [994](#)  
 GetTempFileNameS [997](#)  
 Getting [565](#)  
     configuration file value [603](#)  
     date [988](#)  
     default batch file name [565](#)  
     default system configuration file name [601](#)  
     destination folder [531](#)  
     environment variables [954](#)  
     feature data [855](#)  
     file attributes [959](#)  
     file's supported languages [1644](#)  
     free disk space [950](#)  
     free memory [973](#)  
     integer values from initialization files [976](#)  
     lines from a file [971](#)  
     list elements [1072](#)  
     media information [1123](#)  
     path [1153](#)  
     project settings [1123](#)  
     screen dimensions [957](#)  
     status of an object [988](#)  
     string length (bytes) [1580](#)



- string length (chars) 1581
- string values from initialization files 978
- time 988
- total memory 988
- valid drive list 1001
- window handle 581
- GetTrueTypeFontFileInfo 999
- GetUpdateStatus 1000
- GetUpdateStatusReboot 1001
- GetValidDrivesList 1001
- GetValidDrivesList Example 1002
- GetWCharArrayFromISStringArray 1004
- GetWindowHandle 1004
- GetWindowHandle Example 1005
- Global Event Handlers 378
- Global variables 303
- goto 68
- Graphics 1288
  - displaying in dialog 1288
- GREATER\_THAN 155, 1638
- GREEN 155, 1448
- Grep 919
- GTFIS\_OPTION\_DELETE\_TEMP\_FILE 155
- GTFIS\_OPTION\_DONT\_CREATE\_DIR 155
- GTFIS\_OPTION\_DONT\_RESOLVE\_TEXTSUBS 155
- GTFIS\_OPTION\_NONE 156

## H

- Handle 581
- Handler
  - window 1004
- HandlerEx example 1009
- HELP 1007
- HELP (InstallScript constant) 156
- Help handler 759
- Help topics 1307
- Hexadecimal values 62
- HIBYTE 1010
- Hidden files 793
- HIDE\_DISABLED\_BTNS 156
- HIWORD 1011
- HIWORD Example 1012
- HKEY\_CLASSES\_ROOT 156, 1207
- HKEY\_CURRENT\_USER 157, 1240
- HKEY\_LOCAL\_MACHINE 157, 1191
- HKEY\_USER\_SELECTABLE 158
- HKEY\_USER\_SELECTABLE\_AUTO 328
- HKEY\_USERS 157, 1191
- HKEYCURRENTROOTKEY 328
- HOURGLASS 158, 754
- HTTP location 855
- Hungarian notation 59
- HWND data type 289
- HWND\_DESKTOP 158, 1004
- HWND\_INSTALL 158, 1004

## I

- Icons 726
  - adding to program folder 519
  - deleting from program folder 726
  - replacing in folders 1263
- IDCANCEL 158
- Identifiers 57
- IDOK 159
- IDS\_IFX\_ERROR\_INVALID\_MEDIA\_PASSWORD 159
- if 67
- IFX\_COMPANY\_NAME 328
- IFX\_DISK1INSTALLED 329
- IFX\_INITIALIZED 329
- IFX\_INSTALLED\_DISPLAY\_VERSION 329
- IFX\_INSTALLED\_VERSION 329
- IFX\_KEYPATH\_PRODUCT\_INFO 329
- IFX\_MULTI\_INSTANCE\_SUFFIX 330
- IFX\_ONNEXTDISK\_PACKAGE\_CAPTION 159
- IFX\_ONNEXTDISK\_PACKAGE\_MSG 160
- IFX\_PRODUCT\_COMMENTS 330
- IFX\_PRODUCT\_DISPLAY\_NAME 330
- IFX\_PRODUCT\_DISPLAY\_VERSION 330
- IFX\_PRODUCT\_ICON 331
- IFX\_PRODUCT\_KEY 331
- IFX\_PRODUCT\_NAME 331
- IFX\_PRODUCT\_README 331
- IFX\_PRODUCT\_REGISTEREDCOMPANY 331
- IFX\_PRODUCT\_REGISTEREDOWNER 332
- IFX\_PRODUCT\_REGISTEREDSERIALNUM 332
- IFX\_PRODUCT\_SUPPORT\_CONTACT 332
- IFX\_PRODUCT\_SUPPORT\_PHONE 333
- IFX\_PRODUCT\_SUPPORT\_URL 333
- IFX\_PRODUCT\_UPDATE\_URL 333
- IFX\_PRODUCT\_URL 333
- IFX\_PRODUCT\_VERSION 334
- IFX\_SETUP\_TITLE 334
- IFX\_SUPPORTED\_VERSIONS 334
- Include files 369
  - specifying 369
- INCLUDE\_SUBDIR 160, 1686
- Indirection operator 504
- INDVFILESTATUS 160
- INFOFILENAME 559
- INFORMATION 160, 1518
- Information functions 463
- Initialization files 1683
  - adding lines 527
  - changing strings 1266
  - getting values 976
  - InstallScript functions 464

- restarting Windows 1335
- writing 1683
- Initialization Handlers 378
- initialize 863
- InitProperties 515
- Inserting 922
  - lines into a text file 922
- INSTALL\_GUID 335
- InstallationInfo 1670
- INSTALLDIR 334
- InstallScript Language Reference 51
- INSTALLSCRIPTMSI script variable 280
- INSTALLSCRIPTMSIEUI script variable 280
- InstallShield Silent 1365
  - SdMakeName 1365
- int data type 289
- Is 1012
- Is Example 1020
- IS\_386 190, 988
- IS\_486 190, 988
- IS\_ALPHA 190, 988
- IS\_CDROM 190, 988
- IS\_EGA 190, 988
- IS\_FIXED 191, 988
- IS\_FOLDER 191, 1263
- IS\_ITEM 191, 1263
- IS\_NULLSTR\_PTR 336
- IS\_PENTIUM 191, 988
- IS\_PERMISSIONS\_OPTION\_64BIT\_OBJECT 161
- IS\_PERMISSIONS\_OPTION\_ALLOW\_ACCESS 161
- IS\_PERMISSIONS\_OPTION\_DENY\_ACCESS 161
- IS\_PERMISSIONS\_OPTION\_NO\_APPLYDOWN 161
- IS\_PERMISSIONS\_TYPE\_FILE 161
- IS\_PERMISSIONS\_TYPE\_FOLDER 162
- IS\_PERMISSIONS\_TYPE\_REGISTRY 162
- IS\_REMOTE 191, 988
- IS\_REMOVABLE 192, 988
- IS\_SVGA 192, 988
- IS\_UNKNOWN 192, 988
- IS\_UVGA 192, 988
- IS\_VGA 192, 988
- IS\_WINDOWS 193, 988
- IS\_WINDOWS9X 193, 988
- IS\_WINDOWSNT 193, 988
- IS\_XVGA 193, 988
- ISCompareServicePack 1021
- ISCompareServicePack Example 1022
- ISDeterminePlatform 1023
- ISDIFX\_OPTION\_DONT\_ASSOCIATE 162
- ISDIFX\_OPTION\_DONT\_RESOLVE\_TEXTSUBS 162
- ISDIFX\_OPTION\_LOG\_IN\_DRIVER\_PACKAGE\_PATH 162
- ISDIFX\_OPTION\_NO\_REPAIR 163
- ISDIFXAPPID 335
- IsEmpty 1023
- IsEmpty Example 1024
- ISERR\_GEN\_FAILURE 163
- ISERR\_SUCCESS 163
- ISLANG constants 298
- ISLANG\_AFRIKAANS 163
- ISLANG\_AFRIKAANS\_STANDARD 163
- ISLANG\_ALBANIAN 164
- ISLANG\_ALBANIAN\_STANDARD 164
- ISLANG\_ALL 164
- ISLANG\_ARABIC 164
- ISLANG\_ARABIC\_ALGERIA 164
- ISLANG\_ARABIC\_BAHRAIN 164
- ISLANG\_ARABIC\_EGYPT 164
- ISLANG\_ARABIC\_IRAQ 164
- ISLANG\_ARABIC\_JORDAN 165
- ISLANG\_ARABIC\_KUWAIT 165
- ISLANG\_ARABIC\_LEBANON 165
- ISLANG\_ARABIC\_LIBYA 165
- ISLANG\_ARABIC\_MOROCCO 165
- ISLANG\_ARABIC\_OMAN 165
- ISLANG\_ARABIC\_QATAR 165
- ISLANG\_ARABIC\_SAUDIARABIA 165
- ISLANG\_ARABIC\_SYRIA 166
- ISLANG\_ARABIC\_TUNISIA 166
- ISLANG\_ARABIC\_UAE 166
- ISLANG\_ARABIC\_YEMEN 166
- ISLANG\_BASQUE 166
- ISLANG\_BASQUE\_STANDARD 166
- ISLANG\_BELARUSIAN 166
- ISLANG\_BELARUSIAN\_STANDARD 166
- ISLANG\_BULGARIAN 167
- ISLANG\_BULGARIAN\_STANDARD 167
- ISLANG\_CATALAN 167
- ISLANG\_CATALAN\_STANDARD 167
- ISLANG\_CHINESE 167
- ISLANG\_CHINESE\_HONGKONG 167
- ISLANG\_CHINESE\_PRC 167
- ISLANG\_CHINESE\_SINGAPORE 167
- ISLANG\_CHINESE\_TAIWAN 168
- ISLANG\_CROATIAN 168
- ISLANG\_CROATIAN\_STANDARD 168
- ISLANG\_CZECH 168
- ISLANG\_CZECH\_STANDARD 168
- ISLANG\_DANISH 168
- ISLANG\_DANISH\_STANDARD 168
- ISLANG\_DUTCH 168
- ISLANG\_DUTCH\_BELGIAN 169
- ISLANG\_DUTCH\_STANDARD 169
- ISLANG\_ENGLISH 169
- ISLANG\_ENGLISH\_AUSTRALIAN 169
- ISLANG\_ENGLISH\_BELIZE 169
- ISLANG\_ENGLISH\_CANADIAN 169
- ISLANG\_ENGLISH\_CARIBBEAN 169
- ISLANG\_ENGLISH\_IRELAND 169

ISLANG\_ENGLISH\_JAMAICA 170  
 ISLANG\_ENGLISH\_NEWZEALAND 170  
 ISLANG\_ENGLISH\_SOUTHAFRICA 170  
 ISLANG\_ENGLISH\_TRINIDAD 170  
 ISLANG\_ENGLISH\_UNITEDKINGDOM 170  
 ISLANG\_ENGLISH\_UNITEDSTATES 170  
 ISLANG\_ESTONIAN 170  
 ISLANG\_ESTONIAN\_STANDARD 170  
 ISLANG\_FAEROESE 171  
 ISLANG\_FAEROESE\_STANDARD 171  
 ISLANG\_FARSI 171  
 ISLANG\_FARSI\_STANDARD 171  
 ISLANG\_FINNISH 171  
 ISLANG\_FINNISH\_STANDARD 171  
 ISLANG\_FRENCH 171  
 ISLANG\_FRENCH\_BELGIAN 171  
 ISLANG\_FRENCH\_CANADIAN 172  
 ISLANG\_FRENCH\_LUXEMBOURG 172  
 ISLANG\_FRENCH\_STANDARD 172  
 ISLANG\_FRENCH\_SWISS 172  
 ISLANG\_GERMAN 172  
 ISLANG\_GERMAN\_AUSTRIAN 172  
 ISLANG\_GERMAN\_LIECHTENSTEIN 172  
 ISLANG\_GERMAN\_LUXEMBOURG 172  
 ISLANG\_GERMAN\_STANDARD 173  
 ISLANG\_GERMAN\_SWISS 173  
 ISLANG\_GREEK 173  
 ISLANG\_GREEK\_STANDARD 173  
 ISLANG\_HEBREW 173  
 ISLANG\_HEBREW\_STANDARD 173  
 ISLANG\_HUNGARIAN 173  
 ISLANG\_HUNGARIAN\_STANDARD 173  
 ISLANG\_ICELANDIC 174  
 ISLANG\_ICELANDIC\_STANDARD 174  
 ISLANG\_INDONESIAN 174  
 ISLANG\_INDONESIAN\_STANDARD 174  
 ISLANG\_ITALIAN 174  
 ISLANG\_ITALIAN\_STANDARD 174  
 ISLANG\_ITALIAN\_SWISS 174  
 ISLANG\_JAPANESE 174  
 ISLANG\_JAPANESE\_STANDARD 175  
 ISLANG\_KOREAN 175  
 ISLANG\_KOREAN\_JOHAB 175  
 ISLANG\_KOREAN\_STANDARD 175  
 ISLANG\_LATVIAN 175  
 ISLANG\_LATVIAN\_STANDARD 175  
 ISLANG\_LITHUANIAN 175  
 ISLANG\_LITHUANIAN\_STANDARD 175  
 ISLANG\_NORWEGIAN 176  
 ISLANG\_NORWEGIAN\_BOKMAL 176  
 ISLANG\_NORWEGIAN\_NYNORSK 176  
 ISLANG\_POLISH 176  
 ISLANG\_POLISH\_STANDARD 176  
 ISLANG\_PORTUGUESE 176  
 ISLANG\_PORTUGUESE\_BRAZILIAN 176  
 ISLANG\_PORTUGUESE\_STANDARD 176  
 ISLANG\_ROMANIAN 177  
 ISLANG\_ROMANIAN\_STANDARD 177  
 ISLANG\_RUSSIAN 177  
 ISLANG\_RUSSIAN\_STANDARD 177  
 ISLANG\_SERBIAN\_CYRILLIC 177  
 ISLANG\_SERBIAN\_LATIN 177  
 ISLANG\_SLOVAK 177  
 ISLANG\_SLOVAK\_STANDARD 177  
 ISLANG\_SLOVENIAN 178  
 ISLANG\_SLOVENIAN\_STANDARD 178  
 ISLANG\_SPANISH 178  
 ISLANG\_SPANISH\_ARGENTINA 178  
 ISLANG\_SPANISH\_BOLIVIA 178  
 ISLANG\_SPANISH\_CHILE 178  
 ISLANG\_SPANISH\_COLOMBIA 178  
 ISLANG\_SPANISH\_COSTARICA 178  
 ISLANG\_SPANISH\_DOMINICANREPUBLIC 179  
 ISLANG\_SPANISH\_ECUADOR 179  
 ISLANG\_SPANISH\_ELSALVADOR 179  
 ISLANG\_SPANISH\_GUATEMALA 179  
 ISLANG\_SPANISH\_HONDURAS 179  
 ISLANG\_SPANISH\_MEXICAN 179  
 ISLANG\_SPANISH\_MODERNSORT 179  
 ISLANG\_SPANISH\_NICARAGUA 179  
 ISLANG\_SPANISH\_PANAMA 180  
 ISLANG\_SPANISH\_PARAGUAY 180  
 ISLANG\_SPANISH\_PERU 180  
 ISLANG\_SPANISH\_PUERTORICO 180  
 ISLANG\_SPANISH\_TRADITIONALSORT 180  
 ISLANG\_SPANISH\_URUGUAY 180  
 ISLANG\_SPANISH\_VENEZUELA 180  
 ISLANG\_SW 181  
 ISLANG\_SWEDISH 180  
 ISLANG\_SWEDISH\_STANDARD 181  
 ISLANG\_THAI 181  
 ISLANG\_THAI\_STANDARD 181  
 ISLANG\_TURKISH 181  
 ISLANG\_TURKISH\_STANDARD 181  
 ISLANG\_UKRAINIAN 181  
 ISLANG\_UKRAINIAN\_STANDARD 181  
 ISLANG\_VIETNAMESE 182  
 ISLANG\_VIETNAMESE\_STANDARD 182  
 ISMSI\_HANDLE 335  
 IsObject 1025  
 ISOS\_ST\_ALL 184  
 ISOS\_ST\_BACKOFFICE 184  
 ISOS\_ST\_DATACENTER 184  
 ISOS\_ST\_ENTERPRISE 185  
 ISOS\_ST\_PROC\_ARCH\_32 185  
 ISOS\_ST\_PROC\_ARCH\_AMD64 185  
 ISOS\_ST\_PROC\_ARCH\_IA64 185  
 ISOS\_ST\_SERVER 186

ISOS\_ST\_SERVER2003\_R2 186  
 ISOS\_ST\_SMALLBUSINESS 186  
 ISOS\_ST\_SMALLBUSINESS\_RESTRICTED 187  
 ISOS\_ST\_TERMINAL 187  
 ISOS\_ST\_WORKSTATION 187  
 ISOS\_ST\_XP\_HOME 187  
 ISOS\_ST\_XP\_PRO 188  
 ISOSL\_ALL 182, 846  
 ISOSL\_NT40 846  
 ISOSL\_NT40\_ALPHA 846  
 ISOSL\_SUPPORTED 182  
 ISOSL\_WIN10 183  
 ISOSL\_WIN2000 846  
 ISOSL\_WIN2000\_ALPHA 846  
 ISOSL\_WIN7\_SERVER2008R2 182  
 ISOSL\_WIN8 182  
 ISOSL\_WIN81 183  
 ISOSL\_WIN95 846  
 ISOSL\_WIN98 846  
 ISOSL\_WINSERVER2003 183  
 ISOSL\_WINVISTA 183  
 ISOSL\_WINVISTA\_SERVER2008 183  
 ISOSL\_WINXP 184  
 ISRES 336  
 ISURL\_COMPONENTS 289  
 ISUS\_AGENT\_FEATURE 188  
 ISUS\_MAIN\_FEATURE 188  
 ISUS\_PRODUCT\_CODE 281  
 ISUS\_TEXTSUB\_HOST 188  
 ISUS\_TEXTSUB\_INTERVAL 188  
 ISUS\_TEXTSUB\_LANGUAGE 189  
 ISUS\_TEXTSUB\_LOGO 189  
 ISUS\_TEXTSUB\_MANAGER 189  
 ISUS\_TEXTSUB\_VERSION 189  
 ISUS\_UPDATERMANAGER\_FEATURE 189  
 ISUSER 336  
 ISVERSION 337

## K

KBYTES 193, 952  
 KEY\_CREATE\_LINK 194  
 KEY\_CREATE\_SUB\_KEY 194  
 KEY\_ENUMERATE\_SUB\_KEYS 194  
 KEY\_NOTIFY 194  
 KEY\_QUERY\_VALUE 194  
 KEY\_SET\_VALUE 195  
 Keywords 66
 

- abort 65
- BYREF 502
- BYVAL 503
- case 73
- default 73
- downto 66

else 69  
 elseif 70  
 endfor 66  
 endif 68  
 endprogram 56  
 endswitch 73  
 endwhile 75  
 exit 65  
 export 66  
 external 66  
 for 66  
 goto 67  
 if 70  
 method 70  
 program 56  
 property() 70  
 prototype 71  
 repeat 71  
 return 72  
 step 66  
 switch 73  
 then 67  
 to 66  
 typedef 295  
 until 71  
 while 75

## L

LAAW\_OPTION\_CHANGEDIRECTORY 195  
 LAAW\_OPTION\_FIXUP\_PROGRAM 195  
 LAAW\_OPTION\_HIDDEN 195  
 LAAW\_OPTION\_MAXIMIZED 195  
 LAAW\_OPTION\_MINIMIZED 196  
 LAAW\_OPTION\_NO\_CHANGEDIRECTORY 196  
 LAAW\_OPTION\_NOWAIT 196  
 LAAW\_OPTION\_SET\_BATCH\_INSTALL 196  
 LAAW\_OPTION\_SHOW\_HOURLASS 197  
 LAAW\_OPTION\_USE\_CALLBACK 197  
 LAAW\_OPTION\_USE\_SHELLEXECUTE 197  
 LAAW\_OPTION\_WAIT 198  
 LAAW\_OPTION\_WAIT\_INCL\_CHILD 198  
 LAAW\_PARAMETERS 337  
 LAAW\_PROCESS\_INFORMATION 339  
 LAAW\_SHELLEXECUTEINFO 339  
 LAAW\_SHELLEXECUTEVERB 340  
 LAAW\_STARTUPINFO 340  
 Label 67  
 LANGUAGE 198  
 Language 298
 

- AskYesNo dialog 544
- filtering 843
- IDs 298
- keywords 65

- selected language 349
  - supported by file 1644
- LANGUAGE\_SUPPORTED constant 198
- LaunchApp 1025
- LaunchApp Example 1025
- LaunchAppAndWait 1026
- LaunchAppAndWait Example 1026
- LaunchAppAndWaitInitStartupInfo 1028
- LaunchApplication 1030
- LaunchApplicationInit 1036
- Launching 453
  - another setup script 453
  - program from a setup script 1026
- LESS\_THAN 199, 912
- License agreement 1346, 1348, 1351, 1354, 1356, 1359
- LINE\_NUMBER 199, 916
- LIST
  - data type 289
- List 1043
- LIST\_NULL 200, 1051
  - adding elements 1043
- ListAddItem 1039
- ListAddItem Example 1040
- ListAddList 1042
- ListAddString 1043
- ListAddString Example 1044
- ListAppendFromArray 1046
- ListAppendToArray 1046
- LISTBOX\_ENTER 199, 672
- LISTBOX\_SELECT 199, 672
  - components 868
- ListConvertNumToStr 1047
- ListConvertStrToNum 1048
- ListCount 1049
  - counting elements 1049
- ListCount Example 1050
- ListCreate 1051
  - creating 1051
- ListCreate Example 1052
- ListCurrentItem 1053
- ListCurrentItem Example 1054
- ListCurrentString 1055
- ListCurrentString Example 1056
- ListDeleteAll 1057
- ListDeleteItem 1058
- ListDeleteItem Example 1059
- ListDeleteString 1061
- ListDeleteString Example 1062
- ListDestroy 1064
  - destroying 1064
  - finding an element 1070
- ListDestroy Example 1064
- ListFindItem 1065
- ListFindItem Example 1066
- ListFindKeyValueString 1068
- ListFindString 1070
- ListFindString Example 1070
- LISTFIRST 199, 1090
- ListGetFirstItem 1072
- ListGetFirstItem Example 1073
- ListGetFirstString 1074
- ListGetFirstString Example 1075
- ListGetIndex 1076
- ListGetNextItem 1077
- ListGetNextItem Example 1078
- ListGetNextString 1079
  - getting elements 1055
  - InstallScript functions 465
- ListGetNextString Example 1080
- ListGetType 1082
- ListGetType Example 1082
- LISTLAST 200, 1090
- LISTNEXT 200, 1090
- LISTPREV 200, 1090
- ListProcessing 1098
- ListReadFromFile 1083
- ListReadFromFile Example 1084
- ListSetCurrentItem 1085
- ListSetCurrentItem Example 1086
- ListSetCurrentString 1088
- ListSetCurrentString Example 1089
- ListSetIndex 1090
- ListSetIndex Example 1092
- ListValid 1094
- ListValidType 1096
- ListWriteToFile 1098
- ListWriteToFile Example 1099
- ListWriteToFileEx 1100
- LoadStringFromStringTable 1101
- LOBYTE 1102
- Local variables 303
- Locked files 1012
  - InstallScript functions 472
  - testing 1012
- LOCKEDFILE 200, 1012
- Log file 626
- LOGGING 201, 1012
- Logical operators 504
- LogReadCustomNumber 1103
- LogReadCustomNumber Example 1104
- LogReadCustomString 1105
- LogReadCustomString Example 1107
- LogWriteCustomNumber 1108
- LogWriteCustomNumber Example 1109
- LogWriteCustomString 1110
- LogWriteCustomString Example 1111
- LONG data type 289
- Long file name functions 467

Long filenames [1112](#)  
 LongPathFromShortPath [1112](#)  
 LongPathFromShortPath Example [1113](#)  
 LongPathToQuote [1114](#)  
 LongPathToQuote Example [1115](#)  
 LongPathToShortPath [1116](#)  
 LongPathToShortPath Example [1117](#)  
 Loop [71](#)  
 LOWER\_LEFT [201](#), [1160](#)  
 LOWER\_RIGHT [201](#), [1166](#)  
 LOWORD [1118](#)  
 LOWORD Example [1119](#)  
 LPSTR data type [289](#)  
 LPWSTR data type [289](#)  
 LWFT\_OPTION\_WRITE\_AS\_ANSI [202](#)  
 LWFT\_OPTION\_WRITE\_AS\_UNICODE [202](#)  
 LWTF\_OPTION\_APPEND\_TO\_FILE [201](#)  
 LWTF\_OPTION\_WRITE\_AS\_UNICODE [202](#)

## M

MAGENTA [202](#), [1448](#)  
 MAINT\_OPTION [343](#)  
 MAINT\_OPTION\_MULTI\_INSTANCE [343](#)  
 MAINT\_OPTION\_NONE [343](#)  
 MAINT\_OPTION\_STANDARD [343](#)  
 MAINTENANCE [343](#)  
 MaintenanceStart [1120](#)  
 Math coprocessor [1012](#)  
 MATH\_COPROCESSOR [202](#), [1012](#)  
 MB\_STYLE [1518](#)  
 MBYTES [203](#), [952](#)  
 MEDIA [343](#)  
 Media library [343](#)  
 MEDIA\_FIELD\_COMPANY\_NAME [203](#)  
 MEDIA\_FIELD\_MEDIA\_FLAGS [204](#)  
 MEDIA\_FIELD\_PREVIOUS\_VERSIONS [204](#)  
 MEDIA\_FIELD\_PRODUCT\_COMMENTS [204](#)  
 MEDIA\_FIELD\_PRODUCT\_EXE [204](#)  
 MEDIA\_FIELD\_PRODUCT\_ICON [205](#)  
 MEDIA\_FIELD\_PRODUCT\_NAME [205](#)  
 MEDIA\_FIELD\_PRODUCT\_NOMODIFY [203](#)  
 MEDIA\_FIELD\_PRODUCT\_NOREMOVE [203](#)  
 MEDIA\_FIELD\_PRODUCT\_README [205](#)  
 MEDIA\_FIELD\_PRODUCT\_SUPPORT\_CONTACT [206](#)  
 MEDIA\_FIELD\_PRODUCT\_SUPPORT\_PHONE [206](#)  
 MEDIA\_FIELD\_PRODUCT\_SUPPORT\_URL [206](#)  
 MEDIA\_FIELD\_PRODUCT\_UPDATE\_URL [206](#)  
 MEDIA\_FIELD\_PRODUCT\_URL [207](#)  
 MEDIA\_FIELD\_PRODUCT\_VERSION [207](#)  
 MEDIA\_FIELD\_TARGETDIR [207](#)  
 MEDIA\_FLAG\_FORMAT\_DIFFERENTIAL [208](#)  
 MEDIA\_FLAG\_FORMAT\_PATCH [208](#)  
 MEDIA\_FLAG\_UPDATESMODE\_SUPPORTED [208](#)  
 MEDIA\_PASSWORD\_KEY [208](#)  
 MediaGetData [1123](#)  
 MediaGetDataEx [1123](#)  
 Member [295](#)  
     data structure [295](#)  
     operator [505](#)  
 Memory [988](#)  
     extended [988](#)  
     free [973](#)  
     total [988](#)  
 MessageBeep [1126](#)  
 MessageBeep Example [1126](#)  
 MessageBox [1127](#)  
 MessageBox Example [1128](#)  
 MessageBoxEx [1129](#)  
     send to windows [1438](#)  
 METAFILE [209](#), [1514](#)  
 Metafiles [1454](#)  
     method [70](#)  
 MIDI files [1170](#)  
 Miscellaneous Event Handlers [408](#)  
 Miscellaneous functions [467](#)  
 MMEDIA\_AVI [209](#), [1170](#)  
 MMEDIA\_MIDI [209](#), [1170](#)  
 MMEDIA\_PLAYASYNCH [209](#), [1170](#)  
 MMEDIA\_PLAYCONTINUOUS [210](#), [1170](#)  
 MMEDIA\_PLAYSYNCH [210](#), [1170](#)  
 MMEDIA\_STOP [210](#), [1170](#)  
 MMEDIA\_SWF [210](#), [1170](#)  
 MMEDIA\_WAVE [210](#), [1170](#)  
 MODE [344](#)  
 MODIFY [211](#)  
 modulus operator [499](#)  
 Move Data Handlers [390](#)  
 Moving [1155](#)  
     lines in a batch file [567](#)  
     path in path buffer [1155](#)  
 MSI\_TARGETDIR [344](#)  
 MULTI\_INSTANCE\_COUNT [344](#)

## N

Nested if-then-else structure [69](#)  
 Nested while Example [75](#)  
 Networks [1001](#)  
     getting driver name from System.ini [988](#)  
     mapping drives [1001](#)  
     remote registry [1191](#)  
 Newline character [60](#)  
 NEXT [211](#), [1307](#)  
 NEXTBUTTON [212](#), [754](#)  
 NO [212](#), [544](#)  
 NO\_SUBDIR [213](#)  
 NONEXCLUSIVE [212](#), [1281](#)

NORMAL\_PRIORITY\_CLASS [213](#)  
 NORMALMODE [213](#), [344](#)  
 NOSET [213](#), [789](#)  
 Not operator [504](#)  
 NOTEXISTS [213](#), [783](#)  
 NOTHING [72](#)  
 NOWAIT [761](#)  
 NULL [214](#), [519](#)  
 NUMBER data type [289](#)  
 NUMBERLIST [214](#), [1051](#)  
 NumToStr [1131](#)  
 NumToStr Example [1132](#)

## O

Object handlers [514](#)  
     InitProperties [515](#)  
     ReadProperties [515](#)  
     WriteProperties [515](#)  
 Objects [511](#)  
     InstallScript functions [468](#)  
 Objects object  
     setting status [1480](#)  
 OFF [214](#), [1567](#)  
 OK [214](#)  
 ON [215](#), [1567](#)  
 OnAbort [411](#)  
 OnAdminInstallUIAfter [411](#)  
 OnAdminInstallUIBefore [411](#)  
 OnAdminPatchUIAfter [412](#)  
 OnAdminPatchUIBefore [412](#)  
 OnAdvertisementAfter [412](#)  
 OnAdvertisementBefore [412](#)  
 OnAppSearch [384](#)  
 OnBegin [385](#)  
 OnCanceling [412](#)  
 OnCCPSearch [386](#)  
 OnCheckMediaPassword [379](#)  
 OnComponentError [413](#)  
 OnCustomizeUninstInfo [392](#)  
 OnDIFxLogCallback [414](#)  
 OnEnd [402](#)  
 OnError [415](#)  
 OnException [415](#)  
 OnFileError [416](#)  
 OnFileLocked [416](#)  
 OnFileReadOnly [417](#)  
 OnFilesInUse [418](#)  
 OnFilterComponents [379](#)  
 OnFirstUIAfter [402](#)  
 OnFirstUIBefore [386](#)  
 OnGeneratedMSIScript [392](#)  
 OnGeneratingMSIScript [392](#)  
 OnHelp [419](#)  
 OnIISComponentInstalled [392](#)  
 OnIISInitialize [386](#)  
 OnIISUninitialize [402](#)  
 OnIISVRootUninstalling [393](#)  
 OnInstalled [405](#)  
 OnInstalledFile [393](#)  
 OnInstalledFontFile [393](#)  
 OnInstallFilesActionAfter [394](#)  
 OnInstallFilesActionBefore [394](#)  
 OnInstalling [406](#)  
 OnInstallingFile [394](#)  
 OnInternetError [419](#)  
 OnLaunchAppAndWaitCallback [420](#)  
 OnLogonUserSetMsiProperties [421](#)  
 ONLYDIR [215](#), [722](#)  
 OnMaintUIAfter [403](#)  
 OnMaintUIBefore [387](#)  
 OnMD5Error [421](#)  
 OnMoved [395](#)  
 OnMoveData [395](#)  
 OnMoving [395](#)  
 OnMsiSilentInstall [423](#)  
 OnNetApiCreateUserAccount [396](#)  
 OnNextDisk [423](#)  
 OnOutOfDiskSpace [423](#)  
 OnPatchUIAfter [423](#)  
 OnPatchUIBefore [424](#)  
 OnRebooted [424](#)  
 OnRemovingSharedFile [424](#)  
 OnResumeUIAfter [425](#)  
 OnResumeUIBefore [426](#)  
 OnRMFilesInUse [426](#)  
 OnSelfRegistrationError [427](#)  
 OnSetTARGETDIR [380](#)  
 OnSetUpdateMode [381](#)  
 OnShowUI [429](#)  
 OnSQLBatchScripts [396](#)  
 OnSQLComponentInstalled [396](#)  
 OnSQLComponentUninstalled [397](#)  
 OnSQLLogin [387](#)  
 OnSQLServerInitialize [387](#)  
 OnSQLServerInitializeMaint [388](#)  
 OnSuiteInstallAfter [403](#)  
 OnSuiteInstallBefore [388](#)  
 OnSuiteMaintAfter [403](#)  
 OnSuiteMaintBefore [388](#)  
 OnSuiteShowUI [431](#)  
 OnSuiteUpdateAfter [403](#)  
 OnSuiteUpdateBefore [388](#)  
 OnUninstall (InstallScript) [431](#)  
 OnUnInstalled [406](#)  
 OnUninstalledFile [397](#)  
 OnUninstalling [407](#)  
 OnUninstallingDIFxDriverFile [398](#)



- OnUninstallingFile 397
- OnUninstallingFontFile 398
- OnUpdateUIAfter 404
- OnUpdateUIBefore 389
- OnWarning 428
- OnXMLComponentInstalled 399
- OnXMLComponentUninstalling 399
- OnXMLInitialize 389
- OnXMLUninitialize 404
- OpenFile 1133
- OpenFile Example 1134
- OpenFileMode 1135
- OpenFileMode Example 1138
- Opening 1133
  - files 1133
- Operating system
  - filtering 846
  - type 988
  - version 988
- Operators 497
  - address 299
  - arithmetic 498
  - assignment 500
  - bit 501
  - BYREF 502
  - BYVAL 503
  - indirection 504
  - logical 504
  - member 505
  - precedence 498
  - preprocessor directives 365
  - relational 507
  - string 508
  - string constant 508
  - string entries 294
  - structure pointer 509
- Options 534
  - getting user options 1370
- Or operator 504
- OTHER\_FAILURE 215, 912
- OUT\_OF\_DISK\_SPACE 215, 922

## P

- PACKAGE\_LOCATION 345
- Pad with blanks or zeros 62
- PARALLEL 216, 988
- Parallel ports 988
- ParsePath 1139
- ParsePath Example 1141
- ParseUrl 1142
- Parsing, strings 1577
- PARTIAL 216, 1147
- Password 855
- PATH 216, 1139
- Path 498
- Path buffer 1147
  - adding a search directory 1144
  - deleting a search directory 1147
  - getting a path string 1153
  - InstallScript functions 469
  - repositioning a search directory 1155
  - searching for a directory 1150
  - storing a search directory 1158
- PATH\_EXISTS 216, 1012
- PathAdd 1144
  - adding to batch file 547
  - appending to 498
- PathAdd Example 1145
- PathDelete 1147
  - exists 1012
- PathDelete Example 1148
- PathFind 1150
- PathFind Example 1151
- PathGet 1153
- PathGet Example 1153
- PathMove 1155
  - parsing 1139
  - removing a trailing backslash 1584
  - removing drive designation 943
- PathMove Example 1156
- PathSet 1158
- PathSet Example 1159
- Pausing 721
- PCRESTORE 217
- Percent sign 62
- PERSONAL 217, 1175
- PlaceBitmap 1160
- Placebitmap Examples 1165
- Placeholders 1422
- PlaceWindow 1166
- PlaceWindow Example 1169
- PlayMMedia 1170
- PlayMMedia Example 1172
- POINTER data type 289
- Ports 988
  - parallel 988
  - serial 988
- PostShowComponentDlg 1173
- Predefined constants 79
- Predefined Script Variables 279
- Preprocessor directives 365
- Preprocessor statements 365
  - #define 294
  - #elif 367
  - #error 367
  - #ifdef 368
  - #ifndef 368



- #include 369
- #undef 370
- #warning 370
- PreShowComponentDlg 1174
- PROCESSOR\_AMD\_X8664 357
- PROCESSOR\_ARCH\_AMD64 351
- PROCESSOR\_ARCH\_IA64 351
- PROCESSOR\_ARCH\_INTEL 351
- PROCESSOR\_ARCHITECTURE\_AMD64 357
- PROCESSOR\_ARCHITECTURE\_IA64 357
- PROCESSOR\_ARCHITECTURE\_INTEL 357
- PROCESSOR\_ARCHITECTURE\_UNKNOWN 357
- PROCESSOR\_INTEL\_386 357
- PROCESSOR\_INTEL\_486 357
- PROCESSOR\_INTEL\_IA64 357
- PROCESSOR\_INTEL\_PENTIUM 357
- Product 1378
- PRODUCT\_GUID 345
  - name 1405
- PRODUCT\_INSTALLED 345
- ProgDefGroupType 327
- program 56
- Program folder items 519
  - adding 519
- Program folders 327
  - creating 631
  - creating through InstallScript 647
  - deleting 728
  - deleting icons 726
  - displaying 1497
  - FOLDER\_PROGRAMS 327
  - selecting 1307
- PROGRAMFILES 345
- PROGRAMFILES64 346
- Progress indicator 1482
  - initializing 1482
  - updating 1567
- property() keyword 70
- prototype keyword 71
- Punctuation rules 57

## Q

- Query 1231
- QueryProgItem 1177
  - registry 1231
- QueryProgItem Example 1179
- QueryShellMgr 1181
- QueryShellMgr Example 1182
- Quotation marks 61
  - inserting in a string 60

## R

- READ\_CONTROL 217
- ReadArrayProperty 1183
- ReadBoolProperty 1184
- ReadBytes 1185
- ReadBytes Example 1186
- Reading 1083
  - binary files 1185
  - text files 971
- ReadNumberProperty 1187
- Read-only files 142
- ReadProperties 515
- ReadStringProperty 1188
- RebootDialog 1189
- RebootDialog Example 1190
- REBOOTED 217
- Rebooting target machine
  - through SdFinishReboot 1335
- RECORDMODE 218
- RED 218, 1448
- Reference keys 565
  - batch files 562
  - configuration files 598
- REGDB\_APPPATH 218, 1243
- REGDB\_APPPATH\_DEFAULT 218, 1218
- REGDB\_BINARY 219, 1237
- REGDB\_ERR\_CONNECTIONEXISTS 219, 1191
- REGDB\_ERR\_CORRUPTEDREGISTRY 219, 1191
- REGDB\_ERR\_INITIALIZATION 219, 1191
- REGDB\_ERR\_INVALIDHANDLE 220, 1191
- REGDB\_ERR\_INVALIDNAME 220, 1191
- REGDB\_KEYPATH\_APPPATHS 220
- REGDB\_KEYPATH\_DOTNET\_10 220
- REGDB\_KEYPATH\_DOTNET\_11 220
- REGDB\_KEYPATH\_DOTNET\_20 221
- REGDB\_KEYPATH\_DOTNET\_30 221
- REGDB\_KEYPATH\_DOTNET\_30\_SP 221
- REGDB\_KEYPATH\_DOTNET\_35 222
- REGDB\_KEYPATH\_DOTNET\_40\_CLIENT 222
- REGDB\_KEYPATH\_DOTNET\_40\_FULL 222
- REGDB\_KEYPATH\_ISUNINSTINFO 223
- REGDB\_KEYPATH\_RUN 223
- REGDB\_KEYPATH\_RUNONCE 223
- REGDB\_KEYPATH\_RUNONCEEX 223
- REGDB\_KEYPATH\_SHARED DLLS 223
- REGDB\_KEYPATH\_UNINSTALL 224
- REGDB\_KEYPATH\_WINCURREVER 224
- REGDB\_KEYPATH\_WINCURREVER\_AUTO 224
- REGDB\_KEYPATH\_WINNTCURREVER 224
- REGDB\_KEYS 224, 1231
- REGDB\_NAMES 224, 1231
- REGDB\_NUMBER 225, 1237
- REGDB\_OPTION\_DISABLETEXTSUBS 346

- REGDB\_OPTION\_NO\_DELETE\_OLD\_MAJMIN\_VERSION 346
- REGDB\_OPTION\_USE\_DEFAULT\_OPTIONS 346
- REGDB\_OPTION\_WOW64\_64KEY 346
- REGDB\_OPTIONS 346
- REGDB\_STRING 225, 1237
- REGDB\_STRING\_EXPAND 225, 1223
- REGDB\_STRING\_MULTI 225, 1248
- REGDB\_UNINSTALL\_COMMENTS 226
- REGDB\_UNINSTALL\_CONTACT 226
- REGDB\_UNINSTALL\_DISPLAY\_VERSION 227
- REGDB\_UNINSTALL\_DISPLAYICON 226
- REGDB\_UNINSTALL\_HELPLINK 227
- REGDB\_UNINSTALL\_HELPTELEPHONE 227
- REGDB\_UNINSTALL\_INSTALLDATE 227
- REGDB\_UNINSTALL\_INSTALLLOC 228
- REGDB\_UNINSTALL\_INSTALLSOURCE 228
- REGDB\_UNINSTALL\_LANGUAGE 228
- REGDB\_UNINSTALL\_LOGFILE 229
- REGDB\_UNINSTALL\_MAINT\_OPTION 229
- REGDB\_UNINSTALL\_MAJOR\_VERSION 229
- REGDB\_UNINSTALL\_MAJOR\_VERSION\_OLD 230
- REGDB\_UNINSTALL\_MINOR\_VERSION 230
- REGDB\_UNINSTALL\_MINOR\_VERSION\_OLD 230
- REGDB\_UNINSTALL\_MODIFYPATH 231
- REGDB\_UNINSTALL\_NAME 231, 1243
- REGDB\_UNINSTALL\_NOMODIFY 231
- REGDB\_UNINSTALL\_NOREMOVE 231
- REGDB\_UNINSTALL\_NOREPAIR 232
- REGDB\_UNINSTALL\_PRODUCTGUID 232
- REGDB\_UNINSTALL\_PRODUCTID 232
- REGDB\_UNINSTALL\_PUBLISHER 232
- REGDB\_UNINSTALL\_README 233
- REGDB\_UNINSTALL\_REGCOMPANY 233
- REGDB\_UNINSTALL\_REGOWNER 233
- REGDB\_UNINSTALL\_STRING 233
- REGDB\_UNINSTALL\_SYSTEMCOMPONENT 234
- REGDB\_UNINSTALL\_URLINFOABOUT 234
- REGDB\_UNINSTALL\_URLUPDATEINFO 234
- REGDB\_UNINSTALL\_VERSION 235
- REGDB\_VALUENAME\_APPPATH 235
- REGDB\_VALUENAME\_APPPATHDEFAULT 235
- REGDB\_VALUENAME\_INSTALL 235
- REGDB\_VALUENAME\_INSTALLSUCCESS 235
- REGDB\_VALUENAME\_SP 235
- REGDB\_VALUENAME\_UNINSTALL\_COMMENTS 236
- REGDB\_VALUENAME\_UNINSTALL\_CONTACT 236
- REGDB\_VALUENAME\_UNINSTALL\_DISPLAYICON 236
- REGDB\_VALUENAME\_UNINSTALL\_DISPLAYNAME 236
- REGDB\_VALUENAME\_UNINSTALL\_DISPLAYVERSION 236
- REGDB\_VALUENAME\_UNINSTALL\_HELPLINK 236
- REGDB\_VALUENAME\_UNINSTALL\_HELPTELEPHONE 236
- REGDB\_VALUENAME\_UNINSTALL\_INSTALLDATE 237
- REGDB\_VALUENAME\_UNINSTALL\_INSTALLLOCATION 237
- REGDB\_VALUENAME\_UNINSTALL\_INSTALLSOURCE 237
- REGDB\_VALUENAME\_UNINSTALL\_LANGUAGE 237
- REGDB\_VALUENAME\_UNINSTALL\_LOGFILE 237
- REGDB\_VALUENAME\_UNINSTALL\_LOGMODE 237
- REGDB\_VALUENAME\_UNINSTALL\_MAJORVERSION 237
- REGDB\_VALUENAME\_UNINSTALL\_MAJORVERSION\_OLD 238
- REGDB\_VALUENAME\_UNINSTALL\_MINORVERSION 238
- REGDB\_VALUENAME\_UNINSTALL\_MINORVERSION\_OLD 238
- REGDB\_VALUENAME\_UNINSTALL\_MODIFYPATH 238
- REGDB\_VALUENAME\_UNINSTALL\_NOMODIFY 238
- REGDB\_VALUENAME\_UNINSTALL\_NOREMOVE 238
- REGDB\_VALUENAME\_UNINSTALL\_NOREPAIR 238
- REGDB\_VALUENAME\_UNINSTALL\_PRODUCTGUID 239
- REGDB\_VALUENAME\_UNINSTALL\_PRODUCTID 239
- REGDB\_VALUENAME\_UNINSTALL\_PUBLISHER 239
- REGDB\_VALUENAME\_UNINSTALL\_README 239
- REGDB\_VALUENAME\_UNINSTALL\_REGCOMPANY 239
- REGDB\_VALUENAME\_UNINSTALL\_REGOWNER 239
- REGDB\_VALUENAME\_UNINSTALL\_SYSTEMCOMPONENT 239
- REGDB\_VALUENAME\_UNINSTALL\_UNINSTALLSTRING 240
- REGDB\_VALUENAME\_UNINSTALL\_URLINFOABOUT 240
- REGDB\_VALUENAME\_UNINSTALL\_URLUPDATEINFO 240
- REGDB\_VALUENAME\_UNINSTALL\_VERSION 240
- REGDB\_VALUENAME\_UNINSTALLKEY 240
- REGDB\_VALUENAME\_WINCURRVER\_REGORGANIZATION 240
- REGDB\_VALUENAME\_WINCURRVER\_REGOWNER 241
- REGDB\_WINCURRVER\_REGORGANIZATION 241
- REGDB\_WINCURRVER\_REGOWNER 241
- RegDBConnectRegistry 1191
- RegDBConnectRegistry Example 1194
- RegDBCopyKeys 1195
- RegDBCopyValues 1198
- RegDBCreateKeyEx 1200
- RegDBCreateKeyEx Example 1202
- RegDBDeleteItem 1203
- RegDBDeleteKey 1207
- RegDBDeleteKey Example 1208
- RegDBDeleteValue 1209
- RegDBDeleteValue Example 1210
- RegDBDisconnectRegistry 1211
- RegDBDisconnectRegistry Example 1212
- RegDBGetAppInfo 1214
- RegDBGetAppInfo Example 1215
- RegDBGetDefaultRoot 1217
- RegDBGetItem 1218
- RegDBGetItem Example 1222
- RegDBGetKeyValueEx 1223
- RegDBGetKeyValueEx Example 1225
- RegDBGetUninstCmdLine 1227
- RegDBKeyExist 1228
- RegDBKeyExist Example 1229
- RegDBQueryKey 1231
- RegDBQueryKey Example 1232
- RegDBQueryKeyCount 1234

[RegDBQueryStringMultiStringCount](#) 1235  
[REGDBREMOTEREGCONNECTED](#) 218  
[RegDBSetAppInfo](#) 1237  
[RegDBSetAppInfo Example](#) 1239  
[RegDBSetDefaultRoot](#) 1240  
[RegDBSetDefaultRoot Example](#) 1241  
[RegDBSetItem](#) 1243  
[RegDBSetItem Example](#) 1246  
[RegDBSetKeyValueEx](#) 1248  
[RegDBSetKeyValueEx Example](#) 1251  
[RegDBSetVersion](#) 1252  
[REGFONT\\_OPTION\\_DEFAULT](#) 241  
[REGFONT\\_OPTION\\_DONTBROADCASTFONTCHANGEMSG](#) 241  
[REGFONT\\_OPTION\\_DONTUPDATEREGISTRY](#) 242  
[RegisterFontResource](#) 1253  
[Registration](#) 1383  
[Registry](#)  
     check whether a key exists 1228  
     company name 1383  
     connecting to remote registry 1191  
     creating registry keys 1200  
     creating registry sets 633  
     default root 1240  
     delete a key's value 1209  
     delete registry keys 1207  
     disconnecting from remote registry 1211  
     getting information from the registry 1214  
     InstallScript functions 470  
     querying keys 1231  
     serial number 1383  
     setting information in the registry 1248  
     user name 1379  
   registry functions 470  
[REGISTRYFUNCTIONS\\_USETEXTSUBS](#) 242  
[REINSTALLMODE](#) 348  
[Relational operators](#) 507  
[ReleaseDialog](#) 1256  
[ReleaseDialog Example](#) 1257  
[Remote registry](#) 1191  
[REMOTE\\_DRIVE](#) 242, 1001  
[REMOVE](#) 242, 1160  
[REMOVEABLE\\_DRIVE](#) 242, 1001  
[REMOVEALL](#) 243  
[REMOVEALLMODE](#) 348  
[REMOVEONLY](#) 348  
[RenameFile](#) 1259  
[RenameFile Example](#) 1261  
[Renaming](#) 1259  
     files 1259  
[REPAIR](#) 243  
[repeat](#) 71  
[REPLACE](#) 243, 789  
[ReplaceFolderIcon](#) 1263

[ReplaceFolderIcon Example](#) 1265  
[ReplaceProfString](#) 1266  
[ReplaceProfString Example](#) 1268  
[ReplaceShortcut](#) 1269  
     example 1272  
[Replacing](#) 793  
     string in initialization file 1266  
     text in batch files 793  
[Reserved words](#) 64  
[RESET](#) 243, 931  
[Resize](#) 1273  
[Response file](#) 1502  
[RESTART](#) 244, 1150  
[Restarting target machine](#)  
     through SdFinishReboot 1335  
[return](#) 72  
[RGB](#) 1274  
[RGB Example](#) 1274  
[ROOT](#) 244, 722  
[RUN\\_MAXIMIZED](#) 244, 1263  
[RUN\\_MINIMIZED](#) 244, 1177  
[RUN\\_SEPARATEMEMORY](#) 1263  
[runas](#) 1030  
[Running a program from a setup script](#) 1025

## S

[Saving](#) 1098  
[Screen dimensions](#) 957  
[Script](#) 56  
[Script files](#) 57  
[Script-created features](#) 457  
     declarations 56  
     identifiers 57  
     overview 55  
     program block 56  
     punctuation rules 57  
     structure 56  
     using white space 58  
[SdAskDestPath](#) 1277  
[SdAskDestPath Example](#) 1278  
[SdAskDestPath2](#) 1279  
[SdAskOptions](#) 1281  
[SdAskOptions Example](#) 1284  
[SdAskOptionsList](#) 1285  
[SdAskOptionsList Example](#) 1287  
[SdBitmap](#) 1288  
[SdBitmap Example](#) 1290  
[SdComponentDialog](#) 438  
[SdComponentDialog2](#) 438  
[SdComponentDialogAdv](#) 438  
[SdComponentMult](#) 438  
[SdComponentTree](#) 438  
[SdConfirmNewDir](#) 1291

- SdConfirmNewDir Example 1292
- SdConfirmRegistration 1294
- SdConfirmRegistration Example 1295
- SdCustomerInformation 1296
- SdCustomerInformation Example 1299
- SdCustomerInformationEx 1300
- SdCustomerInformationEx Example 1303
- SdDiskSpace2 1304
- SdDiskSpace2 Example 1305
- SdDiskSpaceRequirements 1306
- SdDisplayTopics 1307
- SdDisplayTopics Example 1308
- SdExceptions 1310
- SdExceptions Example 1311
- SdFeatureDialog 1312
- SdFeatureDialog Example 1315
- SdFeatureDialog2 1316
- SdFeatureDialog2 Example 1318
- SdFeatureDialogAdv 1319
- SdFeatureDialogAdv Example 1321
- SdFeatureMult 1322
- SdFeatureMult Example 1324
- SdFeatureTree 1325
- SdFeatureTree Example 1327
- SdFilesInUse 1328
- SdFilesInUse Example 1330
- SdFinish 1331
- SdFinish Example 1332
- SdFinishEx 1334
- SdFinishEx Example 1335
- SdFinishReboot 1335
- SdFinishReboot Example 1337
- SdFinishUpdate 1338
- SdFinishUpdateEx 1338
- SdFinishUpdateReboot 1340
- SdFinishUpdateReboot Example 1342
- SdGeneralInit 1342
- SdGeneralInit example 1343
- SdInit 1345
- SdInit Example 1345
- SdLicense 1346
- SdLicense Example 1347
- SdLicense2 1348
- SdLicense2Ex 1351
- SdLicense2Rtf 1354
- SdLicenseEx 1356
- SdLicenseRtf 1359
- SdLoadString 1361
- SdLoadString Example 1362
- SdLogonUserBrowse 1363
- SdLogonUserCreateUser 1363
- SdLogonUserInformation 1363
- SdLogonUserListGroup 1364
- SdLogonUserListServers 1365
- SdLogonUserListUsers 1365
- SdMakeName 1365
- SdMakeName Example 1366
- SdOptionsButtons 1370
- SdOptionsButtons Example 1373
- SdOutOfDiskSpace 1375
- SdPatchWelcome 1376
- SdPatchWelcome Example 1377
- SdProductName 1378
- SdProductName Example 1379
- SdRegisterUser 1379
- SdRegisterUser Example 1382
- SdRegisterUserEx 1383
- SdRegisterUserEx Example 1385
- SdRMFilesInUse 1386
- Sdsadlg.rul 1399
- SdSelectFolder 1388
- SdSelectFolder Example 1389
- SdSetupCompleteError 1390
- SdSetupCompleteError Example 1391
- SdSetupType 1392
- SdSetupType Example 1393
- SdSetupType2 1394
- SdSetupType2 Example 1396
- SdSetupTypeEx 1397
- SdSetupTypeEx Example 1398
- SdShowAnyDialog 1399
- SdShowAnyDialog Example 1400
- SdShowDlgEdit1 1401
- SdShowDlgEdit1 Example 1402
- SdShowDlgEdit2 1403
- SdShowDlgEdit2 Example 1404
- SdShowDlgEdit3 1405
- SdShowDlgEdit3 Example 1408
- SdShowFileMods 1409
- SdShowFileMods Example 1410
- SdShowInfoList 1412
- SdShowInfoList Example 1413
- SdShowMsg 1414
- SdShowMsg Example 1416
- SdStartCopy 1416
- SdStartCopy Example 1418
- SdStartCopy2 1419
- SdSubstituteProductInfo 1422
- SdWelcome 1422
- SdWelcome Example 1423
- SdWelcomeMaint 1424
- SdWelcomeMaint Example 1425
- Search path 1150
- Searching 1575
  - string 1575
  - text files 919
- Secondary shell 1025
- SeekBytes 1426

- SeekBytes Example 1427
- SelectDir 1429
- SelectDir Example 1431
- SelectDirEx 1432
- SelectDirEx Example 1435
- SELECTED\_LANGUAGE 349
- SELECTFOLDER 245
- SelectFolder 1436
- SelectFolder Example 1437
- Selecting 1436
  - features 1370
  - program folders 1436
- SELFREGISTER 245, 1686
- SELFREGISTERBATCH 245, 769
- SELFREGISTRATIONPROCESS 245, 759
- SendMessage 1438
- SendMessage Example 1439
- SERIAL 245, 988
- Serial ports 988
- service functions 472
- SERVICE\_ADAPTER 246
- SERVICE\_ALL\_ACCESS 246
- SERVICE\_AUTO\_START 246
- SERVICE\_BOOT\_START 246
- SERVICE\_CHANGE\_CONFIG 247
- SERVICE\_CONTINUE\_PENDING 247
- SERVICE\_DEMAND\_START 247
- SERVICE\_DIFX\_32 757, 771
- SERVICE\_DIFX\_AMD64 757, 771
- SERVICE\_DIFX\_IA64 757, 771
- SERVICE\_DISABLED 248
- SERVICE\_ENUMERATE\_DEPENDENTS 248
- SERVICE\_ERROR\_CRITICAL 248
- SERVICE\_ERROR\_IGNORE 248
- SERVICE\_ERROR\_NORMAL 249
- SERVICE\_ERROR\_SEVERE 249
- SERVICE\_FILE\_SYSTEM\_DRIVER 249
- SERVICE\_FLAG\_DIFX\_32 250
- SERVICE\_FLAG\_DIFX\_AMD64 250
- SERVICE\_FLAG\_DIFX\_IA64 250
- SERVICE\_FLAG\_ISFONTREG 250
- SERVICE\_INTERACTIVE\_PROCESS 251
- SERVICE\_INTERROGATE 251
- SERVICE\_IS\_PARAMS 281
- SERVICE\_IS\_STATUS 283
- SERVICE\_ISFONTREG 251
- SERVICE\_ISUPDATE 251
- SERVICE\_KERNEL\_DRIVER 252
- SERVICE\_PAUSE\_CONTINUE 252
- SERVICE\_PAUSE\_PENDING 252
- SERVICE\_PAUSED 252
- SERVICE\_QUERY\_CONFIG 253
- SERVICE\_QUERY\_STATUS 253
- SERVICE\_RECOGNIZER\_DRIVER 253
- SERVICE\_RUNNING 254
- SERVICE\_START 254
- SERVICE\_START\_PENDING 254
- SERVICE\_STOP 254
- SERVICE\_STOP\_PENDING 255
- SERVICE\_STOPPED 255
- SERVICE\_SYSTEM\_START 255
- SERVICE\_USER\_DEFINED\_CONTROL 256
- SERVICE\_WIN32\_OWN\_PROCESS 256
- SERVICE\_WIN32\_SHARE\_PROCESS 256
- ServiceAddService 1441
- ServiceExistsService 1443
- ServiceGetServiceState 1443
- ServiceInitParams 1444
- ServiceRemoveService 1445
- ServiceStartService 1446
- ServiceStopService 1447
- SET command 786
  - adding to batch file 786
- set keyword 72
- SetColor 1448
- SetColor Example 1451
- SetDialogTitle 1452
- SetDialogTitle Example 1453
- SetDisplayEffect 1454
- SetDisplayEffect Example 1455
- SetErrorMsg 1457
- SetErrorMsg Example 1458
- SetErrorTitle 1459
- SetErrorTitle Example 1460
- SetExtendedErrInfo 1461
- SetFileInfo 1462
- SetFileInfo Example 1464
- SetFont 1465
- SetFont Example 1466
- SetInstallationInfo 1467
- SetObjectPermissions 1468
- SetObjectPermissions Example 1474
- SetShortcutProperty 1475
  - example 1478
- SetStatus 1479
- SetStatusEx 1480
- SetStatusExStaticText 1481
- SetStatusWindow 1482
- SetStatusWindow Example 1483
- Setting 1448
  - background and status bar 1448
  - color 1274
  - configuration file value 611
  - default batch file name 570
  - default registry root 1240
  - default system configuration file 609
  - dialog titles 1452
  - display effects 1454

- error message box 1459
- feature properties and data 886
- file mode 1135
- setup type 898
- SetTitle 1484
- SetTitle Example 1488
- Setup script 57
  - comments 57
  - declarations 56
  - including 369
  - launching 761
  - limits for 55
  - overview 55
  - program block 56
  - punctuation rules 57
  - structure 56
  - using white space 58
- Setup types 1397
  - getting setup type data 893
  - selecting 1392
  - setting 898
- SETUP\_PACKAGE 258
- Setup.exe 337
  - version 337
- Setup.ini
  - limits for 55
- Setup.inx 761
- Setup.rul
  - limits for 55
- SetUpdateStatus 1488
- SetUpdateStatusReboot 1489
- SETUPTYPE 256
- SetupType 1489
- SetupType Example 1491
- SETUPTYPE\_INFO\_DESCRIPTION 257
- SETUPTYPE\_INFO\_DISPLAYNAME 257
- SETUPTYPE\_STR\_COMPACT 257
- SETUPTYPE\_STR\_COMPLETE 257
- SETUPTYPE\_STR\_CUSTOM 257
- SETUPTYPE\_STR\_TYPICAL 258
- SetupType2 1493
- SetupType2 Example 1495
- SEVERE 258, 1127
- Shared files 472
- SHAREDFILE 258, 1659
- SHAREDSUPPORTDIR 349
- Shell 473
- SHELL\_OBJECT\_FOLDER 349
  - alternate 964
  - getting name 1181
  - InstallScript functions 473
  - secondary 1026
- SHORT data type 289
- Shortcuts 964
  - adding through InstallScript 637
  - configuring Shell properties through InstallScript 1475
  - configuring Shell properties through InstallScript while creating 637
  - configuring through InstallScript 1475
  - deleting through InstallScript 730
  - replacing in folders through InstallScript 1269
- SHOW\_PASSWORD\_DIALOG 350
- ShowObjWizardPages 1497
- ShowProgramFolder 1497
- ShowProgramFolder Example 1498
- ShowWindow 1499
- Silent installations 1502
  - reading the log file 1502
  - silent mode 344
  - writing the log file 1508
- Silent mode 344
- SILENTMODE 259, 344
- SilentReadData 1502
- SilentReadData Example 1504
- SilentWriteData 1508
- SilentWriteData Example 1510
- SizeOf 1514
- SizeWindow 1514
- SizeWindow Example 1516
- SKIN\_LOADED 259
- Sound 1170
  - playing sound files 1170
- Special Registry-Related Functions 475
- Sprintf 62
- Sprintf Example 1517
- SprintfBox 62
- SprintfBox Example 1521
- SprintfMsiLog 1522
- SQL
  - InstallScript functions 477
- SQL\_BATCH\_INSTALL 259
- SQL\_BATCH\_UNINSTALL 259
- SQL\_BROWSE\_ALIAS 259
- SQL\_BROWSE\_ALL 260
- SQL\_BROWSE\_LOCAL 260
- SQL\_BROWSE\_REMOTE 260
- SQL\_ERROR\_GET\_SCHEMA\_VERSION 260
- SQL\_ERROR\_SCRIPT\_COMMAND\_ERROR 260
- SQL\_ERROR\_SCRIPT\_CONNECTION\_NOT\_OPEN 261
- SQL\_ERROR\_SCRIPT\_UNABLE\_OPEN\_FILE 261
- SQL\_ERROR\_SET\_SCHEMA\_VERSION 261
- SQLBrowse 1523
- SQLBrowse2 1524
- SQLDatabaseBrowse 1525
- SQLRTComponentInstall 1526
- SQLRTComponentUninstall 1527
- SQLRTConnect 1528
- SQLRTConnect2 1529



- SQLRTConnectDB 1531
- SQLRTDoRollbackAll 1532
- SQLRTGetBatchList 1533
- SQLRTGetBatchMode 1534
- SQLRTGetBrowseOption 1535
- SQLRTGetComponentScriptError 1537
- SQLRTGetComponentScriptError2 1538
- SQLRTGetConnectionAuthentication 1540
- SQLRTGetConnectionInfo 1541
- SQLRTGetConnections 1542
- SQLRTGetDatabases 1543
- SQLRTGetErrorMessage 1544
- SQLRTGetLastError 1545
- SQLRTGetLastError2 1546
- SQLRTGetScriptErrorMessage 1546
- SQLRTGetServers 1547
- SQLRTGetServers2 1548
- SQLRTInitialize 1549
- SQLRTInitialize2 1550
- SQLRTPutConnectionAuthentication 1551
- SQLRTPutConnectionInfo 1551
- SQLRTPutConnectionInfo2 1552
- SQLRTServerValidate 1553
- SQLRTSetBrowseOption 1555
- SQLRTTestConnection 1556
- SQLRTTestConnection2 1558
- SQLServerLogin 1560
- SQLServerSelect 1561
- SQLServerSelectLogin 1562
- SQLServerSelectLogin2 1563
- SQLServerSelectLoginEx 1566
- SRCDIR 350
- SRCDISK 350
- SRCINSTALLDIR 261
- SRCTARGETDIR 262
- SSP\_PROPERTY\_NO\_NEW\_INSTALL\_HIGHLIGHT 262
- SSP\_PROPERTY\_NO\_STARTSCREEN\_PIN 262
- SSP\_PROPERTY\_PREVENT\_PINNING 262
- STANDARD\_RIGHTS\_ALL 263
- STANDARD\_RIGHTS\_EXECUTE 263
- STANDARD\_RIGHTS\_READ 263
- STANDARD\_RIGHTS\_REQUIRED 263
- STANDARD\_RIGHTS\_WRITE 263
- Start Programs menu 327
- Startup 327
  - folder 327
- STATUS 264, 769
- Status bar 1448
- Status, setting object status 1480
- STATUSBAR 264, 1448
- STATUSBBD 264
- STATUSDLG 264, 769
- STATUSEX 265, 754
- STATUSOLD 265, 754
- StatusUpdate 1567
- StatusUpdate Example 1569
- stdcall 73
- step 66
- StrAddLastSlash 1570
- StrCompare 1571
- StrCompare Example 1572
- StrConvertSizeUnit 1573
- StreamFileFromBinary 1574
- StrFind 1575
- StrFind Example 1575
- StrFindEx 1576
- StrGetTokens 1577
- StrGetTokens Example 1578
- STRING data type 289
- String Variables 305
- STRINGLIST 265, 1051
- Strings 1584
  - adding to batch file 789
  - adding to configuration file 798
  - appending to path 498
  - changing case 1595
  - comparing 1571
  - comparing versions 1638
  - concatenating 503
  - constants 294
  - convert a number to a string 1131
  - convert a string to a character 1589
  - convert a string to a number 1592
  - converting unit constants to 1573
  - copy substrings 616
  - create formatted string 1516
  - embedding quotation marks 61
  - find substring 509
  - getting drive designation from path 945
  - getting substring 1587
  - indexing 306
  - inserting special characters 60
  - InstallScript functions 482
  - length (bytes) 1580
  - length (chars) 1581
  - operators 508
  - parsing 1577
  - parsing path 1139
  - removing a trailing backslash 1584
  - removing drive designation from path 943
  - removing leading and trailing spaces and tabs from 1597
  - sizing 306
  - string constant operator 508
  - string entries 294
- StrLength 1580
- StrLength Example 1580
- StrLengthChars 1581
- StrLengthChars Example 1582

StrPutTokens 1583  
 StrRemoveLastSlash 1584  
 StrRemoveLastSlash Example 1585  
 StrReplace 1586  
 StrSub 1587  
 StrSub Example 1588  
 STRTOCHAR 1589  
 StrToLower 1590  
 StrToLower Example 1591  
 StrToNum 1592  
 StrToNum Example 1593  
 StrToNumHex 1594  
 StrToUpper 1595  
 StrToUpper Example 1596  
 StrTrim 1597  
 Structure pointer operator 509  
 STYLE\_BOLD 265, 967  
 STYLE\_ITALIC 265, 967  
 STYLE\_NORMAL 266, 967  
 STYLE\_SHADOW 266, 1465  
 STYLE\_UNDERLINE 266, 967  
 Substring 616  
     copying 616  
     finding 509  
     getting 1587  
 SUITE\_HOSTED 286  
 Suite/Advanced UI interaction InstallScript functions 483  
 SuiteFormatString 1598  
 SuiteFormatString Example 1599  
 SuiteGetProperty 1600  
 SuiteGetProperty Example 1601  
 SuiteLogInfo 1602  
 SuiteLogInfo Example 1603  
 SuiteReportError 1604  
 SuiteResolveString 1605  
 SuiteResolveString Example 1606  
 SuiteSetProperty 1607  
 SuiteSetProperty Example 1608  
 SUPPORTDIR 350  
 SW\_MAXIMIZE 266, 1497  
 SW\_MINIMIZE 267, 1497  
 SW\_RESTORE 267, 1497  
 SW\_SHOW 267, 1497  
 switch 73  
 SYNCHRONIZE 267  
 Syntax 61  
     identifiers 57  
     punctuation rules 57  
     quotation marks in strings 61  
 SYS\_BOOTMACHINE 267, 1335  
 SYS\_BOOTWIN 1189  
 SYSINFO 351  
 SYSPROCESSORINFO 357  
 System 601

System (InstallScript function) 1609  
 System Example 1610  
 System variables 327  
     configuration file 801  
     DISK1SETUPEXENAME 322  
     DISK1TARGET 322  
     ERRORFILENAME 323  
     files 1462  
     FOLDER\_DESKTOP 324  
     FOLDER\_PROGRAMS 327  
     FOLDER\_STARTMENU 327  
     FOLDER\_STARTUP 327  
     INFOFILENAME 334  
     information 988  
     IS\_NULLSTR\_PTR 336  
     ISRES 336  
     ISUSER 336  
     ISVERSION 337  
     MEDIA 343  
     MODE 344  
     overview 307  
     PRODUCT\_GUID 345  
     PROGRAMFILES 345  
     REMOVEONLY 348  
     SELECTED\_LANGUAGE 349  
     SHARED SUPPORTDIR 349  
     SHELL\_OBJECT\_FOLDER 349  
     SRCDIR 350  
     SRCDISK 350  
     SUPPORTDIR 350  
     TARGETDIR 359  
     TARGETDISK 359  
     UNINST 359  
     UNINSTALL\_STRING 361  
     WINDIR 361  
     WINDISK 361  
     WINSYSDIR 362  
     WINSYSDISK 363  
 SYSTEM\_DPI 268  
 SYSTEM\_DPI\_SCALING 268

## T

Tab character 60  
 TARGETDIR  
     InstallScript variable 359  
 TARGETDISK 359  
 TBYTES 268  
 Text files 922  
     closing 579  
     creating 626  
     deleting lines 916  
     inserting lines 922  
     opening 1133



- reading 971
- searching 919
- writing 1677
- Text substitution 484
- TextSubGetValue 1610
- TextSubGetValue Example 1611
- TextSubParseTextSub 1612
- TextSubParseTextSub Example 1613
- TextSubSetValue 1614
- TextSubSetValue Example 1615
- TextSubSubstitute 1616
- TextSubSubstitute Example 1617
- then 68
- TILED 268, 1160
- TIME 268, 988
- Title 1452
  - dialog 1452
- to 66
- Transferring files 1686
  - CopyFile 620
  - features 871
  - XCopyFile 1686
- TRUE 269, 1114
- TTFONTFILEINFO\_FONTTITLE 269
- typedef 295
- TYPICAL 270, 1489

## U

- Unary arithmetic operators 500
- UNINST 359
- UNINSTALL\_DISPLAYNAME 360
- UNINSTALL\_STRING 361
- UninstallApplication 1619
- Uninstallation Functions 485
- Uninstaller 720
  - abort 65
  - enabling 720
  - registry 1200
  - uninstallation key 360
- UNINSTALLKEY 360
- until 71
- UnUseDLL 1621
- UnUseDLL Example 1621
- UPDATE\_SERVICE\_INSTALL 270
  - used with Disable 757
  - used with Enable 772
- UPDITEMODE 361
- UpdateServiceCheckForUpdates 1623
- UPDATESERVICECOMPONENT 270
- UpdateServiceCreateShortcut 1623
- UpdateServiceEnableUpdateManagerInstall 1624
- UpdateServiceGetAgentTarget 1624
- UpdateServiceOnEnabledStateChange 1624
- UpdateServiceRegisterProduct 1624
- UpdateServiceRegisterProductEx 1625
- UpdateServiceSetHost 1625
- UpdateServiceSetLanguage 1625
- UPPER\_LEFT 270, 1166
- UPPER\_RIGHT 270, 1166
- URL 271
- URLs 1142
  - parsing 1142
- USE\_LOADED\_SKIN 272
- UseDLL 1625
- UseDLL Example 1628
- User interface
  - InstallScript functions 486
- User name 1294
- USER\_ADMINISTRATOR 271, 1012
- USER\_INADMINGROUP 271
- USER\_POWERUSER 271, 1012
- USERPROFILE 1177
  - registering 1294
- Using white space 58

## V

- VALID\_PATH 272, 1012
- Validating features 909
- Variables 307
  - declaring 303
  - local vs. global 303
  - scope 303
  - system 327
- VARIANT data type 289
- VarInit 1629
- VarRestore 1630
- VarRestore Example 1633
- VarSave 1634
- VarSave Example 1636
- VarSave Stack Example 1637
- VER\_DLL\_NOT\_FOUND 274, 1655
- VER\_NT\_DOMAIN\_CONTROLLER 351
- VER\_NT\_SERVER 351
- VER\_NT\_WORKSTATION 351
- VER\_SUITE\_BACKOFFICE 351
- VER\_SUITE\_DATACENTER 351
- VER\_SUITE\_ENTERPRISE 351
- VER\_SUITE\_PERSONAL 351
- VER\_SUITE\_SMALLBUSINESS 351
- VER\_SUITE\_SMALLBUSINESS\_RESTRICTED 351
- VER\_SUITE\_TERMINAL 351
- VER\_UPDATE\_ALWAYS 274, 1659
- VER\_UPDATE\_COND 274, 1655
- VER\_UPDATE\_CONDFILE\_INSTALLED 1659
- VerCompare 1638
- VerCompare Example 1639

[VerFindFileVersion](#) 1641  
[VerFindFileVersion Example](#) 1642  
[VerGetFileLanguages](#) 1644  
[VerGetFileVersion](#) 1646  
[VerGetFileVersion Example](#) 1647  
[VerProductCompareVersions](#) 1648  
[VerProductGetInstalledVersion](#) 1649  
[VerProductIsVersionSupported](#) 1650  
[VerProductNumToStr](#) 1651  
[VerProductStrToNum](#) 1652  
[VerProductVerFromVerParts](#) 1653  
[VerProductVerPartsFromVer](#) 1654  
[VerSearchAndUpdateFile](#) 1655  
[VerSearchAndUpdateFile Example](#) 1658  
[Version checking](#) 1641
 

- [comparing versions](#) 1638
- [finding file version and location](#) 1641
- [getting files based on version](#) 1646
- [installing newer versions of files](#) 1655
- [InstallScript functions](#) 487
- [product version](#) 1648

[VERSION\\_COMPARE\\_RESULT\\_NEWER](#) 272  
[VERSION\\_COMPARE\\_RESULT\\_NEWER\\_NOT\\_SUPPORTED](#) 272  
[VERSION\\_COMPARE\\_RESULT\\_NOT\\_INSTALLED](#) 273  
[VERSION\\_COMPARE\\_RESULT\\_OLDER](#) 273  
[VERSION\\_COMPARE\\_RESULT\\_SAME](#) 273  
[VERSION\\_PREVIOUS\\_VERSION\\_DELIMITER](#) 273  
[Version,](#) 1648  
[VerUpdateFile](#) 1659  
[VerUpdateFile Example](#) 1663  
[VIDEO](#) 274, 988  
[Video](#) 1170
 

- [adapter type](#) 988
- [displaying animation files](#) 1170

[VIRTUAL\\_MACHINE\\_TYPE](#) 275  
[Visual interface](#) 1567
 

- [disabling elements](#) 754
- [enabling elements](#) 769
- [getting window handle](#) 933
- [placing bitmaps](#) 1160
- [placing window](#) 1166
- [playing video and sound](#) 1170
- [progress indicator](#) 1482
- [setting background and status bar color](#) 1448
- [setting custom colors](#) 1274
- [setting dialog titles](#) 1452
- [setting display effects](#) 1454
- [setting error message box](#) 1457
- [setting fonts](#) 1465
- [setting main window title](#) 1484
- [sizing objects](#) 1514
- [updating the progress indicator](#) 1567

[void](#) 74  
[Volume label](#) 988

[VOLUMELABEL](#) 275, 988

## W

[WAIT](#) 761  
[WaitForApplication](#) 1664  
[WaitOnDialog](#) 1667  
[WaitOnDialog Example](#) 1668  
[WARNING](#) 275, 1518  
[WAV files](#) 1170  
[WEB\\_BASED\\_SETUP](#) 275  
[WELCOME](#) 275  
[Welcome](#) 1670  
[Welcome Example](#) 1671  
[while](#) 75  
[WHITE](#) 276, 1484  
[WILL\\_REBOOT](#) 276, 1335  
[WINDIR](#) 361  
[Windir environment variable](#) 1177  
[WINDISK](#) 361  
[Window](#) 1484
 

- [getting handle for](#) 1004
- [placing](#) 1166

[Windows](#) 1026  
[Windows Installer APIs](#) 488
 

- [sample script](#) 495

[Windows Installer Functions](#) 488  
[Windows NT](#) 1211
 

- [adding icons](#) 519
- [administrator](#) 1012
- [GetProfInt](#) 976
- [InstallShield system variables](#) 327
- [program group type](#) 1175
- [remote registry](#) 1211
- [Service Pack number](#) 1021
- [setting group type](#) 1175
- [USERPROFILE](#) 1177

[WINDOWS\\_SHARED](#) 276, 1012
 

- [API](#) 1518
- [disk](#) 361
- [DLLs](#) 1625
- [folder](#) 361
- [identifying shell](#) 1181
- [InstallShield system variables](#) 327
- [restart](#) 1189
- [setting main window title](#) 1484
- [System folder](#) 363
- [windir environment variable](#) 964

[WINMAJOR](#) 276, 988  
[WINMINOR](#) 276, 988  
[WINSYSDIR](#) 362  
[WINSYSDIR64](#) 362  
[WINSYSDISK](#) 363  
[WizardDirection](#) 1672

WM\_COMMAND [1667](#)  
Working with Batch Files [437](#)  
WOW64FSREDIRECTION [277](#)  
WRITE\_DAC [277](#)  
WRITE\_OWNER [277](#)  
WriteArrayProperty [1673](#)  
WriteBoolProperty [1674](#)  
WriteBytes [1675](#)  
WriteBytes Example [1676](#)  
WriteLine [1677](#)  
WriteLine Example [1678](#)  
WriteNumberProperty [1680](#)  
WriteProfInt [1680](#)  
WriteProfInt Example [1682](#)  
WriteProfString [1683](#)  
WriteProfString Example [1684](#)  
WriteProperties [515](#)  
WriteStringProperty [1685](#)  
Writing [1098](#)

- binary files [1675](#)
- initialization files [1683](#)
- text files [1677](#)

WSTRING data type [289](#)

## X

XCopyFile [1686](#)  
XCopyFile Example [1691](#)

## Y

YELLOW [277](#), [1484](#)  
YES [278](#), [544](#)

