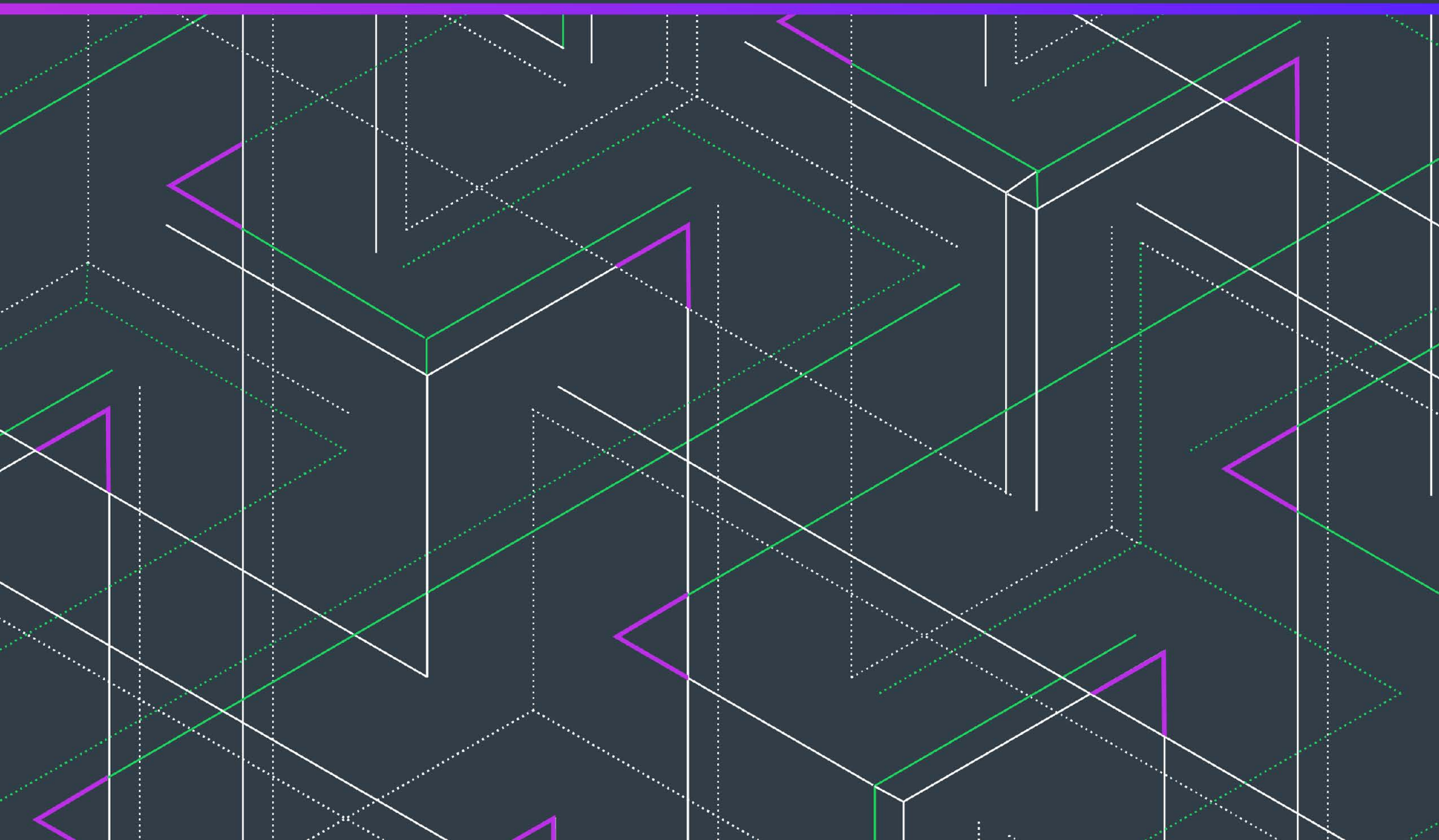
revenera

# InstallShield 2025

## InstallScript Debugger User Guide



# Legal Information

|  |  |
|---|---|
| **Book Name:** | InstallShield 2025 InstallScript Debugger User Guide |
| **Part Number:** | ISP-3100-UG00 |
| **Product Release Date:** | July 2025 |

## Copyright Notice

## Intellectual Property

## Restricted Rights Legend

# Contents

# Debugging InstallScript-Based Installations

Debugging is the process of finding and correcting logic errors in computer software. Those are the errors that can cause a script to operate incorrectly or to halt unexpectedly. Unlike syntax errors, logic errors are not detected by the compiler, which ensures only that statements are *expressed* correctly. Statements with logic errors look perfectly fine to the compiler. It cannot tell whether or not the instructions expressed by those statements are complete and correct.

## Detecting Logic Errors

The best way to detect logic errors in a completed script is to execute that script and observe its operation. Most serious logic errors will turn up while you're developing and testing a script; however, some logic errors can remain hidden in scripts long after they are put into use, only to surface unexpectedly when the script is run on a particular computer or when a user selects a certain sequence of functions or inputs a particular value.

Logic errors can be fatal, bringing the script to an abrupt halt and displaying a run-time error message to the user. Or they can be subtle; the script executes, but there are problems with its appearance and/or behavior. Perhaps it installed files in a folder other than the one selected or it may not have placed a shortcut on the desktop or in the Windows Start menu, even though that option was selected during the setup.

## Resolving Logic Errors

Once you have observed a bug or received a report of a bug, you should follow the steps below.

1. Run the setup and reproduce the error. Be sure you understand the nature of the error before going on.
2. Review your script and identify the probable location of the error.
3. Use the **InstallScript Debugger** to analyze the error.
4. Use the **InstallScript** view to correct the error in your script.
5. Recompile your script and verify that the error has been corrected.

# InstallScript Debugger

The InstallScript Debugger is a source-level debugger. It displays debugging controls and your installation script in different panes of the same window. In the script pane of the window, the statement to be executed next is indicated with a visual marker, called the execution point.

From the InstallScript Debugger you can execute your script, statement by statement, and trace the flow of control by watching the execution point as it moves in the script pane of the InstallScript Debugger window. You can also monitor the value of any variable in your script at any point during script execution. With these methods, you can more easily identify sources of script error and inefficiency.

## Script Window

The script window in the InstallScript Debugger displays your script so that you can view it as you run it. The next statement to be executed is positioned in the window and indicated by a yellow arrow. Lines with breakpoints have a red circle on the left. You cannot set and remove breakpoints by double-clicking a line.

The title bar of the script window shows the full name of the setup file. The script window scrolls automatically if necessary when you use the Step Into or Step Over commands to trace through a set of statements. Likewise, when script execution is halted at a breakpoint, the window is scrolled automatically to display the location of the breakpoint.

## Watch Window

The Watch window in the InstallScript Debugger displays the current value of each string and numeric variable that you have selected to watch. The Watch window is displayed in the lower-right corner of the InstallScript Debugger. You can add and delete variables from this window at any time while you are debugging.

## Button Controls

The buttons located on the InstallScript Debugger toolbar enable you to control the execution of the script.

**Table 1 •** InstallScript Debugger Button Descriptions

| Button | Description |
|---|---|
| **Open** | Allows you to open a script file. |
| **Toggle Breakpoint** | The toggle breakpoint button allows you to place or remove a breakpoint. |
| **Go** | Begins script execution. Use this button to execute to the next breakpoint or to the end of the script if there are no remaining breakpoints. |
| **Break** | Allows you to set and clear breakpoints. |

**Table 1 ▪** InstallScript Debugger Button Descriptions (cont.)

| Button | Description |
|---|---|
| **Step Into** | Executes the next statement in the script. If that statement specifies a call to a user-defined function, the debugger displays that function in the code window and positions the execution point at the statement following the keyword begin. |
| **Step Over** | Executes the next statement in the script. If that statement specifies a call to a user-defined function, all of the statements in that function are executed and the execution point is positioned at the next statement to be executed after the function call. |
| **Step Out** | Skips the current routine. |
| **Show Next Statement** | Displays the next statement in your script file. |

## User Variables

The User variables section of the code window consists of controls that enable you to inspect the current value of local variables and to add local variables to the Watch window.

# Reproducing the Error

The first goal in the debugging process is to reproduce the bug by running the script and following the steps that you believe will lead to the error. This step is crucial for several reasons:

- If the bug has been reported to you by an end user, the details of the report may be inaccurate or incomplete. If so, you may waste time looking for the wrong problem in the wrong place.
- The bug itself may be one that occurs only in a specific hardware and software environment. If a user reports a bug that you can't reproduce, you'll want to determine how the user's system differs from yours before going any further.
- If you yourself observed the bug, you'll want to watch it happen again, when you're expecting it and can document the exact circumstances under which it occurred.
- The process of reproducing the bug will very likely give you insights into where and why the bug is occurring.
- If you don't know with certainty how to produce the bug, you can't know later for certain that you've actually corrected the problem.

# Locating the Source of the Error

The second step in the debugging process is to locate those parts of the setup that may be involved in the error.

If you have observed the symptoms yourself, you may already have a pretty good idea of where to look for the cause. Still, before you start debugging, it is wise to take a moment to think about the error and to identify obvious design or coding mistakes that could have caused it.

Even the simplest errors can often lead you to several different parts of your setup. For example, suppose the product name is not showing up in Sd dialogs. The error might be found where those dialogs are called. Or it may be that the initialization section fails to establish the product name by calling **SdProductName**. Or it could be that the call to **SdProductName** references an undefined string entry.

You probably would not need the debugger to handle that particular bug. But in cases where you would, you will make better use of the debugger if you have planned which parts of the script to trace and analyze for clues.

## Analyzing the Cause of the Error

Without a good debugging tool, you would be forced to analyze the code at a potential error location in your script by "running" it in your head. You'd read each line in the suspect area carefully, following the logic and keeping track of the variables that came into play there. You would do that until you found the exact place where your script wandered off the path. Then you would do it again (and again) until you could explain exactly why it was behaving as it was. Because that is the objective of this debugging step: finding the exact cause of the error.

A software debugger is a tool designed specifically for this part of the debugging process. A debugger does not analyze your script for you, but it does make it easier for you to find the reason your setup is misbehaving. With the debugger, you can follow the flow of control by executing your script one statement at a time. And you can determine the value of any variable in your script at any point during script execution.

## Correcting the Error

Once you have found the exact source of the error in your script, you are ready to close the debugger and fix the problem. To do that, you will have to make some changes to your installation project in InstallShield. Start by planning your correction, then use the InstallScript view to make the changes. Finally, recompile your script.

## The Execution Point

The execution point is the location of the *next* statement to be executed in the script. That location is displayed in the script window with indicating attributes. When you trace through a script using the Step Into or Step Over buttons, the movement of the execution point reveals the flow of control.

## Step Controls

Step controls include the Step Into and Step Over buttons. These controls allow you to execute all or parts of your script one statement at a time. Use these controls to find help find bugs:

- Step through suspect areas of your script and watch the order in which statements are executed. This process can reveal bugs that are caused by conditional expressions that do not resolve as intended.
- After executing a statement with a step control, check the value of the variables you believe may be causing the bug. The easiest way to track variables is to add them to the Watch window. Verify that the variables you're tracking contain the values you expect them to have at each point in your analysis.

- Use step controls in conjunction with breakpoints. First, set a breakpoint at the statement that starts the section of your script that you want to analyze. Then, click the Go button to execute to that statement. Finally, click the Step Into or Step Over buttons to trace through the statements that follow.

## Stepping into a User-Defined Function

When the execution point is at a statement that calls a user-defined function, click the Step Into button, which is located on the InstallScript Debugger's toolbar. The script window will scroll automatically to display the first executable statement in the user-defined function. That statement is not executed, but it does become the execution point. The statement will be executed next when you click the Step Into, Step Over, or Go button.

***Note** ▪ When you step into a function that is called from another file, such as a DLL, the title bar displays the name of that file.*

## Stepping over a User-Defined Function

When the execution point is at a statement that calls a user-defined function, click the Step Over button, which is located on the InstallScript Debugger's toolbar. All of the statement in the user-defined function will be executed. The execution point then moves to the next statement in the current block.

# InstallScript Breakpoints

A breakpoint is a location in your script where execution will stop when the script is run under the InstallScript Debugger’s control. You can set breakpoints by clicking in the left margin near the statement at which you want execution to stop. Once a breakpoint has been set, it remains in effect until it is cleared or the InstallScript Debugger is closed.

The role of breakpoints in debugging is to interrupt execution at those places in the script where you want to make a close analysis. Once the script has executed to a breakpoint, you can then step through the statements that follow and watch the variables that are integral to that section of the script. To execute to the next breakpoint, click the Go button on the toolbar.

Statements with breakpoints are displayed with indicating attributes. When the script is executed under debugger control, execution stops immediately *before* a statement that has been set as a breakpoint.

## Setting a Breakpoint by Clicking the Margin Near a Source Line

***Task*** ***To set a breakpoint by clicking a source line:***

1. Scroll through the script window to display the line at which you want to set a breakpoint.
2. Click the left margin near the line.

The InstallScript Debugger marks the line with a breakpoint indicator (red circle).

***Note*** ▪ *If you click a breakpoint (red circle) that has already been set, the InstallScript Debugger clears that breakpoint.*

## Setting a Breakpoint at a Function

***Task*** ***To set a breakpoint at a function:***

1. In the script window, click the left margin next to the first line of the function before which you would like to set a breakpoint.
2. When you run your script in the debugger, the debugger will stop at this breakpoint.

When the breakpoint has been set, the breakpoint itself is set at the first executable line of code in the function.

***Note*** ▪ *You cannot set a breakpoint at a built-in InstallScript function.*

## Executing to a Breakpoint

To execute to the next breakpoint, click the Go button, which is located on the InstallScript Debugger’s toolbar. If you click the Go button when no breakpoints have been set, or when the script has already executed to a statement that follows the last breakpoint, the script will execute to the end.

## Clearing a Breakpoint

Use one the following methods to clear breakpoints in your script:

- Clear a breakpoint by clicking red circle in the left margin.
- On the **Debug** menu, click **Clear All Breakpoints**.

# Inspecting, Watching, and Modifying Script Data

Some of the most common software bugs occur because a variable does not hold the correct value at a key point in the program. That can happen for a variety of reasons:

- The variable was not initialized.
- The variable was not updated.
- An incorrect value was assigned to the variable.

The repercussions of these kinds of errors can be enormous and varied. For example, if parts of a script are executed only when a variable is set to a specific value, your installation will be incomplete if that controlling variable is set incorrectly. If the variable is used to control a *while loop*, that loop may never be executed, or it may execute forever, hanging the installation.

To fully analyze a bug, you must be able to investigate the value of script data as you trace through the script. The InstallScript Debugger provides you with three ways to do so:

1. Inspect a variable to determine its current value. At any breakpoint or while stepping through the script, you can check the value of any global variable in the script. When the execution point is within a user-defined function, you can also check any variables that are local to that function.
2. Watch a variable to observe any changes in its value as you use the Step In and Step Over commands to trace script execution. The control window includes a watch window that lets you insert one or more variables that you want to monitor in your debugging session.
3. Change the value of a variable at a key point in the program to test your theories about the role of that variable in the bug that you are analyzing or to override the effects of a logic error that you discovered in your script.

## Inspecting a Variable

To inspect a variable, locate the variable that you want to inspect in the variable window. In this window, the name of the variable is detailed, followed by the current local value of the variable.

### Restrictions

Structured variables and list variables cannot be inspected.

The local user variable controls are active only when the execution point is within the begin and end statements of a user-defined function. When the execution point moves beyond the end of the function, the value shown in the Watch window for that variable is changed to the message <variable not found.>

## Watching a Variable

The Variable window lists all of the local variables in the current user-defined function.

You can type the name of a global variable into the Watch window to see its current value.

### Restrictions

- Structured variables cannot be selected for the **Watch** window. To watch the value of a structure member, return to the script editor and add statements that assign the value of that member to a simple data type at locations in your script where you want to know its value. Then recompile your script and start the debugger. Finally, put that simple variable in the **Watch** window.
- A List variable can be selected for the **Watch** window, but the debugger cannot display its elements. Use the method described above for structure members to watch elements of a list.
- The local user variable controls are active only when the execution point is within the begin and end statements of a user-defined function. When the execution point moves beyond the end of the function, the value shown in the **Watch** window for that variable is changed to the message <variable not found>.

### Changing the Value of a Variable

In the InstallScript Debugger, a list of variables is displayed in the Variable window.

Click on the variable in the list that you want to change. Enter the value that you would like to change the variable to.

### Watching Return Values from Built-In Functions

In the Watch window, type in LAST_RESULT. This system variable contains the value returned by the most recent call to an InstallScript function.

### Deleting a Variable from the Watch Window

***Task*** ***To delete a variable from the Watch window:***

1. In the Watch window, click the variable that you want to delete.
2. Press the **Delete** key.

## Stopping a Script that Is in an Endless Loop

To stop a script that is in an endless loop:

1. To change the focus from your installation's window to the InstallScript Debugger's windows, press ALT+TAB.
2. In the debugger's window, click the **Stop** button.

## Syntax Errors

Syntax errors are language usage errors, such as misspelled keywords, references to undeclared variables, and missing semicolons at the end of statements. Syntax errors prevent script compilation and are reported by the compiler. The debugger has no role in detecting syntax errors. Your program must be free of syntax errors and compile successfully before it can by run under the control of the debugger.

## Logic Errors

Logic errors are found in a script's algorithm, that is, in the structure of the script. Logic errors produce run-time errors, so called because they become apparent only when the script is executed. Some logic errors produce run-time errors that cause a script to terminate. Others simply cause your script to operate incorrectly or to stop responding to input. These are the kinds of errors the debugger can help you help you analyze.

# Debugging an Installation on Any Computer

In order to find bugs that are occurring only with certain hardware or software configurations, you may need to debug your installation on a system other than your development machine. In that case, it is not necessary to install InstallShield on that computer.

***Task*** ***To debug the installation on a debug machine, instead of on your installation development machine:***

1. Compile and build your installation on the development machine.
2. Move copies of the following files from your development machine to the debug machine.
    - `InstallShield Program Files Folder\System\ISDbg.exe`

      This is the executable file for the InstallScript Debugger.
    - `InstallShield Program Files Folder\System\SciLexer.dll`

      This file must be present on the debug machine in the same folder as the `ISDbg.exe` file.
    - *`InstallShield Program Files Folder`*`\Program\0409\ISDbg.chm` (for the English version of InstallShield) or *`InstallShield Program Files Folder`*`\Program\0411\ISDbg.chm` (for the Japanese version of InstallShield)

      This is the help file that you can optionally copy to the debug machine so that you can access the InstallScript Debugger help from within the debugger.
3. Register the `ISDbg.exe` file by passing it the `/REGSERVER` command-line parameter.
4. Copy your project file and your project folder to the debug machine. This should include your built setup disk images (`Setup.exe` and media and support files from the `Disk Images\DiskN` folders).

    ***Tip*** *▪ If the project files on your development machine are accessible to the debug machine over a network, you can skip step 4.*
5. If the debug machine does not have Visual Studio 2012 installed, then you must install the redistributable Visual C++ Redistributable for Visual Studio 2012 on the debug machine.

    ***Note*** *▪ The Visual C++ Redistributable Packages install runtime components of Visual C++ libraries that are required to run applications (such as InstallShield) that were developed using Visual Studio 2012.*
6. Launch `Setup.exe` with the `/d` command-line parameter to debug the installation and provide the location of the debug symbols (.dbg) file. For example, if `Setup.dbg` is in `C:\Test`, type the following statement at the command line:

    ```
    setup /d"C:\Test"
    ```

    Note that if your installation, script, and debug files are all in their original location on the development system, you do not need to specify the path to the debug file—use the following:

    ```
    setup /d
    ```

## Debugging a DLL Function Called from the Installation Script

*Task* ***To debug a DLL function called from your setup script:***

1. Open the Microsoft Visual Studio project that contains your DLL.
2. Shut down any setups that may be running. Also, verify that no Setup.exe processes are running by using Windows' Task Manager. If any Setup.exe processes are running, end them.
3. In the DLL project's project settings' Debug tab, specify in the "Executable for debug session" field the location of your setup's Setup.exe: `<project folder>\Media\<media name>\Disk Images\Disk1\Setup.exe`.
4. Specify the following in the "Program arguments" field:

   ```
   -deleter -d
   ```
5. Press **F5** to launch Setup.exe and begin debugging. The debugger will launch Setup.exe. The setup will initialize and run. The debugger should stop at any breakpoints set in your DLL function.

## Correcting Identified Errors

The InstallScript Debugger neither identifies errors nor edits code. Only you can find errors that escaped the compiler, and only an editor can actually change the code.

When running the debugger, you may have the Script Editor open, and it may contain the same file (probably `Setup.rul`) as does the debugger's code window. When you stop the debugger and go to edit your code, you must do so in the IDE. If another file is there, or that window isn't open, you must set it up for editing the offending script.

When you have made the necessary changes, you must:

- Recompile the script (select Compile `Setup.rul` from the Build menu).
- Rebuild the setup.
- Debug the script.
- Adjust breakpoints and variables, if appropriate, to include the changes just made.

## Testing the Corrected Installation

Your job is not done until you have tested your revised script and verified that the reported bug is gone and the setup now operates as intended. To do that, execute the script exactly as you did when you verified the bug. Watch carefully to be sure that your modifications did not introduce a new bug.

# Troubleshooting InstallScript Debugger Issues

If the debugger does not appear when you select Debug InstallScript from the Build menu, check for the presence of the following files:

**Table 2 ▪** Files and Descriptions

| File | Location |
|---|---|
| **ISDbg.exe** | *InstallShield Program Files Folder*\System<br><br>This file must be registered.If it is not, register it with the `/REGSERVER` command-line parameter. |
| **SciLexer.dll** | *InstallShield Program Files Folder*\System |
| **Setup.dbg** | *InstallShield Project Folder*\Project Name\Script Files |
| **Setup.rul** | *InstallShield Project Folder*\Project Name\Script Files |
| **Setup.inx** | *InstallShield Project Folder*\Project Name\Script Files |
| **Setup.exe** | *InstallShield Project Folder*\Project Name*Product Configuration**Release Name*\DiskImages\Disk1 |

`ISDbg.exe` and `SciLexer.dll` are installed with InstallShield. `Setup.rul` is the primary script that you made for your project. InstallShield creates `Setup.exe` and `Setup.inx` as part of the build process.

## Displaying Custom Dialogs

One common technical support issue is the failure of custom dialogs to be displayed. Building and debugging custom dialogs can be a sophisticated process, but you can often find the bug that prevents a custom dialog from being displayed by confirming that all of the following conditions are true:

- The DLL containing the dialog resides on the disk that you intend to ship.
- Your installation copies the DLL to the path and subfolder where your script expects to find it.
- The dialog that you are loading is in the DLL.
- You are using the correct ID to address the dialog if the dialog is in the DLL.

## Creating Program Folders and Shortcuts

If you are having difficulty creating program folders and shortcuts, isolate the section of your script in which you create them and run it separately.

Use the **SprintfBox** function to display the parameters of your **AddFolderIcon** function. Make sure that all of the values that are passed to the parameters are valid and in the correct order.

## Switching Disks

Your script may encounter problems when it attempts to prompt the user to remove one disk and insert the next. For example, you may get a "File not found" error message at the point where the user is supposed to take out Disk 1 and insert Disk 2.

Usually when you encounter such a problem, it occurs because you left a file open on your diskette. DOS will raise this error message when it finds a new disk volume ID or file allocation table and cannot access the open file.

To avoid this problem, do *not* open files directly on the diskette during your setup. Do not use your information files, bitmap files, or DLLs directly from diskette. Instead, copy the files to the target hard disk and access them from the hard disk.

If you encounter such a problem while debugging, verify that you do not open any files before transferring them to the target disk.

## Before You Call Support

Try to isolate the problem by determining from your user exactly what is wrong. Are many of your customers reporting the problem, a few, or only this customer? Here are some areas to check in diagnosing your customer's problem:

- Has the user tried to install your software more than once? If not, ask the user to install it again on the same system. If there is still a problem, ask the user to try installing it on another system or on several, if possible. It is unreasonable to establish conclusions using only one system.
- Is the problem related to the distribution media? If you ship more than one type of media, ask the user to try the different types. This may help identify a disk problem or a drive problem. If the same problem occurs on both sets of disks, you have significantly narrowed down the problem. If you are confident there is nothing wrong with your setup, send the user another set of disks.
- Use brand-name distribution media whenever possible. If you had disk problems before, you know it does not pay to cut corners when you buy disks. A good disk costs only ten or fifteen cents more than a disk of marginal quality, but the difference between a happy customer and an annoyed one is all the difference in the world.
- If you determine that your customer's problem occurs on only one system, you need to find out how that system differs from others.
- Ask the customer how much memory and system resources are free. If the system resources are extremely low, it is possible some other program is using them. Ask the user to exit Windows to free system resources. If there is ample memory and system resources free after restarting Windows, ask the user to try the setup program again.
- Determine what version of Windows the user is running. Setup.exe typically requires a minimum of 16 MB of RAM on the system. However, it is possible that if you are using an extensive script and large bitmaps, you will require more RAM. If your customer is pushing these limits, they will need more RAM.
- Look at the Config.sys and Autoexec.bat files. Ask your customer to send you a copy of these files. If the Config.sys has many entries, and your customer is loading a lot of programs to specific locations, this could cause problems. Special memory managers and drivers with a lot of parameters can cause problems. Use only the most basic drivers. Ask the customer to remove as many drivers as possible from their Config.sys. In

general, the more lines that you find in the Config.sys, the greater the chance for conflict. Have them temporarily comment out (rem) all unnecessary statements and reboot the system.

## Display Drivers

A display driver is a software interface that enables Windows to communicate with the hardware video adapter. Windows provides several standard display drivers that work with any video adapter; however, many Windows users install a custom driver provided by the manufacturer of the display adapter in their computer. These custom drivers offer more colors and higher resolutions than are available with the standard Windows display drivers.

When a customer cannot *start* a setup properly, it is often due to a problem with a custom display driver. The following symptoms usually indicate that the problem is the display driver:

- The setup starts but does not appear on the screen.
- The system hangs when the setup attempts to display a bitmap (.bmp) file.
- The screen display malfunctions when the setup attempts to fade in a bitmap.

When a customer reports one of these symptoms, determine which video driver is installed in their system. Suggest that they install a standard Microsoft Windows VGA driver and then run the setup again. If necessary, the customer can always return to the non-standard driver after installing your software.

## Antivirus Programs

Antivirus programs that have been configured to prevent executable files from being copied to the hard disk will interfere with an installation. When an end user reports that an installation appeared to run normally but did not transfer the application to the hard drive, virus protection may be the problem.

In most cases, the default settings of antivirus programs do not block file transfers; most users will not have trouble installing your software. For those users whose antivirus software is set to block certain file transfers, you may want to include a note in your users guide or Readme file advising them of the potential problem.

The solution to the problem is simple enough: Disable the antivirus software and then run the installation again.

## The Target Drive

No matter how well it has been designed or how thoroughly it has been tested, your installation may fail to perform as expected in some cases due to errors on the target drive of an end user’s computer. There are a variety of problems that can occur with magnetic media:

- Flaws and/or damage to the surface of the disk can prevent data from being read from or written to certain disk sectors.
- File allocation table errors, such as lost and cross-linked clusters, can prevent your installation from opening or overwriting files.
- Extreme disk fragmentation, which impairs disk performance, can cause your installation and the installed application to run very slowly.

When you suspect that the target disk may be the cause of an end user’s problems with your installation or the installed application, you can recommend the following tactics:

1. If you suspect that flaws in the disk surface or errors in the file allocation table are the source of the trouble, instruct your end user to run ScanDisk. This utility can lock out bad sectors so that programs no longer attempt to read from or write to them. It will also repair errors in the file allocation table.
2. If you suspect that extreme disk fragmentation is the source of the trouble, recommend that your end user run the Windows Disk Defragmenter.

***Tip*** *▪ You should advise your end users to perform a disk backup before running disk defragmentation software.*

# Index