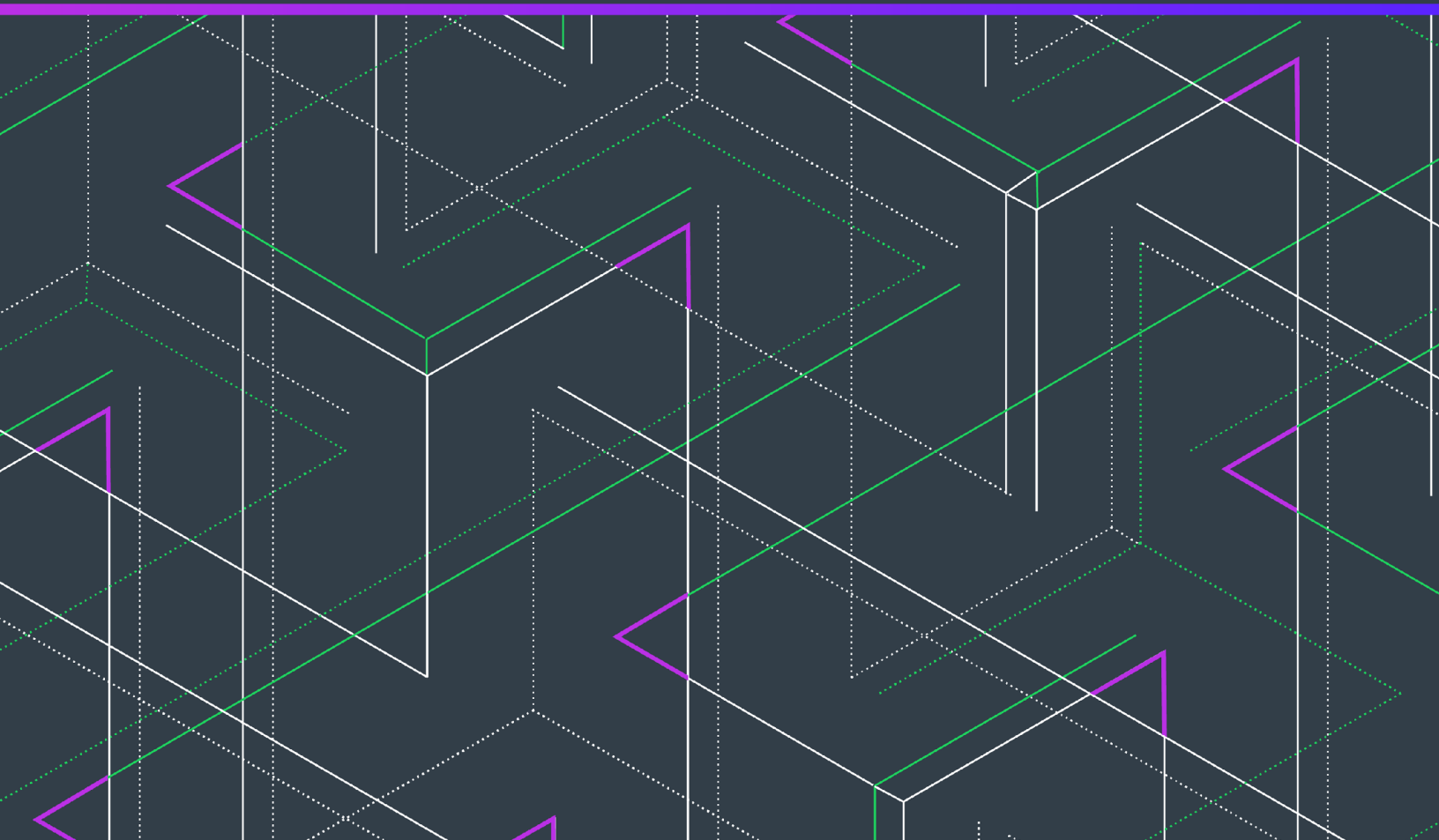
revenera™

Usage Intelligence

Reporting API v2.0.0 Guide



Legal Information

Book Name: Usage Intelligence Reporting API v2.0.0 Guide

Part Number: FUI-200-APIUG02

Product Release Date: October 2020

Copyright Notice

Intellectual Property

Restricted Rights Legend

Contents

1

Usage Intelligence Reporting API v2.0.0 Guide

The Usage Intelligence Reporting API is targeted for advanced users who would like to build their own dashboards or integrate Usage Intelligence reporting (charts or raw data) within third party applications. The API can also be used to export data and statistics out of the Usage Intelligence servers for archiving or custom processing by other solutions.

The Usage Intelligence Reporting API v2.0.0 Guide is organized into the following sections.

Table 1-1 • Usage Intelligence Reporting API v2.0.0 Guide

Section	Description
Quick Start Guide	Helps you learn how to use the API by trying out simple queries, and then start working on more complex queries.
POST vs GET Requests	Explains the two request method options - HTTP POST or HTTP GET requests.
Raw vs. Formatted Responses	Describes the two major methods to retrieve data - either in raw JSON format, or else in readily-formatted charts or tables wrapped in an HTML page.
Authentication	Explains how to authenticate, obtain a session ID, and log out.
Metadata Queries	Lists what filters are available and what filter values are possible.
Event Tracking Management	Explains how to create request to get a list of event names and categories that have been reported by your application to Usage Intelligence, know which ones have been enabled for collection, and also set which ones should be collected.
Generic Reports	Describes the date-range and current date reports, which show new, active, and lost users, and which can then be split by a number of segmentation options such as by country, by product version, etc.

Table 1-1 • Usage Intelligence Reporting API v2.0.0 Guide (cont.)

Section	Description
User Engagement Histograms	Describes the User Engagement Histogram Report, which shows the number of days clients were active within the specified date range, number of times users launched your application, and the total amount of time, in hours, users spent interacting with your application.
Event Tracking Reports	These reports are meant to provide insight of what features in your product are most popular and how they are used.
Churn-Related Reports	This collection of reports is meant to provide insight on the lifetime of lost users. These reports are meant to report what happened during the whole lifetime of these installations rather than what happened during a defined date range.
License Key Registry Management	This section explains how to retrieve and search license keys from the key registry, and how to update and insert new keys in the key registry.
Custom Event Tracking	In Custom Event Tracking reports, data can either be previewed by retrieving the latest data in JSON format or else, zipped CSV files can be downloaded for offline processing.
Exception Tracking	In Exception Tracking reports, data can either be previewed by retrieving the latest data in JSON format or else, zipped CSV files can be downloaded for offline processing.
Client Profile Report	This report retrieves a subset or all of the data about a client or a set of clients.
Raw Data Export	Explains how to download raw data export files. The list of files can be retrieved, and then a temporary URL may be requested for downloading.

Product Support Resources

The following resources are available to assist you with using this product:

- Revenera Product Documentation
- Revenera Community
- Revenera Learning Center
- Revenera Support

Revenera Product Documentation

You can find documentation for all Revenera products on the Revenera Product Documentation site:

https://docs.revenera.com

Revenera Community

On the Revenera Community site, you can quickly find answers to your questions by searching content from other customers, product experts, and thought leaders. You can also post questions on discussion forums for experts to answer. For each of Revenera's product solutions, you can access forums, blog posts, and knowledge base articles.

https://community.revenera.com

Revenera Learning Center

The Revenera Learning Center offers free, self-guided, online videos to help you quickly get the most out of your Revenera products. You can find a complete list of these training videos in the Learning Center.

https://learning.revenera.com

Revenera Support

For customers who have purchased a maintenance contract for their product(s), you can submit a support case or check the status of an existing case by making selections on the **Get Support** menu of the Revenera Community.

https://community.revenera.com

Contact Us

Revenera is headquartered in Itasca, Illinois, and has offices worldwide. To contact us or to learn more about our products, visit our website at:

http://www.revenera.com

You can also follow us on social media:

- Twitter
- Facebook
- LinkedIn
- YouTube
- Instagram

2

Quick Start Guide

The Usage Intelligence API has been built from the ground up to be as flexible as possible. Every report available on the Usage Intelligence Web UI is also available through the API. It is best to start learning how to use the API by trying out simple queries, and then start working on more complex queries at a later stage.

Before building your first queries you should understand the following:

- Authentication Method
- HTTPS Method
- Raw vs Formatted Results
- Example Request

Authentication Method

Before generating a report you must first authenticate with the API server by using one of the 2 methods below:

- Using user and session ID that are sent manually with every request
- Using cookies

If the API is to be accessed via a browser, then the cookie method is normally the most convenient way how to manage the authentication session. If you are writing a script or an application that needs to authenticate automatically and retrieve data without involving web browsers, then it is normally easier to avoid having to use cookies and managing the user name and session ID within your script. For more details about authentication, see Authentication.

HTTPS Method

Report queries may be sent either using POST or GET requests. Whenever possible, POST is the preferred option. When using POST requests, the request data is to be sent as the POST data. GET requests are to be used in cases where using POST is not possible such as when requesting data as an iframe source. In such cases, the request data is to be sent URL encoded as part of the URL inside a parameter named “query”. For more details about these 2 methods, see POST vs GET Requests.

Raw vs Formatted Results

The results that the server returns can either be in a raw JSON format or else can be pre-formatted by the server as an HTML page that includes a chart or table which can be embedded in your page. The raw JSON format is meant to be read by scripts or applications and used for further processing. At the moment, the response can either be sent in raw JSON, or else as a Highcharts chart. Please note that if you choose to use Highcharts in your application, you may require a Highcharts license.

Example Request

In this very basic example, it is assumed that we are accessing the API from a web browser and we would like to get a timeline of the last 60 days showing the number of new, active, and lost users. Normally, the target URL should be used to set the target URL of an iframe, however, for testing, you may simply paste the URL in the browser address bar.

In this example, we are using cookie-based authentication. Since we are getting the data using GET requests, the API server will automatically redirect us to the login page if we are not already logged in. Therefore, we do not need to worry about authentication at this stage.

Building the JSON Request Object

The following is the JSON object which we will be sending as the request. For this example, we will be requesting data from the demo product account.

```
{
    "productId": 2376158762,
    "startDate": "NOW-60",
    "stopDate": "NOW",
    "dateSplit": "day",
    "clientStatus": [
        "new",
        "active",
        "lost"
    ],
    "daysUntilDeclaredLost": 30,
    "dateReportedLost": "dateDeclaredLost"
}
```

Encoding the Request and Getting the Data

The JSON object should be sent as part of the request URL in a parameter named “query”. In this case, the URL should be as follows:

```
https://api.revulytics.com/reporting/generic/dateRange?query={"productId":2376158762,"startDate":"NOW-60","stopDate":"NOW","dateSplit":"day","clientStatus":["new","active","lost"],"daysUntilDeclaredLost":30,"dateReportedLost":"dateDeclaredLost"}&resultFormat=highcharts&hcType=line
```

The “query” value needs to be URL encoded. All modern browsers automatically apply URL encoded to unencoded parameter values. However, it is always recommended to apply encoding before sending the URL. In such case, the encoded URL would be as follows:

```
https://api.revulytics.com/reporting/generic/
dateRange?query=%7B%22productId%22%3A2376158762%2C%22startDate%22%3A%22NOW-
60%22%2C%22stopDate%22%3A%22NOW%22%2C%22dateSplit%22%3A%22day%22%2C%22clientStatus%22%3A%5B%22new%22%2C
%22active%22%2C%22lost%22%5D%2C%22daysUntilDeclaredLost%22%3A30%2C%22dateReportedLost%22%3A%22dateDecla
redLost%22%7D&resultFormat=highcharts&hcType=line
```

3

POST vs GET Requests

The Usage Intelligence API offers 2 request method options - HTTP POST or HTTP GET requests. These options are being offered for flexibility and to allow insertion of reports inside different applications which may range from desktop applications, scripts, browser-side Javascript, and even static HTML pages.

When to Use GET

HTTP GET is the kind of request that is normally used to request a web page. This kind of request is not meant to send large amounts of data. The advantage of GET requests is that all data that is to be sent, is encoded within the URL itself. Therefore, the URL can be copied to other users, used as an iframe src, or saved as a browser bookmark.

When requesting formatted reports which are meant for immediate display rather than for further processing, GET is normally the best option. When using GET requests in order to include a report as part of a web page, one does not need to worry about how to do the request itself. The easiest way is to create an iframe and simply place the URL in the iframe src attribute.

All the examples within this documentation use POST requests. However, all requests other than `POST /auth/login` can be accessed via a GET request. To request a report via GET, the same URL should be called, and the JSON request object should be formatted in exactly the same way. However, instead of sending the JSON object as POST data, it should be sent URL encoded inside a parameter named query.

When to Use POST

HTTP POST requests are designed to send data over the web. This is the type of request that is used after filling an HTML form or when uploading a file. The advantage of this type of request is that it can send large amounts of data while keeping the URL short and simple. However, if the URL is copied, all data is lost.

When requesting raw (JSON) reports, which are meant for further local processing, POST requests are the preferred request method. This is because it is the method that is designed for long data request, and is cleaner and easier to understand. Therefore, if you are building a desktop application or writing a script in which your code (and not a browser) will be handling the request mechanism, it is recommended to use POST whenever possible.

Example Request

The following is an example request of a date range report with no global filters and 1 segmentation level showing only the number of active users using each different product language. In this example, we are assuming that cookie-based authentication is being used, and therefore, the user and sessionId properties are not being used. Also, we are requesting a HighCharts column chart by adding the following parameters to the URL: `&resultFormat=highcharts&hcType=column`

The following is the JSON object which is to be sent, regardless of whether this is a GET or POST request:

```
{
    "productId": 2376158762,
    "startDate": "NOW-60",
    "dateSplit": null,
    "clientStatus": [
        "active"
    ],
    "levels": {
        "level1": {
            "property": "prodLanguage",
            "segments": [
                {
                    "type": "regex",
                    "value": ".*"
                    "split": true
                }
            ]
        }
    }
}
```

The following is how the request URL should look with URL encoding:

```
https://api.revulytics.com/reporting/generic/
current?query=%7B%22productId%22%3A2376158762%2C%22startDate%22%3A%22NOW-
60%22%2C%22dateSplit%22%3Anull%2C%22clientStatus%22%3A%5B%22active%22%5D%2C%22levels%22%3A%7B%22level1%
22%3A%7B%22property%22%3A%22prodLanguage%22%2C%22segments%22%3A%5B%7B%22type%22%3A%22regex%22%2C%22valu
e%22%3A%22.*%22%2C%22split%22%3Atrue%7D%5D%7D%7D%7D&resultFormat=highcharts&hcType=column
```

To better understand the above example, the following is the same URL without using URL encoding:

```
https://api.revulytics.com/reporting/generic/current?query={"productId":2376158762,"startDate":"NOW-
60","dateSplit":null,"clientStatus":["active"],"levels":{"level1":{"property":"prodLanguage","segments"
:[{"type":"regex","value":".*","split":true}]}}}&resultFormat=highcharts&hcType=column
```

4

Raw vs. Formatted Responses

The Usage Intelligence API offers 2 major methods how to retrieve the data - either in raw JSON format which is meant to be processed by your application or scripts, or else in readily-formatted charts or tables wrapped in an HTML page which is meant to be used as an iframe or HTML frame source.

At the moment, formatted charts are generated using Highcharts Javascript components. Tables are generated using a custom HTML format.

Requesting Formatted Reports

By default, all reports are presented in raw JSON. If you would like to get HTML formatted data, you need to add a GET parameter named `resultFormat`, with its value being `highcharts` or `table`. If you are building a chart, then the value should be `highcharts`, while `table` should be used for tabular data.

If you are requesting raw data, then the `resultFormat` property should either be left out, or else, its value should be `raw`. If you are requesting a `highcharts` response, then you also need to add another parameter: `hcType`. This is used to specify the type of chart you would like to receive. The possible values are `pie`, `bar`, `column`, or `line`.

Note • *Line charts should be used when `dateSplit` is not null, while the other chart types are to be used when you are not splitting by date (`dateSplit` is null).*

Example Request

In this example, we are requesting a column chart comparing the number of active users using the different product editions in the last 60 days. For simplicity, we will be sending a GET request, and use cookie authentication. The following is the JSON query object that we should send:

```
{
```

```
    "productId": 2376158762,
    "startDate": "NOW-60",
    "stopDate": "NOW",
    "dateSplit": null,
    "clientStatus": [
        "active"
    ],
    "levels": {
        "level1": {
            "property": "prodEdition",
            "segments": [
                {
                    "type": "regex",
                    "value": ".*",
                    "split": true
                }
            ]
        }
    }
}
```

The following is how the request URL should look with URL encoding:

```
https://api.revulytics.com/reporting/generic/
dateRange?query=%7B%22productId%22%3A2376158762%2C%22startDate%22%3A%22NOW-
60%22%2C%22stopDate%22%3A%22NOW%22%2C%22dateSplit%22%3Anull%2C%22clientStatus%22%3A%5B%22active%22%5D%2
C%22levels%22%3A%7B%22level1%22%3A%7B%22property%22%3A%22prodEdition%22%2C%22segments%22%3A%5B%7B%22typ
e%22%3A%22regex%22%2C%22value%22%3A%22.*%22%2C%22split%22%3Atrue%7D%5D%7D%7D%7D&resultFormat=highcharts
&hcType=column
```

The result should look similar to the following image:

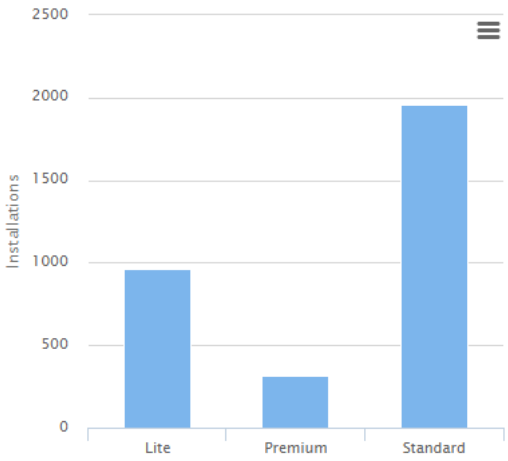


Figure 4-1: Highcharts Example

5

Authentication

Information about authentication is presented in the following sections:

Authenticating and Obtaining a Session ID

Before being able to request any data, one needs to authenticate with the API and get a session ID. The session ID can be retrieved either as part of the JSON response, or else as a cookie.

If authentication is not done, further requests will be rejected. If a POST request for a report is done and no authentication has been made, a 401 error code will be returned. If the same request is done using GET, instead of an error, a 302 redirect is returned to the authentication page.

You use the following POST request to authenticate a user by verifying username and password.

```
POST /auth/login
```

This authentication request has the following properties.

Table 5-1 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account • password (string)—The corresponding password of your Usage Intelligence user account • useCookies (boolean)—Optional parameter to specify whether the response should set a cookie (if true), or include the session ID as part of the JSON response (if false). Default is false.

Table 5-1 • Request Properties

Property	Description
Request Headers	• **Content-Type**—Can be set to `application/json` or `text/javascript` • **Accept**—Should be set to `text/javascript`
Response Headers	• **Content-Type**—Will contain `text/javascript`
Status Codes	• **200 OK**—OK (no error) • **400 Bad Request**—Malformed request • **403 Forbidden**—Wrong credentials supplied
Response JSON Object	• status (string)—Contains OK if successful or SYNTAX ERROR or AUTH ERROR. • reason (string)—Present only if status is not OK. Contains error message (reason). • sessionId (string)—Present only if status is OK and `useCookies` is false. Contains session ID to be used in all further requests.

Example Request

```
POST /auth/login  HTTP/1.1
Host: api.revulytics.com
Content-Type: application/json
Accept: application/json

{
    "user": "testuser@test.com",
    "password": "mypassword1",
    "useCookies": false
}
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "status": "OK",
  "sessionId": "VSB8E2BzSC2eZSJm4QmTpA"
}
```

Logging Out

In order to log out, it is required to invalidate the user session that was created when logging in. If cookies are being used, the cookie will also be invalidated. Two options for logging out are being offered - either invalidate a single session or else, invalidate all sessions that are active on your user ID.

Invalidating a Single Session

You use the following POST request to invalidate a single session.

`POST /auth/logout`

If no JSON data is sent, the system expects to find a valid authentication cookie. If such a cookie is found, the cookie is deleted, and the session is invalidated from the server. If JSON data is sent, then the session and user ID are read from the JSON object as described below.

If this request is done via POST, then a JSON response as described below will be returned. If GET is used, the response is not in JSON but in user-friendly HTML. If you are using GET and still would like to receive a JSON response, add the parameter `resultFormat=raw` so the URL should be `https://api.revulytics.com/auth/logout?resultFormat=raw`.

This request to invalidate a single session the following properties.

Table 5-2 • Invalidating a Single Session Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account • sessionId (string)—The `sessionId` obtained via `POST /auth/login`.
Request Headers	• **Content-Type**—Can be set to `application/json` or `text/javascript` • **Accept**—Should be set to `text/javascript`
Response Headers	• **Content-Type**—Will contain `text/javascript`
Status Codes	• **200 OK**—OK (no error) • **400 Bad Request**—Malformed request • **403 Forbidden**—Wrong credentials supplied
Response JSON Object	• status (string)—Contains OK if successful or `SYNTAX ERROR` or `AUTH ERROR`. • reason (string)—Present only if status is not OK. Contains error message (reason).

6

Metadata Queries

Usage Intelligence reports support filtering and segmentation by a number of different properties. In order to be able to build filters and custom segments, you first need to know what filters are available and what filter values are possible.

Information about metadata queries is presented in the following sections:

- Getting a List of Filter / Segmentation Properties
- Getting a List of Possible Property Values
- Getting Oldest Permitted Date

Getting a List of Filter / Segmentation Properties

This request is to be used to get a list of properties by which you can filter or segment your reports.

Request/Response Parameters Summary

You use the following POST request to get a list of properties by which you can filter or segment your reports.

`POST /meta/productProperties`

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 6-1 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. Required only for non-cookie authentication. • **sessionId (string)**—The sessionId obtained via POST /auth/login. Required only for non-cookie authentication. • **productId (integer)**—The product ID on which this request is being done/

Table 6-1 • Request Properties

Property	Description
Request Headers	• **Content-Type**—Can be set to `application/json` or `text/javascript` • **Accept**—Should be set to `text/javascript`
Response Headers	• **Content-Type**—Will contain `text/javascript`
Status Codes	• **200 OK**—OK (no error) • **400 Bad Request**—Malformed request • **403 Forbidden**—Wrong username, sessionId, session expired, or not authorized
Response JSON Object	• status (string)—Contains OK if successful or SYNTAX ERROR or AUTH ERROR. • reason (string)—Present only if status is not OK. Contains error message (reason). • **results (array)**—Present only if status is OK. Contains the list of available properties as described below.

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "status": "OK",
    "results": [
        {
            "category": "Product Properties",
            "properties": [
                {
                    "name": "prodVersion",
                    "friendlyName": "Product Version",
                    "filterType": "standard",
                    "dataType": "string"
                },
                {
                    "name": "prodEdition",
                    "friendlyName": "Product Edition",
                    "filterType": "standard",
                    "dataType": "string"
                },
                {
                    "name": "prodLanguage",
                    "friendlyName": "Product Language",
                    "filterType": "standard",
                    "dataType": "string"
                },
                {
                    "name": "prodBuild",
                    "friendlyName": "Product Build",
                    "filterType": "standard",
                    "dataType": "string"
```

```
            }
        ]
    },
    {
        "category": "Licensing Properties",
        "properties": [
            {
                "name": "licenseType",
                "friendlyName": "License Type",
                "filterType": "standard",
                "dataType": "string"
            },
            {
                "name": "licenseStatus",
                "friendlyName": "License Status",
                "filterType": "standard",
                "dataType": "special",
                "subProperties": [
                    {
                        "name": "activated",
                        "friendlyName": "Activated",
                        "dataType": "boolean"
                    },
                    {
                        "name": "blacklisted",
                        "friendlyName": "Blacklisted",
                        "dataType": "boolean"
                    },
                    {
                        "name": "expired",
                        "friendlyName": "Expired",
                        "dataType": "boolean"
                    },
                    {
                        "name": "whitelisted",
                        "friendlyName": "Whitelisted",
                        "dataType": "boolean"
                    }
                ]
            }
        ]
    },
    {
        "category": "Platform Properties",
        "properties": [
            {
                "name": "osLanguage",
                "friendlyName": "OS Language",
                "filterType": "standard",
                "dataType": "string"
            },
            {
                "name": "os",
                "friendlyName": "OS",
                "filterType": "standard",
                "dataType": "special",
```

```
        "granularityLevels": [
            {
                "name": "platform",
                "friendlyName": "Platform",
                "dataType": "string"
            },
            {
                "name": "version",
                "friendlyName": "Version",
                "dataType": "string"
            },
            {
                "name": "edition",
                "friendlyName": "Edition",
                "dataType": "string"
            }
        ]
    },
    {
        "name": "geography",
        "friendlyName": "Geography",
        "filterType": "standard",
        "dataType": "special",
        "granularityLevels": [
            {
                "name": "continent",
                "friendlyName": "Continent",
                "dataType": "string"
            },
            {
                "name": "country",
                "friendlyName": "Country",
                "dataType": "string"
            },
            {
                "name": "usState",
                "friendlyName": "US State",
                "dataType": "string"
            }
        ]
    },
    {
        "name": "formFactor",
        "friendlyName": "Form Factor",
        "filterType": "currentData",
        "dataType": "string"
    },
    {
        "name": "cpuType",
        "friendlyName": "CPU Type",
        "filterType": "currentData",
        "dataType": "string"
    },
    {
        "name": "cpuCores",
        "friendlyName": "CPU Cores",
```

```
                "filterType": "currentData",
                "dataType": "numeric",
                "units": ""
            },
            {
                "name": "displayCount",
                "friendlyName": "Monitor Count",
                "filterType": "currentData",
                "dataType": "numeric",
                "units": ""
            },
            {
                "name": "ram",
                "friendlyName": "RAM",
                "filterType": "currentData",
                "dataType": "numeric",
                "units": "MB"
            },
            {
                "name": "resolutionWidth",
                "friendlyName": "Resolution - Horizontal",
                "filterType": "currentData",
                "dataType": "numeric",
                "units": "px"
            },
            {
                "name": "resolutionHeight",
                "friendlyName": "Resolution - Vertical",
                "filterType": "currentData",
                "dataType": "numeric",
                "units": "px"
            },
            {
                "name": "osWordLength",
                "friendlyName": "OS Word Length",
                "filterType": "currentData",
                "dataType": "numeric",
                "units": "bit"
            }
        ]
    }
]
}
```


Getting a List of Possible Property Values

This request is used in order to get a list of possible values for the selected property. This data is then used to build filters or segments as required.

Request/Response Parameters Summary

`POST /meta/propertyValues`

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 6-2 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. Required only for non-cookie authentication. • **sessionId (string)**—The `sessionId` obtained via POST /auth/login. Required only for non-cookie authentication. • **productId (integer)**—The product ID on which this request is being done. • **property (string)**—The name of the property to retrieve listing for • **filter (object)**—An optional value used to filter very long meta lists. This feature is not available if the property is geography. Details about filtering can be found in the Filtering Long Lists section. • **granularity (string)**—Required only when property is os, geography, or gpu. The values can be as follows: **If geography:** ◦ continent ◦ country ◦ usState **If os:** ◦ platform ◦ version ◦ edition **If gpu:** ◦ vendor ◦ model • **startAfter (string)**—Optional value. To be used for paging on long lists. When paging, this should contain the value of `lastValue` from the previous response. • **getFullList (boolean)**—Optional value. Can be used when property is os, `osLanguage`, or geography. Instructs the server to return a list of all known values by the system instead of showing only the ones that were collected in this particular product. Default is false.
Request Headers	• **Content-Type**—Can be set to `application/json` or `text/javascript` • **Accept**—Should be set to `text/javascript`
Response Headers	• **Content-Type**—Will contain `text/javascript`

Table 6-2 • Request Properties

Property	Description
Status Codes	• **200 OK**—OK (no error) • **400 Bad Request**—Malformed request • **403 Forbidden**—Wrong username, sessionId, session expired, or not authorized

Response JSON Object	• status (string)—Contains OK if successful or SYNTAX ERROR or AUTH ERROR.
	• reason (string)—Present only if status is not OK. Contains error message (reason).
	• **results (array)**—Present only if status is OK. Contains the list of property values. Each item is presented in a JSON object as follows:
	  ◦ **value (string)**—Present in all responses. This contains the value that is to be used for filtering and segmentation.
	  ◦ **shortName (string)**—Present if property is os. Contains an abbreviated OS name that can fit better in some user interfaces.
	  ◦ **fullName (string)**—Present if property is geography or osLanguage. Contains the full continent, country, US state, or language name while the value property contains an abbreviated code.
	• **truncated (boolean)**—Present only if the list is too long. Use filtering in order to retrieve shorter lists or use paging by using the `startAfter` parameter. See Filtering Long Lists.
	• **lastValue (boolean)**—Contains the last value of the list. To be used for paging. This value is to be sent as the `startAfter` parameter in the next request.

Example Request

```
POST /meta/propertyValues  HTTP/1.1
Host: api.revulytics.com
Content-Type: application/json
Accept: application/json

{
    "user": "testuser@test.com",
    "sessionId": "VSB8E2BzSC2eZSJm4QmTpA",
    "productId": 12345678901,
    "property": "os",
    "granularity": "version"
}
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
     "status": "OK",
     "results": [
                     {
                         "value": "Microsoft Windows XP",
```

```
                "shortName": "MS Win XP"
            },
            {
                "value": "Microsoft Windows Server 2003",
                "shortName": "MS Win Srv 2003"
            },
            {
                "value": "Microsoft Windows Vista",
                "shortName": "MS Win Vista"
            },
            {
                "value": "Microsoft Windows Server 2008",
                "shortName": "MS Win Srv 2008"
            },
            {
                "value": "Microsoft Windows 7",
                "shortName": "MS Win 7"
            }
        ]
}
```

Filtering Long Lists

When the list of property values is too long, the list will get truncated. This is especially common on requests related with custom properties. The filter property is a JSON object which should contain the following values:

- **type (string)**—Can be either `regex` or `string`. Normally, `regex` should be used because string is to be used only to verify existence of a certain value.
- **value (string)**—A string normally containing a regular expression that defines the filter to be applied.

Getting Oldest Permitted Date

This request returns the oldest allowed start date based on the product plan.

```
POST /meta/oldestReportedDate
```

Example Request

```
POST /meta/oldestReportedDate  HTTP/1.1
Host: api.revulytics.com
Content-Type: application/json
Accept: application/json

{
    "user": "testuser@test.com",
    "sessionId": "VSB8E2BzSC2eZSJm4QmTpA",
    "productId": 12345678901
}
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
    "status": "OK",
    "oldestDate": "2018-03-20"
}
```

7

Event Tracking Management

These requests are used to get a list of event names and categories that have been reported by your application to Usage Intelligence, know which ones have been enabled for collection, and also set which ones should be collected. At this stage, only listing of event names is available. To select which ones should be collected, please visit the old analytics UI.

Listing Event Categories and Names

These requests are used to get a list of event names and categories that have been reported by your application to Usage Intelligence, know which ones have been enabled for collection, and also set which ones should be collected.

`POST /eventTracking/listEventNames`

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 7-1 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. Required only for non-cookie authentication. • **sessionId (string)**—The sessionId obtained via POST /auth/login. Required only for non-cookie authentication. • **productId (integer)**—The product ID on which this request is being done. • **showEvents (array)**—An array of strings that can contain either “all” to list all known events, or else can contain “basic”, “advanced”, or both basic and advanced. This acts as a filter to show only those event names that have been enabled in the basic or advanced event tracking whitelist.

Table 7-1 • Request Properties

Property	Description
Response JSON Object	status (string)—Contains OK if successful or SYNTAX ERROR or AUTH ERROR. reason (string)—Present only if status is not OK. Contains error message (reason). **results (object)**—Contains the results as requested represented as a JSON array. Present only if status is OK. Each array element represents a category. Each of these category elements is formatted as a JSON object and contains the following: • **category (string) or (null)**—Contains the name of the category or null if there are events which have not been assigned a category. • **categoryEventNames (array)**—An array of JSON objects - one element for each event name. Each of these JSON objects contains the following: ◦ **eventName (string)** – The event name as reported by your application ◦ **basic (boolean) (optional)** – Present only if showEvents is set to all or contains more than 1 item. Contains true if this event is enabled in basic event tracking collection. ◦ **advanced (boolean) (optional)** – Present only if showEvents is set to all or contains more than 1 item. Contains true if this event is enabled in advanced event tracking collection.

In the example below, we are requesting for a complete list of all known event names. If we request for only events that have been enabled for basic event tracking, or similarly if we want only those that have been enabled for advanced tracking, the showEvents value should be [“basic”] or [“advanced”] respectively. If you are only requesting a single type, and not all or basic and advanced in a single request, the basic and advanced boolean properties would not be included in the response.

Example Request

```
POST /eventTracking/listEventNames  HTTP/1.1
Host: api.revulytics.com
Content-Type: application/json
Accept: application/json

{
    "user": "testuser@test.com",
    "sessionId": "VSB8E2BzSC2eZSJm4QmTpA",
    "productId": 12345678901,
    "showEvents": ["all"]
}
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json

"status": "OK",
"results": [
  {
      "category": "File Operations",
```

```
        "categoryEventNames": [
            {
                "eventName": "Open",
                "basic": true,
                "advanced": true
            },
            {
                "eventName": "Save",
                "basic": true,
                "advanced": true
            }
        ]
    },
    {
        "category": "Install Wizard",
        "categoryEventNames": [
            {
                "eventName": "Step 1",
                "basic": true,
                "advanced": false
            },
            {
                "eventName": "Step 2",
                "basic": true,
                "advanced": false
            }
        ]
    }
]
```

8

Generic Reports

These are the most versatile reports provided by Usage Intelligence. These consist of reports showing new, active, and lost users, which can then be split by a number of segmentation options such as by country, by product version, etc.

These reports can be based either on a date range - where a start and stop date can be specified by the user, or else on the current data (latest known data about active users).

- **Date-range reports** provide the flexibility of viewing historical data and can also be used to show data on a daily, weekly, or monthly data (line charts). Both date-range and current data reports can be used to show bar charts, pie charts, or tables segmented with up to 3 levels.
- **Current data reports** offer more filtering and segmentation options such as architecture data (cpu type, RAM, number of monitors, etc.).

For more information, see:

- Generic Date-Range Reports
- Generic Current Reports

Generic Date-Range Reports

This reporting mechanism is to be used for generating reports regarding user activity within a particular specified date range. Depending on the request, this can create timeline charts, pie/bar charts, geographical maps, or hierarchical tables.

- Request/Response Parameters Summary
- Global Filters
- Segmentation and Levels (Date-Range Reports)
- Results Format for Reports Using Date Splitting
- Results Format for Reports Without Date Splitting

Request/Response Parameters Summary

This is used to generate reports within a particular specified date range.

`POST /reporting/generic/dateRange`

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 8-1 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. Required only for non-cookie authentication. • **sessionId (string)**—The `sessionId` obtained via POST /auth/login. Required only for non-cookie authentication. • **productId (integer)**—The product ID on which this request is being done • **groupBy (string)**—Optional parameter to specify the property with which to group installations. By default, this value is considered to be `clientId`. Other possible options are `machineId`, `licenseKey` or any custom property of type 3. • **dateSplit (string/null)**—Whether to present results by day, week, or month or to combine all results into a pie/bar chart or table • **startDate (string)**—The first date of the date range on which to base the report. This is to be formatted as YYYY-MM-DD. • **stopDate (string)**—The last date of the date range on which to base the report. This is to be formatted as YYYY-MM-DD. • **clientStatus (array)**—The client statuses to show in the report. Each report can show any combination of new, active, and lost users. This is to be presented as an array of up to 3 strings which can be new, active, and lost. • **daysUntilDeclaredLost (integer)**—Required only if the `clientStatus` array contains lost. This specifies the number of consecutive days of inactivity that have to pass until a client installation is declared lost. • **dateReportedLost (string)**—Required only if the `clientStatus` array contains lost. When an installation is lost, it can either be shown as lost on the date it last contacted the Usage Intelligence servers (`dateLastSeen`) or else it can be shown as lost when it was declared lost (last date when it contacted the Usage Intelligence servers + the number of days specified in `daysUntilDeclaredLost`) (`dateDeclaredLost`). Therefore, the permitted values are `dateLastSeen` and `dateDeclaredLost`. • **globalFilters (object)**—JSON object containing the filters to be applied to the available data. Details about these filters can be found in the Global Filters section. • **levels (object)**—Optional JSON object that describes how data is to be split into segments in each level. Requests where `dateSplit` is null can have up to 3 levels while when date splitting is being done, 0 or 1 levels can be specified. Details about levels and segmentation can be found in the Segmentation and Levels (Date-Range Reports) section.

Table 8-1 • Request Properties

Property	Description
Response JSON Object	• status (string)—Contains OK if successful or SYNTAX ERROR or AUTH ERROR. • reason (string)—Present only if status is not OK. Contains error message (reason). • **levelLabels (object)**—Present only on responses where `dateSplit` is null. When present, this contains the property names of each level as requested. • **results (object)**—Contains the results as requested represented as a JSON object. The structure differs between reports using date splitting (Results Format for Reports Using Date Splitting) and those that have date splitting set to null (Results Format for Reports Without Date Splitting). The result format is described below.

Global Filters

Most of the available filter properties are string-based. This means that when applying a filter, the requested field can be represented as a `string`, `stringArray` or regex. There are also some filters which are numeric. These filters should be represented as `number` or `numberRange`.

- String-Based Filters
- Numeric Filters
- Date Range Filters
- Boolean Filters
- Special Filters
- <NULL> Values in Global Filters (Date-Range Reports)

String-Based Filters

The following properties are stored as strings:

```
machineId *
clientId *
prodVersion
prodEdition
prodBuild
prodLanguage
licenseType
formFactor *
osLanguage
osWordLength *
cpuType *
dotNetVersion *
javaVersion *
javaVendor *
javaRuntime *
javaGraphics *
javaVmVersion *
```

```
javaVmName *
vm *
C01 .. C20 (Custom properties)
licenseKey *
```

Note • *licenseKey requires a special user permission to be used as a filter.*

Note • *Properties marked with an asterisk (*) are based on the current (latest known) values.*

The **type** field in the above filters needs to be `string`, `stringArray` or `regex`. A **value** field is always required. The contents of this field should be according to the specified type.

- If `string` is specified, then the value field must contain a single string that needs to be matched precisely with the stored data.
- If `stringArray` is specified, then the value field must contain an array of strings where one of which needs to match precisely with the stored data.
- If specifying a `regex`, the value field should contain a string which is treated as a regular expression and the stored data will be matched against it using regular expression rules.

Example Filter Using a String Value

In this example, the product build value needs to be exactly "`3014.int-12214`":

```
{
      "prodBuild":
          {
              "type": "string",
              "value": "3014.int-12214"
          }
}
```

Example Filter Using a String Array

In this example, the product build value needs to be either "`3014.int-12214`", "`3017.enx-57718`", or "`4180.vrx-81059`". Note that since the type is declared as `stringArray`, the value field needs to contain an array. Consider all elements in the array to have an OR logical expression between them.:

```
{
    "prodBuild":
        {
            "type": "stringArray",
            "value": ["3014.int-12214", "3017.enx-57718", "4180.vrx-81059"]
        }
}
```

Example Filter Using a Regular Expression

In this example, the product build value needs to start with "`30`" and end with "`18`" whilst having 10 characters in between:

```
{
     "prodBuild":
         {
```

```
            "type": "regex",
            "value": "^30.{10}18$"
        }
}
```

Numeric Filters

The following properties are stored as numeric values:

```
cpuCores *
displayCount *
ram *
resolutionWidth *
resolutionHeight *
lifetimeRuntimeMinutes *
lifetimeSessionCount *
screenPpi *
javaVmRam *
```

Note • *Properties marked with an asterisk (*) are based on the current (latest known) values.*

The type field in the above filters needs to be `number` or `numberRange`.

- If number is specified, then a value field must also be present. The value field should contain a number, which may contain a decimal point if required.
- If `numberRange` is specified, then the value field should NOT be used. Instead, the properties min and max are to be used. These refer to the minimum and maximum number to be included in the report. If only one limit needs to be set, the other property is to be left out. Therefore, if you want to include installations with up to 2 display devices, you would not specify a min value, but instead specify only a max and set it as 2.

Example Filter Using a Number Value

In this example, the number of display devices needs to be exactly 3:

```
{
      "displayCount":
          {
              "type": "number",
              "value": 3
          }
}
```

Example Filter Using a Number Range Value

In this example, the RAM needs to be between 1025MB and 4096MB (both included):

```
{
      "ram":
          {
              "type": "numberRange",
              "min": 1025,
              "max": 4096
          }
```

```
}
```

Date Range Filters

The following properties are stored as dates:

`dateInstalled`
`dateLastSeen`

The type field in the above filters needs to be `date` or `dateRange`.

- If date is specified, then a value field must also be present. The value field should contain a date.
- If `dateRange` is specified, then the value field should NOT be used. Instead, the properties min and max are to be used. These refer to the minimum and maximum dates to be included in the report. If only one limit needs to be set, the other property is to be left out.

In the following example, users installed after January 1st 2018 are to be shown:

```
{
    "dateInstalled":
        {
            "type": "dateRange",
            "min": "2018-01-01"
        }
}
```

Note that all dates must be in ISO 8601 format.

Boolean Filters

The following property is stored as boolean:

`touchScreen`

The type field in the above filters needs to be boolean. The value must be true or false.

In the following filter, clients with a touch screen are being requested.

```
{
    "touchScreen":
        {
            "value": true
        }
}
```

Special Filters

Some filters need to be represented in a special format due to their unique requirements. These special filters are:

- Special Filter: licenseStatus
- Special Filter: os
- Special Filter: geography
- Special Filter: gpu

- Special Filters: optOut and backOff
- Special Filter: lifetimeEventUsage
- Special Filter: reachOutDeliveries

Special Filter: licenseStatus

The `licenseStatus` filter is made up of 4 sub-values: `activated`, `blacklisted`, `expired` and `whitelisted`. These are presented as boolean values.

Unlike other filters, this filter is presented as an array of JSON objects. Each object can contain a subset (or all) of these 4 boolean values.

Consider the following example. In this example, for a client to be included, the license has to either be activated AND whitelisted, or else it can be not whitelisted AND expired. In other words, **(** (activated AND whitelisted) OR ((NOT)whitelisted AND expired) **)**.

```
{
    "licenseStatus":
        [
            {
                "activated": true,
                "whitelisted": true
            },
            {
                "whitelisted": false,
                "expired": true
            }
        ]
}
```

Special Filter: os

The os filter is made up of 3 granularity levels. These are `platform`, `version`, and `edition`. These are meant to split the OS name into levels of detail as required by the user. Consider the following:

- **platform**: Microsoft Windows
- **version**: Microsoft Windows 7
- **edition**: Microsoft Windows 7 Professional

If a filter is set on the version “Microsoft Windows 7”, the result would include all editions of Windows 7. One or more of these granularity levels may be specified. If more than 1 granularity level is specified, the values are ORed together.

In the following example, all editions of “Microsoft Windows 7” are included, and also “Microsoft Windows Vista Home Premium”:

```
{
  "type": "string",
  "version": "Microsoft Windows 7",
  "edition": "Microsoft Windows Vista Home Premium"
}
```

In the following example, the type is `stringArray`. Note that an array needs to be passed if the type is set as such, even if it is to contain only 1 element. In this case, the version can be either “Microsoft Windows 7” or “Microsoft Windows 8” (which are ORed together). Also, clients running on “Microsoft Windows XP Professional” are to be included.

```
{
    "type": "stringArray",
    "version": ["Microsoft Windows 7", "Microsoft Windows 8"],
    "edition": ["Microsoft Windows XP Professional"]
}
```

Special Filter: geography

The geography filter is made up of 3 granularity levels. These are `continent`, `country`, and `usState`.

The `usState` value applies only to United States. Continents and countries are presented in 2-letter codes. Countries follow ISO standard 3166-1 alpha-2. US states are presented in ISO 3166-2:US format.

In the following example, the clients have to be either:

- In the continents *`Asia`* or *`Oceania`*
- In the country *`Germany`*
- In the US states *`New York`*, *`New Jersey`*, or *`Kansas`*

```
{
    "type": "stringArray",
    "continent": ["AS", "OC"],
    "country": ["DE"],
    "usState": ["US-NY", "US-NJ", "US-KS"]
}
```

Important • *In this filter, the type can be string or `stringArray`. Regular expressions are not supported in geography filters.*

Special Filter: gpu

The gpu filter is made up of 2 granularity levels. These are `vendor` and `model`. Both are represented as string values.

In the following example, the clients have to have a GPU:

- From the vendors NVIDIA or Intel
- With the model AMD Radeon HD 4600

```
{
    "type": "stringArray",
    "vendor": ["NVIDIA", "Intel"],
    "model": ["AMD Radeon HD 4600"]
}
```

Special Filters: optOut and backOff

The opt-out mechanism was introduced in SDK version 5.1.0. With this feature, vendors can have their application report to the Usage Intelligence servers that a user does not want to be tracked. Using this property, vendors can filter out installations that were opted-out.

Similarly, backoff filtering was introduced with version 5.0.0. Backoff is when a product account runs over-quota and the server starts rejecting data. Although filtering for backed-off installations was introduced with version 5, it was also backported to previous versions. However, when a new installation with an SDK prior to version 5 tries to register with the server and is rejected, it is not marked as being once backed-off when it is eventually accepted by the server. With version 5 onwards, the server flags an installation as being historically backed-off in such cases.

Both `backOff` and `optOut` filters are made up of 2 boolean sub-values: `historical` and `current`.

- The `historical` value refers to installations that were once backed-off or opted-out. These may include installations that are still currently backed-off or opted-out.
- The `current` value refers to the status during the last time that the client called the server.

Therefore, if an installation was opted-out yesterday but got opted-in today, it will be marked as historically opted-out but not currently opted-out.

In the following example, for a client to be included, the `optOut` status has to either be `historical` AND not `current`, or else it can be not `historical` (i.e. users have to be currently opted-in but used to be opted-out at some point or never opted out).

```
{
    "optOut":
        [
            {
                "historical": true,
                "current": false
            },
            {
                "historical": false
            }
        ]
}
```

Special Filter: lifetimeEventUsage

Using lifetime event usage filters, clients can be filtered based on whether a particular event or set of events occurred or not within the client's lifetime. Alternatively, one can set a filter based on the number of times an event has occurred.

In the following example, clients that are included must have done the "File Operations - Open" event at least 5 times to be counted.

```
{
     "category": "File Operations",
     "name": "Open",
     "min": 5
}
```

In the following example, clients must have done either "File Operations - Open" or "File Operations - Save" for a combined total of between 10 to 50 times.

```
{
    "combiArray": [
                    {
                      "categoryType": "string",
                      "nameType": "string",
                      "category": "File Operations",
                      "name": "Open"
                    },
                    {
                      "categoryType": "string",
                      "nameType": "string",
                      "category": "File Operations",
                      "name": "Save",
                    }
                ],
    "min": 10,
    "max": 50
}
```

In the following example, clients must have done any event within the “File Operations” category for a combined total of not more than 100 times. This is done using a regular expression in the name.

```
{
    "combiArray":  [
                     {
                       "categoryType": "string",
                       "nameType": "regex",
                       "category": "File Operations",
                       "name": ".*"
                     }
                   ],
    "max": 100
}
```

Special Filter: reachOutDeliveries

Using ReachOut delivery filters, clients can be filtered based on whether a particular ReachOut message or a combination of ReachOut messages were delivered or not within the client’s lifetime.

The filter consists of a JSON array that includes one or more objects. Each object is a combination of delivered and undelivered campaigns, and the different combinations are ORed together. Therefore, it is possible to show users that either received ReachOut message 1 but not 2, or else received 3 but not 4 as in the following example.

In the following example, we are looking for clients who either received campaign 1 but not 2, OR received campaign 2 but not 3.

```
[
     {"auto": {"delivered": ["1"], "undelivered":["2"]}},
     {"auto": {"delivered": ["2"], "undelivered":["3"]}}
]
```

The above example contains only “auto” ReachOut campaigns. Manual campaigns can be specified using “manual” instead of “auto” as in the above example. Each object can contain a mix of “auto” and “manual” campaigns.

<NULL> Values in Global Filters (Date-Range Reports)

Most of the available properties can include null values. There are different reasons why a value would be null. When these are properties that are set by the application, the possible reasons why a value would be null are: cases where the value has not been set by the application (such as `prodBuild` never being set), and cases where values are set to an empty string ("") or to a string containing "<NULL>".

One other reason is that although these values have been set, the SDK has not yet had time to sync with the servers to provide this new information.

In cases where the properties are set automatically such as hardware or OS related information, the values would be null if the SDK failed to retrieve that value from the OS or if the server failed to identify the value retrieved by the SDK.

Other reasons include cases where Java version is requested from an application that does not use the Java SDK, US state is requested for users who are not running within the US, etc.

The following are the properties that support null values:

```
prodVersion
prodEdition
prodBuild
prodLanguage
machineId
formFactor
vm
cpuType
cpuCores
ram
resolutionWidth
resolutionHeight
javaVersion
javaVmVersion
javaVmName
javaVendor
javaRuntime
javaGraphics
osLanguage
licenseKey
C01 .. C20 (Custom properties)
os
geography
gpu
```

Null values can be requested either on their own or as part of a filter containing other values.

The following example would return only cases where the `prodVersion` is null:

```
{
      "prodVersion":
          {
              "includeNull": true
          }
}
```

The following example would return cases where the `prodVersion` is either 1.1, 1.2 or null:

```
{
      "prodVersion":
```

```
        {
            "type": "stringArray",
            "value": ["1.1", "1.2"],
            "includeNull": true
        }
}
```

By default, when specifying a filter, null values would not be included. Therefore, in the following example, only clients with `prodVersion` set to 1.1 or 1.2 will be included, while null values are excluded:

```
{
    "prodVersion":
        {
            "type": "stringArray",
            "value": ["1.1", "1.2"]
        }
}
```

However, if no filter is specified, then nulls are included by default. Therefore, if you want to include any value of `prodVersion` as long as it is not null, a `prodVersion` filter needs to be included as follows:

```
{
    "prodVersion":
        {
            "type": "regex",
            "value": ".*",
            "includeNull": false
        }
}
```

In the case of filters that use sub-properties (`os`, `geography`, and `gpu`), the `includeNull` filter is to be included in the sub-property and applies to that specific sub-property only. In order to be able to include the `includeNull` property, instead of providing the value as a string or an array of strings, the value of the sub-property must be a JSON object that contains a property named "`value`", and another named "`includeNull`". Each of these properties is optional, but at least one of them must be present.

In the case of geography, this has a very particular meaning. Requesting for null "country" value does not return all cases where the country could not be retrieved, but only cases where the continent could be retrieved but the country could not. Similarly, requesting null "`usState`" returns cases where the `continent` and `country` could be retrieved but the US state could not. This does not include clients that are not situated in the US. If you are interested in finding clients where we could not detect any geographical data, the `includeNull` filter needs to be applied in the continent sub-property.

In the following example, we are requesting cases where we know that the client is within the US but the state could not be identified:

```
{
    "geography":
        {
            "type": "string",
            "country": "US",
            "usState":
                {
                    "includeNull": true
                }
        }
}
```

In the following example, we are requesting cases where the GPU is either "NVIDIA", "AMD" or null (unidentified):

```
{
      "gpu":
          {
              "type": "stringArray",
              "vendor":
                  {
                      "value": ["NVIDIA", "AMD"],
                      "includeNull": true
                  }
          }
}
```

Segmentation and Levels (Date-Range Reports)

Report data may be segmented in up to 3 levels. Each level represents a property. Using multi-level segmentation, you can create hierarchical reports that are used to drill down based on the property values. For example, you can create a report that splits all the users based on which edition they are using. Then, you can split the user counts for each edition based on product version on level 2. Finally, the user counts for each product version can be split up by product build on level 3.

Such multi-level reporting is available only when not splitting by date. Splitting by date is meant to be used in timeline reports, and creating such a hierarchy for each date would result in a huge data set which would only have limited use. When date splitting is used, segmentation with 1 level is still allowed, however. This allows creating timeline reports where each line represents a property value, such as comparing usage trend for version 1 vs version 2.

Segmentation is optional. In order not to split data by any property, you may either not include a levels property in the request object, or else leave the levels object empty. The following examples show the difference between not requesting any segmentation, requesting a single level, and requesting 2 levels:

Table 8-2 • Segmentation and Levels

Number of Levels	Description
0 Levels	• New Users: 30 • Active Users: 100 • Lost Users: 20

1 Level	**Product Versions:**
	• **Version 1**:
	  ◦ New Users: 10
	  ◦ Active Users: 30
	  ◦ Lost Users: 8
	• **Version 2**:
	  ◦ New Users: 20
	  ◦ Active Users: 70
	  ◦ Lost Users: 12

Table 8-2 • Segmentation and Levels

Number of Levels	Description	

2 Levels	**Product Versions:**
	• **Version 1**:
	  ◦ New Users: 10
	  ◦ Active Users: 30
	  ◦ Lost Users: 8
		Product Editions:
		• **Premium:**
		  ◦ New Users: 4
		  ◦ Active Users: 13
		  ◦ Lost Users: 1
		• **Standard:**
		  ◦ New Users: 6
		  ◦ Active Users: 17
		  ◦ Lost Users: 7

	Product Versions:
	• **Version 2**:
	  ◦ New Users: 20
	  ◦ Active Users: 70
	  ◦ Lost Users: 12
		Product Editions:
		• **Premium:**
		  ◦ New Users: 5
		  ◦ Active Users: 40
		  ◦ Lost Users: 4
		• **Standard:**
		  ◦ New Users: 15
		  ◦ Active Users: 30
		  ◦ Lost Users: 8

In the above examples, the first example, 0 Levels, is showing a case where no segmentation is being applied.

The second example, 1 Level, is showing a response where a single level of segmentation has been requested. In this case, segmentation is being done based on product versions. For each product version, one can see the number of new, active, and lost users within the specified date range. Segmenting by 1 level is also possible in reports that use date splitting, so, in a similar example, one would be able to see how many users were using a specific version on each day within the date range.

The third example, 2 Levels, shows 2 levels of segmentation. In this example, one can see how many new, active, and lost users were using each version, and then, that data is further split by product edition. A further level is also allowed, so, for example one may choose to segment each product edition by product language.

The properties that are available for segmentation are the same ones that are used for Global Filters. There are 4 properties that require special formatting. These are described below.

Segment levels are to be defined in a property inside the main JSON object named “levels”. This property should contain a JSON object which contains the following members:

- **property (string)**—The name of the property by which to segment. Note that in case of `os`, `geography`, `licenseStatus` and `gpu`, a special format is used.
- **segments (array)**—An array containing a number of JSON objects. The format of these object is described in the Level Segments Format section below.

- **sort (string)**—Optional property to specify how the segments in this level are to be sorted. Possible values are alpha, new, active, and lost. alpha refers to alphabetical sorting which is based on the segment label. The other 3 are based on the client statuses. Note that if sorting by new, active, or lost users, that particular client status must be included in the clientStatus array. If this property is not included, the data is sorted alphabetically by default.
- **sortDirection (string)**—Optional property to specify whether to sort in ascending or descending order. Possible values are `asc` and `desc`. If not specified, data is sorted in ascending order by default.

For more information, see:

- Level Segments Format
- String-Based Segmentation Properties
- Numeric Segmentation Properties
- Boolean Segmentation Properties
- Special Segmentation Properties
- <NULL> Values in Segmentation and Levels (Date-Range Reports)

Level Segments Format

Segments are defined as JSON objects. A single JSON object may create a single item on a table (or a single series on a timeline chart), or it may create a number of items/series if splitting is enabled. Each object should contain the following:

- **type (string)**—The data type of the value. Can be `string`, `stringArray`, `regex`, `number` or `numberRange` based on whether the property is string-based (String-Based Filters) or numeric (Numeric Filters).
- value (string/array/number)—An exact string, an array of strings, a regular expression or a numeric value. This property should not be used if type is `numberRange`. Format is based on whether the property is string-based (String-Based Filters) or numeric (Numeric Filters).
- min (number)—Used only if the type is `numberRange`. Contains the minimum numeric value to include in this segment. May be combined with max.
- max (number)—Used only if the type is `numberRange`. Contains the maximum numeric value to include in this segment. May be combined with min.
- split (boolean)—Used only if the type is `stringArray` or `regex`. This specifies whether to split the returned data based on each different value matched by the regular expression or array (true), or to join all the clients that match the value as 1 table value or series (false).
- segmentLabel (string)—Used only if split is false or if type is `numberRange`. This is required to give a name to a series when not splitting by value. It is important that the name given is unique.
- limit (integer)—Optional property to set the limit on the maximum number of table values or series that should be produced by this set of values. To be used only if split is true.

String-Based Segmentation Properties

The following properties are stored as strings:

```
machineId
clientId
```

```
prodVersion
prodEdition
prodBuild
prodLanguage
licenseType
formFactor *
osLanguage
osWordLength *
cpuType *
javaVersion *
javaVendor *
javaRuntime *
javaGraphics *
javaVmVersion *
javaVmName *
vm *
C01 .. C20 (Custom properties)
licenseKey *
```

Note • *`licenseKey` requires a special user permission to be used for segmentation.*

Note • *Properties marked with an asterisk (*) are based on the current (latest known) values.*

The **type** field when using one of the above properties needs to be `string`, `stringArray` or `regex`. A **value** field is always required. The contents of this field should be according to the specified type.

- If `string` is specified, then the value field must contain a single string that needs to be matched precisely with the stored data.
- If `stringArray` is specified, then the value field must contain an array of strings where one of which needs to match precisely with the stored data.
- If specifying a `regex`, the value field should contain a string which is treated as a regular expression and the stored data will be matched against it using regular expression rules.

Example Using 1-Level Segmentation by string, stringArray, and regex Values

```
{
    "level1": {
        "property": "prodVersion",
        "segments": [
            {
                "type": "string",
                "value": "1.0"
            },
            {
                "type": "stringArray",
                "value": ["2.0", "2.1", "3.1"],
                "split": false,
                "segmentLabel": "Versions 2 and 3"
            },
            {
                "type": "regex",
```

```
                "value": "^4\..*",
                "split": false,
                "segmentLabel": "All version 4"
            },
            {
                "type": "regex",
                "value": "^5\..*",
                "split": true
            }
        ]
    }
}
```

In the above example, we are requesting a report with multiple segments. The first segment contains installations running version 1.0. Notice how this does not require a "`split`" property since there is only 1 value and therefore no further splitting is possible. The second segment contains versions 2.0, 2.1 and 3.1. In this case, the "`split`" property is required, and since we are requesting the API to combine these 3 versions, we must provide a "`segmentLabel`" value so that the returned data can be identified. The third segment is similar, although in this case the request is built using a regular expression. In this case, all versions starting with "4." are to be included into one combined segment.

The last segment is different from the rest because we are requesting the API to split the data (split is set to true). Therefore, this can produce much more than 1 segment. In this case, we could see segments such as "5.1", "5.2", etc. Notice how since we are splitting, we should not provide a `segmentLabel` value since the labels are built using the different values that are found in the data.

Numeric Segmentation Properties

The following properties are stored as numeric values:

```
cpuCores *
displayCount *
ram *
resolutionWidth *
resolutionHeight *
lifetimeRuntimeMinutes *
lifetimeSessionCount *
screenPpi *
javaVmRam *
```

Note • *Properties marked with an asterisk (*) are based on the current (latest known) values.*

The **type** field in the above properties needs to be `number` or `numberRange`. If number is specified, then a value field must also be present. The value field should contain a number, which may contain a decimal point if required. If `numberRange` is specified, then the value field should NOT be used. Instead, the properties min and max are to be used. These refer to the minimum and maximum number to be included in the report. If only one limit needs to be set, the other property is to be left out. Therefore, if you want to include installations with up to 2 display devices, you would not specify a min value, but instead specify only a max and set it as 2.

Example Using 1-Level Segmentation by number, and numberRange Values

```
{
    "level1": {
        "property": "cpuCores",
```

```
        "segments": [
            {
                "type": "number",
                "value": 1,
            },
            {
                "type": "numberRange",
                "min": 2,
                "max": 4,
                "segmentLabel": "2 - 4"
            },
            {
                "type": "numberRange",
                "min": 5,
                "segmentLabel": "5 +"
            }
        ]
    }
}
```

In the above example, we are requesting a report with 3 segments. The first segment contains only installations running on 1 CPU core, the second segments contains installations running on 2, 3, or 4 cores (range 2 - 4), while the last segment contains all installations which are running on a machine with 5 or more CPU cores. Note how when the type was `numberRange`, we had do specify a `segmentLabel` which is a free string that will be used by the user to identify what is being included in that specific segment.

Boolean Segmentation Properties

The following properties are stored ad boolean values:

`touchScreen`

The type field needs to be boolean, and the value must be true or false. A `segmentLabel` field is also required

Example Using 1-Level Segmentation by Boolean Value

```
{
    "level1": {
        "property": "touchScreen",
        "segments": [
            {
                "type": "boolean",
                "value": true,
                "segmentLabel": "Yes"
            },
            {
                "type": "boolean",
                "value": false,
                "segmentLabel": "No"
            },
            {
                "includeNull": true,
                "segmentLabel": "Unknown"
            }
        ]
```

```
    }
}
```

In the above example, we are requesting a report with 3 segments. The first segment contains installations on which a touch screen was detected, the second one where no touch screen has been detected, while the last one is where we could not detect whether a touch screen is present due to the client using an old SDK which did not have touch screen detection support.

Special Segmentation Properties

Some properties need to be represented in a special format due to their unique requirements. These special properties are:

- Special Segmentation Format: licenseStatus
- Special Segmentation Format: os
- Special Segmentation Format: geography
- Special Segmentation Format: gpu
- Special Segmentation Format: optOut and backOff

Special Segmentation Format: licenseStatus

The `licenseStatus` value is made up of 4 sub-values: `activated`, `blacklisted`, `expired` and `whitelisted`. These are presented as boolean values. Any number of segments can be defined, and each segment can contain any subset of the 4 sub-values. These values are ANDed together. A `segmentLabel` value is required.

In the following example, 2 segments are specified - the first one showing blacklisted AND not expired and the second one showing whitelisted AND activated:

```
[
    {
        "segmentLabel": "BL and not EXP",
        "blacklisted": true,
        "expired": false
    },
    {
        "segmentLabel": "WL and ACT",
        "whitelisted": true,
        "expired": true
    }
]
```

Special Segmentation Format: os

The os value is made up of 3 granularity levels - `platform`, `version`, and `edition`. A particular level needs to be selected, and this is to be included in the property name such as os.version or os.edition. Different granularity levels can be requested for different segmentation levels. Therefore, it is possible to generate a 3-level hierarchical tree in which level 1 would show the OS platform, level 2 would show the version, and level 3 would show the full name including the OS edition or sub-version. For a description of the differences between the 3 granularity levels, refer to Special Filter: os.

The following example shows the levels object requesting 3 granularity levels as described above:

```
{
    "level1": {
        "property": "os.platform",
        "segments": [
            {
                "type": "regex",
                "value": ".*",
                "split": true
            }
        ]
    },
    "level2": {
        "property": "os.version",
        "segments": [
            {
                "type": "regex",
                "value": ".*",
                "split": true
            }
        ]
    },
    "level3": {
        "property": "os.edition",
        "segments": [
            {
                "type": "regex",
                "value": ".*",
                "split": true
            }
        ]
    }
}
```

In the above example, no filtering is being done, and instead, a regular expression to include everything is set as the value. This will result in all OS platforms, versions, and editions to be included in the hierarchy.

Special Segmentation Format: geography

The geography value is made up of 3 granularity levels - `continent`, `country`, and `usState`. These granularity levels are explained in Special Filter: geography. A particular level needs to be selected, and this is to be included in the property name such as geography.continent or geography.country. Different granularity levels can be requested for different segmentation levels. Therefore, it is possible to generate a 3-level hierarchical tree in which level 1 would show the continent, level 2 would show the country, and level 3 would show the US state (for United States only).

The following example shows the levels object requesting 3 granularity levels as described above:

```
{
    "level1": {
        "property": "geography.continent",
        "segments": [
            {
                "type": "regex",
                "value": ".*",
                "split": true
            }
```

```
            ]
        },
        "level2": {
            "property": "geography.country",
            "segments": [
                {
                    "type": "regex",
                    "value": ".*",
                    "split": true
                }
            ]
        },
        "level3": {
            "property": "geography.usState",
            "segments": [
                {
                    "type": "regex",
                    "value": ".*",
                    "split": true
                }
            ]
        }
}
```

In the above example, no filtering is being done, and instead, a regular expression to include everything is set as the value. This will result in all the continents, countries and US states (where applicable) to be included in the hierarchy.

Special Segmentation Format: gpu

The gpu value is made up of 2 granularity levels - vendor and model. These granularity levels are explained in Special Filter: gpu. A particular level needs to be selected, and this is to be included in the property name, namely gpu.vendor or gpu.model. Different granularity levels can be requested for different segmentation levels. Therefore, it is possible to generate a 2-level hierarchical tree in which level 1 would show the vendor, and level 2 would show the model number.

The following example shows the levels object requesting 2 granularity levels as described above:

```
{
    "level1": {
        "property": "gpu.vendor",
        "segments": [
            {
                "type": "regex",
                "value": ".*",
                "split": true
            }
        ]
    },
    "level2": {
        "property": "gpu.model",
        "segments": [
            {
                "type": "regex",
                "value": ".*",
                "split": true
            }
```

```
        ]
    }
}
```

In the above example, no filtering is being done, and instead, a regular expression to include everything is set as the value. This will result in all the GPU vendors and models to be included in the hierarchy.

Special Segmentation Format: optOut and backOff

Both `backOff` and `optOut` values are made up of 2 boolean sub-values: historical and current. Any number of segments can be defined, and each segment can contain any subset of the 2 sub-values. These values are ANDed together. A `segmentLabel` value is required.

In the following example, 2 segments are specified - the first one showing historical AND not current and the second one showing not historical (i.e. never opted-out):

```
[
    {
        "segmentLabel": "HISTORICAL and not CURRENT",
        "historical": true,
        "current": false
    },
    {
        "segmentLabel": "Never opted-out",
        "historical": false
    }
]
```

<NULL> Values in Segmentation and Levels (Date-Range Reports)

Null values in segmentation are to be requested in a similar way to <NULL> Values in Global Filters (Date-Range Reports). The same properties that support null in filtering also support null in segmentation.

By default, when segmenting, null values are not included within the segments, since only the values that have been specified in each segment are included. Null values don't match any regular expression, so the only way to request null values to be included is to specify "includeNull" as true in a similar way to filtering. In segmentation, null values are returned as "<NULL>". The API considers all cases where the data has never been set from the SDK, set as an empty string, or set as a string containing "<NULL>" to be the same.

The following example requests all values of `prodBuild` including null:

```
{
    "level1": {
        "property": "prodBuild",
        "segments": [
            {
                "type": "regex",
                "value": ".*",
                "includeNull": true
            }
        ]
    }
}
```

In the case of segmentation properties that use sub-properties (os, geography, and gpu), the includeNull value is to be included in the sub-property and applies to that specific sub-property only. In order to be able to include the includeNull property, instead of providing the value as a string or an array of strings, the value of the sub-property must be a JSON object that cointains a property named “value”, and another named “includeNull”. Each of these properties is optional, but at least one of them must be present. The same rules that apply for filtering these types of properties for null values also apply to segmentation.

In the following example, we are requesting segmentation by continent and are also requesting the number of clients where we could not detect the geographical location:

```
{
    "level1": {
        "property": "geography",
        "segments": [
            {
                "type": "regex",
                "continent": {
                                "value": ".*",
                                "includeNull": true
                            }
            }
        ]
    }
}
```

Results Format for Reports Using Date Splitting

The results object contains a number of members having a date as the key and an object as the value. The date is formatted as YYYY-MM-DD regardless of whether the report is being split by day, week, or month. If the report is being split by week or by month, the first date of the week or month is used.

If segmentation level 1 has been specified in the request, each sub-object contains a number of values. The keys are the property values that were specified in the request as level 1. Therefore, if prodVersion was requested as the property of level 1, the keys would be the different product version numbers or the segmentLabel that was specified. The value contains an object with up to 3 members. These are new, active, and lost, referring to new, active, and lost clients within this day, week, or month based on what was specified in the *clientStatus property in the request. Refer to the following example for a sample request and response.

- Full Example Request/Response of Daily Timeline Report with Segmentation
- Results Format for Reports Using Date Splitting with No Segmentation Levels

Full Example Request/Response of Daily Timeline Report with Segmentation

The following example request and response show a timeline report being segmented by selected product versions. Versions 1 and 1.5 are being combined into a single series, while version 2 is in a series on its own.

Example Request

```
POST /reporting/generic/dateRange  HTTP/1.1
Host: api.revulytics.com
Content-Type: application/json
Accept: application/json
```

```
{
    "user": "testuser@test.com",
    "sessionId": "VSB8E2BzSC2eZSJm4QmTpA",
    "productId": 12345678901,
    "dateSplit": "day",
    "startDate": "2018-08-01",
    "stopDate": "2018-08-03",
    "clientStatus": [
        "new",
        "active",
        "lost"
    ],
    "daysUntilDeclaredLost": 30,
    "dateReportedLost": "dateDeclaredLost",
    "globalFilters": {
        "prodLanguage": {
            "type": "stringList",
            "value": ["English", "French"]
        },
        "os": {
            "type": "string",
            "version": "Microsoft Windows 7"
        }
    },
    "levels": {
        "level1": {
            "property": "prodVersion",
            "segments": [
                {
                    "type": "stringArray",
                    "value": [
                        "1",
                        "1.5"
                    ],
                    "segmentLabel": "1 and 1.5",
                    "split": false
                },
                {
                    "type": "string",
                    "value": "2"
                }
            ]
        }
    }
}
```

Example Server Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "status": "OK",
    "results": {
        "2018-08-01": {
```

```
            "2": {
                "active": 80,
                "new": 2,
                "lost": 0
            },
            "1 and 1.5": {
                "active": 133,
                "new": 0,
                "lost": 4
            }
        },
        "2018-08-02": {
            "2": {
                "active": 58,
                "new": 18,
                "lost": 0
            },
            "1 and 1.5": {
                "active": 74,
                "new": 0,
                "lost": 5
            }
        },
        "2018-08-03": {
            "2": {
                "active": 38,
                "new": 20,
                "lost": 0
            },
            "1 and 1.5": {
                "active": 31,
                "new": 0,
                "lost": 1
            }
        }
    }
}
```


Results Format for Reports Using Date Splitting with No Segmentation Levels

If no segmentation levels have been specified in the request, the results object still contains members having a date as the key and an object as the value. However, the sub-object contains the new, active, and lost values or a subset of them.

The following is and example of the results object with only new and active specified in the clientStatus property.


```
{
    "2018-05-01": {
        "active": 643,
        "new": 27
    },
    "2018-05-02": {
        "active": 637,
        "new": 31
    },
```

```
    "2018-05-03": {
        "active": 322,
        "new": 19
    }
}
```

Results Format for Reports Without Date Splitting

The results object contains a hierarchy of sub-objects depending on how many segmentation levels were requested.

If at least 1 level was requested, the results object will contain only 1 member named `level1`. The value of `level1` is a JSON object with a member for each segment of level 1. Therefore, assuming the level 1 property is `prodVersion`, the keys contain product versions or the segment labels that were specified. The value for each of these members will be an object containing an object named totals and if 2 or more levels were requested, another object named `level2`.

The totals object will contain up to 3 members - new, active, and lost, based on what was specified in the `clientStatus` property in the request. If the `level2` object is present, it will be in the same format as `level1`. Similarly, the `level2` object may contain an object named `level3`. Refer to Full Example Request/Response of Report with 2-Level Segmentation for a sample request and response.

- Full Example Request/Response of Report with 2-Level Segmentation
- Results Format for Reports without Date Splitting and with No Segmentation Levels

Full Example Request/Response of Report with 2-Level Segmentation

The following example request and response show a requested report not split by dates but with the data segmented by selected product versions in level 1 and all product editions in level 2. In level 1, product versions 1 and 1.5 are being combined into a single segment, while product version 2 is in a segment on its own. In level 2, each known product edition will be shown in a segment on its own.

Example Request

```
POST /reporting/generic/dateRange  HTTP/1.1
Host: api.revulytics.com
Content-Type: application/json
Accept: application/json

{
    "user": "testuser@test.com",
    "sessionId": "VSB8E2BzSC2eZSJm4QmTpA",
    "productId": 12345678901,
    "dateSplit": null,
    "startDate": "2018-08-01",
    "stopDate": "2018-08-03",
    "clientStatus": [
        "new",
        "active",
        "lost"
    ],
    "daysUntilDeclaredLost": 30,
    "dateReportedLost": "dateDeclaredLost",
    "globalFilters": {
```

```
            "os": {
                "type": "string",
                "version": "Microsoft Windows 7"
            }
        },
        "levels": {
            "level1": {
                "property": "prodVersion",
                "segments": [
                    {
                        "type": "stringArray",
                        "value": [
                            "1",
                            "1.5"
                        ],
                        "segmentLabel": "1 and 1.5",
                        "split": false
                    },
                    {
                        "type": "string",
                        "value": "2"
                    }
                ]
            },
            "level2": {
                "property": "prodEdition",
                "segments": [
                    {
                        "type": "regex",
                        "value": ".*",
                        "split": true
                    }
                ]
            }
        }
    }
```

Example Server Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "status": "OK",
    "levelLabels": {
        "level1": "prodVersion",
        "level2": "prodEdition"
    },
    "results": {
        "level1": {
            "2": {
                "totals": {
                    "new": 40,
                    "active": 138,
                    "lost": 0
                },
```

```
            "level2": {
                "Lite": {
                    "totals": {
                        "new": 5,
                        "active": 30,
                        "lost": 0
                    }
                },
                "Premium": {
                    "total": {
                        "new": 5,
                        "active": 17,
                        "lost": 0
                    }
                },
                "Standard": {
                    "totals": {
                        "new": 30,
                        "active": 91,
                        "lost": 0
                    }
                }
            }
        },
        "1 and 1.5": {
            "totals": {
                "new": 0,
                "active": 174,
                "lost": 10
            },
            "level2": {
                "Lite": {
                    "total": {
                        "new": 0,
                        "active": 48,
                        "lost": 5
                    }
                },
                "Premium": {
                    "totals": {
                        "new": 0,
                        "active": 13,
                        "lost": 1
                    }
                },
                "Standard": {
                    "totals": {
                        "new": 0,
                        "active": 113,
                        "lost": 4
                    }
                }
            }
        }
    }
}
```

```
}
```

Results Format for Reports without Date Splitting and with No Segmentation Levels

If no segmentation levels are specified, the results object will be a simple JSON object with up to 3 members - new, active, and lost, depending on what was requested in the `clientStatus` property in the request. The following is an example of a complete response:

```
{
    "status": "OK",
    "results": {
        "active": 1734,
        "new": 729,
        "lost": 387
    }
}
```

Generic Current Reports

This reporting mechanism is to be used for generating reports regarding the current status of your product user base. Depending on the request, this can create pie/bar charts, geographical maps, or hierarchical tables.

- Request/Response Parameters Summary
- Global Filters
- Segmentation and Levels (Current Reports)
- Results Format

Request/Response Parameters Summary

`POST /reporting/generic/current`

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 8-3 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. Required only for non-cookie authentication. • **sessionId (string)**—The `sessionId` obtained via POST /auth/login. Required only for non-cookie authentication. • **productId (integer)**—The product ID on which this request is being done. • **groupBy (string)**—Optional parameter to specify the property with which to group installations. By default, this value is considered to be `clientId`. Other possible options are `machineId`, `licenseKey` or any custom property of type 3. • **startDate (string)**—The first date of the date range on which to base the report. This is to be formatted as YYYY-MM-DD. • **clientStatus (array)**—The client statuses to show in the report. Each report can show any combination of new, active, and lost users. This is to be presented as an array of up to 3 strings which can be new, active, and lost. • **daysUntilDeclaredLost (integer)**—Required only if the `clientStatus` array contains lost. This specifies the number of consecutive days of inactivity that have to pass until a client installation is declared lost. • **dateReportedLost (string)**—Required only if the `clientStatus` array contains lost. When an installation is lost, it can either be shown as lost on the date it last contacted the Usage Intelligence servers (`dateLastSeen`) or else it can be shown as lost when it was declared lost (last date when it contacted the Usage Intelligence servers + the number of days specified in `daysUntilDeclaredLost`) (`dateDeclaredLost`). Therefore, the permitted values are `dateLastSeen` and `dateDeclaredLost`. • **globalFilters (object)**—JSON object containing the filters to be applied to the available data. Details about these filters can be found in the Global Filters section. • **levels (object)**—Optional JSON object that describes how data is to be split into segments in each level. Requests where `dateSplit` is null can have up to 3 levels while when date splitting is being done, 0 or 1 levels can be specified. Details about levels and segmentation can be found in the Segmentation and Levels (Date-Range Reports) section.
Response JSON Object	• status (string)—Contains OK if successful or `SYNTAX ERROR` or `AUTH ERROR`. • reason (string)—Present only if status is not OK. Contains error message (reason). • **levelLabels (object)**—Present only on responses where `dateSplit` is null. When present, this contains the property names of each level as requested. • **results (object)**—Contains the results as requested represented as a JSON object. The result format is described below.

Global Filters

Most of the available filter properties are string-based. This means that when applying a filter, the requested field can be represented as a string, stringArray or regex. There are also some filters which are numeric. These filters should be represented as number or numberRange.

String-Based Filters

The following properties are stored as strings:

```
machineId
clientId
prodVersion
prodEdition
prodBuild
prodLanguage
licenseType
formFactor
osLanguage
osWordLength
cpuType
dotNetVersion
javaVersion
javaVendor
javaRuntime
javaGraphics
javaVmVersion
javaVmName
vm
C01 .. C20 (Custom properties)
licenseKey
```

Note • *`licenseKey` requires a special user permission to be used as a filter.*

The **type** field in the above filters needs to be string, `stringArray` or `regex`. A **value** field is always required. The contents of this field should be according to the specified type.

- If `string` is specified, then the value field must contain a single string that needs to be matched precisely with the stored data.
- If `stringArray` is specified, then the value field must contain an array of strings where one of which needs to match precisely with the stored data.

- If specifying a regex, the value field should contain a string which is treated as a regular expression and the stored data will be matched against it using regular expression rules.

Example Filter Using a String Value

In this example, the product build value needs to be exactly "3014.int-12214":

```
{
      "prodBuild":
          {
              "type": "string",
              "value": "3014.int-12214"
          }
 }
```

Example Filter Using a String Array

In this example, the product build value needs to be either "3014.int-12214", "3017.enx-57718", or "4180.vrx-81059". Note that since the type is declared as `stringArray`, the value field needs to contain an array. Consider all elements in the array to have an OR logical expression between them.:

```
{
    "prodBuild":
        {
            "type": "stringArray",
            "value": ["3014.int-12214", "3017.enx-57718", "4180.vrx-81059"]
        }
}
```

Example Filter Using a Regular Expression

In this example, the product build value needs to start with "30" and end with "18" whilst having 10 characters in between:

```
{
     "prodBuild":
         {
             "type": "regex",
             "value": "^30.{10}18$"
         }
}
```

Numeric Filters

The following properties are stored as numeric values:

```
cpuCores
displayCount
ram
resolutionWidth
resolutionHeight
lifetimeRuntimeMinutes
lifetimeSessionCount
screenPpi
javaVmRam
```

The **type** field in the above filters needs to be `number` or `numberRange`.

- If `number` is specified, then a value field must also be present. The value field should contain a number, which may contain a decimal point if required.
- If `numberRange` is specified, then the value field should NOT be used. Instead, the properties min and max are to be used. These refer to the minimum and maximum number to be included in the report. If only one limit needs to be set, the other property is to be left out. Therefore, if you want to include installations with up to 2 display devices, you would not specify a min value, but instead specify only a max and set it as 2.

Example Filter Using a Number Value

In this example, the number of display devices needs to be exactly 3:

```
{
      "displayCount":
          {
              "type": "number",
              "value": 3
          }
 }
```

Example Filter Using a Number Range Value

In this example, the RAM needs to be between 1025MB and 4096MB (both included):

```
{
      "ram":
          {
              "type": "numberRange",
              "min": 1025,
              "max": 4096
          }
 }
```

Boolean Filters

The following property is stored as boolean:

`touchScreen`

The type field in the above filters needs to be boolean. The value must be true or false. In the following filter, clients with a touch screen are being requested.

```
{
    "touchScreen":
        {
            "value": true
        }
}
```

Date Range Filters

The following properties are stored as dates:

`dateInstalled`
`dateLastSeen`

The type field in the above filters needs to be `date` or `dateRange`.

- If `date` is specified, then a value field must also be present. The value field should contain a date.
- If `dateRange` is specified, then the value field should NOT be used. Instead, the properties min and max are to be used. These refer to the minimum and maximum dates to be included in the report. If only one limit needs to be set, the other property is to be left out.

In the following example, users installed after January 1st 2018 are to be shown:

```
{
    "dateInstalled":
        {
            "type": "dateRange",
            "min": "2018-01-01"
        }
}
```

Note that all dates must be in ISO 8601 format.

Special Filters

Some filters need to be represented in a special format due to their unique requirements. These special filters are:

- Special Filter: licenseStatus
- Special Filter: os
- Special Filter: geography
- Special Filter: gpu
- Special Filters: optOut and backOff
- Special Filter: lifetimeEventUsage
- Special Filter: reachOutDeliveries

Special Filter: licenseStatus

The `licenseStatus` filter is made up of 4 sub-values: `activated`, `blacklisted`, `expired` and **whitelisted**. These are presented as boolean values. Unlike other filters, this filter is presented as an array of JSON objects. Each object can contain a subset (or all) of these 4 boolean values.

Consider the following example. In this example, for a client to be included, the license has to either be activated AND whitelisted, or else it can be not whitelisted AND expired. In other words, **(** (activated AND whitelisted) OR ((NOT)whitelisted AND expired) **)**.

```
{
    "licenseStatus":
        [
            {
                "activated": true,
                "whitelisted": true
            },
            {
                "whitelisted": false,
```

```
                "expired": true
            }
        ]
}
```

Special Filter: os

The os filter is made up of 3 granularity levels. These are platform, version, and edition. These are meant to split the OS name into levels of detail as required by the user. Consider the following:

- **platform**: Microsoft Windows
- **version**: Microsoft Windows 7
- **edition**: Microsoft Windows 7 Professional

If a filter is set on the version "Microsoft Windows 7", the result would include all editions of Windows 7. One or more of these granularity levels may be specified. If more than 1 granularity level is specified, the values are ORed together.

In the following example, all editions of "Microsoft Windows 7" are included, and also "Microsoft Windows Vista Home Premium":

```
{
    "type": "string",
    "version": "Microsoft Windows 7",
    "edition": "Microsoft Windows Vista Home Premium"
}
```

In the following example, the type is `stringArray`. Note that an array needs to be passed if the type is set as such, even if it is to contain only 1 element. In this case, the version can be either "Microsoft Windows 7" or "Microsoft Windows 8" (which are ORed together). Also, clients running on "Microsoft Windows XP Professional" are to be included.

```
{
    "type": "stringArray",
    "version": ["Microsoft Windows 7", "Microsoft Windows 8"],
    "edition": ["Microsoft Windows XP Professional"]
}
```

Special Filter: geography

The geography filter is made up of 3 granularity levels. These are `continent`, `country`, and `usState`. The `usState` value applies only to United States. Continents and countries are presented in 2-letter codes. Countries follow ISO standard 3166-1 alpha-2. US states are presented in ISO 3166-2:US format.

In the following example, the clients have to be either:

- In the continents Asia or Oceania
- In the country Germany
- In the US states New York, New Jersey, or Kansas

```
{
    "type": "stringArray",
    "continent": ["AS", "OC"],
    "country": ["DE"],
    "usState": ["US-NY", "US-NJ", "US-KS"]
```

```
}
```

Important • *In this filter, the type can be* `string` *or* `stringArray`*. Regular expressions are not supported in geography filters.*

Special Filter: gpu

The gpu filter is made up of 2 granularity levels. These are `vendor` and `model`. Both are represented as string values.

In the following example, the clients have to have a GPU:

- From the vendors `NVIDIA` or `Intel`
- With the model `AMD Radeon HD 4600`

```
{
    "type": "stringArray",
    "vendor": ["NVIDIA", "Intel"],
    "model": ["AMD Radeon HD 4600"]
}
```

Special Filters: optOut and backOff

The opt-out mechanism was introduced in SDK version 5.1.0. With this feature, vendors can have their application report to the Usage Intelligence servers that a user does not want to be tracked. Using this property, vendors can filter out installations that were opted-out.

Similarly, backoff filtering was introduced with version 5.0.0. Backoff is when a product account runs over-quota and the server starts rejecting data. Although filtering for backed-off installations was introduced with version 5, it was also backported to previous versions. However, when a new installation with an SDK prior to version 5 tries to register with the server and is rejected, it is not marked as being once backed-off when it is eventually accepted by the server. With version 5 onwards, the server flags an installation as being historically backed-off in such cases.

Both `backOff` and `optOut` filters are made up of 2 boolean sub-values: `historical` and `current`.

- The `historical` value refers to installations that were once backed-off or opted-out. These may include installations that are still currently backed-off or opted-out.
- The `current` value refers to the status during the last time that the client called the server.

Therefore, if an installation was opted-out yesterday but got opted-in today, it will be marked as historically opted-out but not currently opted-out.

In the following example, for a client to be included, the `optOut` status has to either be `historical` AND not `current`, or else it can be not `historical` (i.e. users have to be currently opted-in but used to be opted-out at some point or never opted out).

```
{
    "optOut":
        [
            {
                "historical": true,
                "current": false
            },
            {
```

```
                "historical": false
            }
        ]
}
```

Special Filter: lifetimeEventUsage

Using lifetime event usage filters, clients can be filtered based on whether a particular event or set of events occurred or not within the client's lifetime. Alternatively, one can set a filter based on the number of times an event has occurred.

In the following example, clients that are included must have done the "File Operations - Open" event at least 5 times to be counted.

```
{
     "category": "File Operations",
     "name": "Open",
     "min": 5
}
```

In the following example, clients must have done either "File Operations - Open" or "File Operations - Save" for a combined total of between 10 to 50 times.

```
{
    "combiArray": [
                    {
                      "categoryType": "string",
                      "nameType": "string",
                      "category": "File Operations",
                      "name": "Open"
                    },
                    {
                      "categoryType": "string",
                      "nameType": "string",
                      "category": "File Operations",
                      "name": "Save",
                    }
                ],
    "min": 10,
    "max": 50
}
```

In the following example, clients must have done any event within the "File Operations" category for a combined total of not more than 100 times. This is done using a regular expression in the name.

```
{
    "combiArray":  [
                     {
                       "categoryType": "string",
                       "nameType": "regex",
                       "category": "File Operations",
                       "name": ".*"
                     }
                  ],
    "max": 100
}
```

Special Filter: reachOutDeliveries

Using ReachOut delivery filters, clients can be filtered based on whether a particular ReachOut message or a combination of ReachOut messages were delivered or not within the client's lifetime.

The filter consists of a JSON array that includes one or more objects. Each object is a combination of delivered and undelivered campaigns, and the different combinations are ORed together. Therefore, it is possible to show users that either received ReachOut message 1 but not 2, or else received 3 but not 4 as in the following example:

In the following example, we are looking for clients who either received campaign 1 but not 2, OR received campaign 2 but not 3.

```
[
     {"auto": {"delivered": ["1"], "undelivered":["2"]}},
     {"auto": {"delivered": ["2"], "undelivered":["3"]}}
]
```

The above example contains only "auto" ReachOut campaigns. Manual campaigns can be specified using "manual" instead of "auto" as in the above example. Each object can contain a mix of "auto" and "manual" campaigns.

<NULL> Values in Global Filters (Current Reports)

Most of the available properties can include null values. There are different reasons why a value would be null. When these are properties that are set by the application, the possible reasons why a value would be null are: cases where the value has not been set by the application (such as prodBuild never being set), and cases where values are set to an empty string (“”) or to a string containing “<NULL>”.

One other reason is that although these values have been set, the SDK has not yet had time to sync with the servers to provide this new information.

In cases where the properties are set automatically such as hardware or OS related information, the values would be null if the SDK failed to retrieve that value from the OS or if the server failed to identify the value retrieved by the SDK.

Other reasons include cases where Java version is requested from an application that does not use the Java SDK, US state is requested for users who are not running within the US, etc.

The following are the properties that support null values:

```
prodVersion
prodEdition
prodBuild
prodLanguage
machineId
formFactor
vm
cpuType
cpuCores
ram
resolutionWidth
resolutionHeight
javaVersion
javaVmVersion
javaVmName
javaVendor
javaRuntime
javaGraphics
```

```
osLanguage
licenseKey
C01 .. C20 (Custom properties)
os
geography
gpu
```

Null values can be requested either on their own or as part of a filter containing other values.

The following example would return only cases where the `prodVersion` is null:

```
{
      "prodVersion":
          {
              "includeNull": true
          }
}
```

The following example would return cases where the `prodVersion` is either 1.1, 1.2 or null:

```
{
      "prodVersion":
          {
              "type": "stringArray",
              "value": ["1.1", "1.2"],
              "includeNull": true
          }
}
```

By default, when specifying a filter, null values would not be included. Therefore, in the following example, only clients with `prodVersion` set to 1.1 or 1.2 will be included, while null values are excluded:

```
{
      "prodVersion":
          {
              "type": "stringArray",
              "value": ["1.1", "1.2"]
          }
}
```

However, if no filter is specified, then nulls are included by default. Therefore, if you want to include any value of `prodVersion` as long as it is not null, a `prodVersion` filter needs to be included as follows:

```
{
      "prodVersion":
          {
              "type": "regex",
              "value": ".*",
              "includeNull": false
          }
}
```

In the case of filters that use sub-properties (`os`, `geography`, and `gpu`), the `includeNull` filter is to be included in the sub-property and applies to that specific sub-property only. In order to be able to include the `includeNull` property, instead of providing the value as a string or an array of strings, the value of the sub-property must be a JSON object that cointains a property named "`value`", and another named "`includeNull`". Each of these properties is optional, but at least one of them must be present.

In the case of geography, this has a very particular meaning. Requesting for null "country" value does not return all cases where the country could not be retrieved, but only cases where the continent could be retrieved but the country could not. Similarly, requesting null "usState" returns cases where the continent and country could be retrieved but the US state could not. This does not include clients that are not situated in the US. If you are interested in finding clients where we could not detect any geographical data, the `includeNull` filter needs to be applied in the continent sub-property.

In the following example, we are requesting cases where we know that the client is within the US but the state could not be identified:

```
{
      "geography":
          {
              "type": "string",
              "country": "US",
              "usState":
                  {
                      "includeNull": true
                  }
          }
}
```

In the following example, we are requesting cases where the GPU is either "NVIDIA", "AMD" or null (unidentified):

```
{
      "gpu":
          {
              "type": "stringArray",
              "vendor":
                  {
                      "value": ["NVIDIA", "AMD"],
                      "includeNull": true
                  }
          }
}
```

Segmentation and Levels (Current Reports)

Report data may be segmented in up to 3 levels. Each level represents a property. Using multi-level segmentation, you can create hierarchical reports that are used to drill down based on the property values. For example, you can create a report that splits all the users based on which edition they are using. Then, you can split the user counts for each edition based on product version on level 2. Finally, the user counts for each product version can be split up by product build on level 3.

Segmentation is optional. In order not to split data by any property, you may either not include a levels property in the request object, or else leave the levels object empty. The following examples show the difference between not requesting any segmentation, requesting a single level, and requesting 2 levels:

Consider the following examples:

Table 8-4 • Segmentation and Levels

Number of Levels	Description	
0 Levels	• New Users: 30 • Active Users: 100 • Lost Users: 20	

1 Level	**Product Versions:**
	• **Version 1**:
	  • New Users: 10
	  • Active Users: 30
	  • Lost Users: 8
	• **Version 2**:
	  • New Users: 20
	  • Active Users: 70
	  • Lost Users: 12	

2 Levels	**Product Versions:**
	• **Version 1**:
	  • New Users: 10
	  • Active Users: 30
	  • Lost Users: 8
		Product Editions:
		• **Premium:**
		  • New Users: 4
		  • Active Users: 13
		  • Lost Users: 1
		• **Standard:**
		  • New Users: 6
		  • Active Users: 17
		  • Lost Users: 7

	Product Versions:
	• **Version 2**:
	  • New Users: 20
	  • Active Users: 70
	  • Lost Users: 12
		Product Editions:
		• **Premium:**
		  • New Users: 5
		  • Active Users: 40
		  • Lost Users: 4
		• **Standard:**
		  • New Users: 15
		  • Active Users: 30
		  • Lost Users: 8

In the above examples, the first example (0 Levels) is showing a case where no segmentation is being applied.

The second example (1 Level) is showing a response where a single level of segmentation has been requested. In this case, segmentation is being done based on product versions. For each product version, one can see the number of new, active, and lost users from the specified start date up till now.

The third example (2 Levels) shows 2 levels of segmentation. In this example, one can see how many new, active, and lost users were using each version, and then, that data is further split by product edition. A further level is also allowed, so, for example one may choose to segment each product edition by product language.

The properties that are available for segmentation are the same ones that are used for Global Filters. There are 3 properties that require special formatting. These are described below.

Segment levels are to be defined in a property named "levels". This property should contain a JSON object which contains 2 members:

- **property (string)**—The name of the property by which to segment. Note that in case of `os`, `geography`, `licenseStatus` and `gpu`, a special format is used.
- **segments (array)**—An array containing a number of JSON objects. The format of these object is described in the Level Segments Format section below.
- **sort (string)**—Optional property to specify how the segments in this level are to be sorted. Possible values are alpha, new, active, and lost. alpha refers to alphabetical sorting which is based on the segment label. The other 3 are based on the client statuses. Note that if sorting by new, active, or lost users, that particular client status must be included in the `clientStatus` array. If this property is not included, the data is sorted alphabetically by default.
- **sortDirection (string)**—Optional property to specify whether to sort in ascending or descending order. Possible values are `asc` and `desc`. If not specified, data is sorted in ascending order by default.

For more information, see:

- Level Segments Format
- String-Based Segmentation Properties
- Numeric Segmentation Properties
- Boolean Segmentation Properties
- Special Segmentation Properties
- <NULL> Values in Segmentation and Levels (Current Reports)

Level Segments Format

Segments are defined as JSON objects. A single JSON object may create a single item on a table, or it may create a number of items/series if splitting is enabled. Each object should contain the following:

- **type (string)**—The data type of the value. Can be `string`, `stringArray`, `regex`, `number` or `numberRange` based on whether the property is string-based (String-Based Filters) or numeric (Numeric Filters).
- value (string/array/number)—An exact string, an array of strings, a regular expression or a numeric value. This property should not be used if type is `numberRange`. Format is based on whether the property is string-based (String-Based Filters) or numeric (Numeric Filters).
- min (number)—Used only if the type is `numberRange`. Contains the minimum numeric value to include in this segment. May be combined with max.

- max (number)—Used only if the type is `numberRange`. Contains the maximum numeric value to include in this segment. May be combined with min.
- split (boolean)—Used only if the type is `stringArray` or `regex`. This specifies whether to split the returned data based on each different value matched by the regular expression or array (true), or to join all the clients that match the value as 1 table value or series (false).
- segmentLabel (string)—Used only if split is false or if type is `numberRange`. This is required to give a name to a series when not splitting by value. It is important that the name given is unique.
- limit (integer)—Optional property to set the limit on the maximum number of table values or series that should be produced by this set of values. To be used only if split is true.

String-Based Segmentation Properties

The following properties are stored as strings:

```
machineId
clientId
prodVersion
prodEdition
prodBuild
prodLanguage
licenseType
formFactor
osLanguage
osWordLength
cpuType
javaVersion
javaVendor
javaRuntime
javaGraphics
javaVmVersion
javaVmName
vm
C01 .. C20 (Custom properties)
licenseKey
```

Note • *licenseKey requires a special user permission to be used for segmentation*

The type field when using one of the above properties needs to be `string`, `stringArray` or `regex`. A value field is always required. The contents of this field should be according to the specified type.

- If `string` is specified, then the value field must contain a single string that needs to be matched precisely with the stored data.
- If `stringArray` is specified, then the value field must contain an array of strings where one of which needs to match precisely with the stored data.
- If specifying a `regex`, the value field should contain a string which is treated as a regular expression and the stored data will be matched against it using regular expression rules.

Example Using 1-Level Segmentation by string, stringArray, and regex Values

```
{
    "level1": {
        "property": "prodVersion",
        "segments": [
            {
                "type": "string",
                "value": "1.0"
            },
            {
                "type": "stringArray",
                "value": ["2.0", "2.1", "3.1"],
                "split": false,
                "segmentLabel": "Versions 2 and 3"
            },
            {
                "type": "regex",
                "value": "^4\..*",
                "split": false,
                "segmentLabel": "All version 4"
            },
            {
                "type": "regex",
                "value": "^5\..*",
                "split": true
            }
        ]
    }
}
```

In the above example, we are requesting a report with multiple segments. The first segment contains installations running version 1.0. Notice how this does not require a "split" property since there is only 1 value and therefore no further splitting is possible. The second segment contains versions 2.0, 2.1 and 3.1. In this case, the "split" property is required, and since we are requesting the API to combine these 3 versions, we must provide a "segmentLabel" value so that the returned data can be identified. The third segment is similar, although in this case the request is built using a regular expression. In this case, all versions starting with "4." are to be included into one combined segment.

The last segment is different from the rest because we are requesting the API to split the data (split is set to true). Therefore, this can produce much more than 1 segment. In this case, we could see segments such as "5.1", "5.2", etc. Notice how since we are splitting, we should not provide a segmentLabel value since the labels are built using the different values that are found in the data.

Numeric Segmentation Properties

The following properties are stored as numeric values:

```
cpuCores
displayCount
ram
resolutionWidth
resolutionHeight
lifetimeRuntimeMinutes
lifetimeSessionCount
screenPpi
```

`javaVmRam`

The `type` field in the above properties needs to be number or `numberRange`. If number is specified, then a value field must also be present. The value field should contain a number, which may contain a decimal point if required. If `numberRange` is specified, then the value field should NOT be used. Instead, the properties min and max are to be used. These refer to the minimum and maximum number to be included in the report. If only one limit needs to be set, the other property is to be left out. Therefore, if you want to include installations with up to 2 display devices, you would not specify a min value, but instead specify only a max and set it as 2.

Example Using 1-Level Segmentation by number, and numberRange Values

```
{
    "level1": {
        "property": "cpuCores",
        "segments": [
            {
                "type": "number",
                "value": 1,
            },
            {
                "type": "numberRange",
                "min": 2,
                "max": 4,
                "segmentLabel": "2 - 4"
            },
            {
                "type": "numberRange",
                "min": 5,
                "segmentLabel": "5 +"
            }
        ]
    }
}
```

In the above example, we are requesting a report with 3 segments. The first segment contains only installations running on 1 CPU core, the second segments contains installations running on 2, 3, or 4 cores (range 2 - 4), while the last segment contains all installations which are running on a machine with 5 or more CPU cores. Note how when the type was `numberRange`, we had do specify a `segmentLabel` which is a free string that will be used by the user to identify what is being included in that specific segment.

Boolean Segmentation Properties

The following properties are stored ad boolean values:

`touchScreen`

The type field needs to be boolean, and the value must be true or false. A `segmentLabel` field is also required

Example Using 1-Level Segmentation by Boolean Value

```
{
    "level1": {
        "property": "touchScreen",
        "segments": [
            {
```

```
                "type": "boolean",
                "value": true,
                "segmentLabel": "Yes"
            },
            {
                "type": "boolean",
                "value": false,
                "segmentLabel": "No"
            },
            {
                "includeNull": true,
                "segmentLabel": "Unknown"
            }
        ]
    }
}
```

In the above example, we are requesting a report with 3 segments. The first segment contains installations on which a touch screen was detected, the second one where no touch screen has been detected, while the last one is where we could not detect whether a touch screen is present due to the client using an old SDK which did not have touch screen detection support.

Special Segmentation Properties

Some properties need to be represented in a special format due to their unique requirements. These special properties are:

`licenseStatus`
`os`
`geography`
`gpu`
`optOut and backOff`

For more information, see:

- Special Segmentation Format: licenseStatus
- Special Segmentation Format: os
- Special Segmentation Format: geography
- Special Segmentation Format: gpu
- Special Segmentation Format: optOut and backOff

Special Segmentation Format: licenseStatus

The `licenseStatus` value is made up of 4 sub-values: `activated`, `blacklisted`, `expired` and `whitelisted`. These are presented as boolean values. Any number of segments can be defined, and each segment can contain any subset of the 4 sub-values. These values are ANDed together. A `segmentLabel` value is required.

In the following example, 2 segments are specified - the first one showing blacklisted AND not expired and the second one showing whitelisted AND activated:

```
[
    {
        "segmentLabel": "BL and not EXP",
```

```
        "blacklisted": true,
        "expired": false
    },
    {
        "segmentLabel": "WL and ACT",
        "whitelisted": true,
        "expired": true
    }
]
```

Special Segmentation Format: os

The os value is made up of 3 granularity levels - `platform`, `version`, and `edition`. A particular level needs to be selected, and this is to be included in the property name such as os.version or os.edition. Different granularity levels can be requested for different segmentation levels. Therefore, it is possible to generate a 3-level hierarchical tree in which level 1 would show the OS platform, level 2 would show the version, and level 3 would show the full name including the OS edition or sub-version. For a description of the differences between the 3 granularity levels, refer to the os special filter section.

The following example shows the levels object requesting 3 granularity levels as described above:

```
{
    "level1": {
        "property": "os.platform",
        "segments": [
            {
                "type": "regex",
                "value": ".*",
                "split": true
            }
        ]
    },
    "level2": {
        "property": "os.version",
        "segments": [
            {
                "type": "regex",
                "value": ".*",
                "split": true
            }
        ]
    },
    "level3": {
        "property": "os.edition",
        "segments": [
            {
                "type": "regex",
                "value": ".*",
                "split": true
            }
        ]
    }
}
```

In the above example, no filtering is being done, and instead, a regular expression to include everything is set as the value. This will result in all OS platforms, versions, and editions to be included in the hierarchy.

Special Segmentation Format: geography

The geography value is made up of 3 granularity levels - `continent`, `country`, and `usState`. These granularity levels are explained in Special Filter: geography. A particular level needs to be selected, and this is to be included in the property name such as geography.continent or geography.country. Different granularity levels can be requested for different segmentation levels. Therefore, it is possible to generate a 3-level hierarchical tree in which level 1 would show the continent, level 2 would show the country, and level 3 would show the US state (for United States only).

The following example shows the levels object requesting 3 granularity levels as described above:

```
{
    "level1": {
        "property": "geography.continent",
        "segments": [
            {
                "type": "regex",
                "value": ".*",
                "split": true
            }
        ]
    },
    "level2": {
        "property": "geography.country",
        "segments": [
            {
                "type": "regex",
                "value": ".*",
                "split": true
            }
        ]
    },
    "level3": {
        "property": "geography.usState",
        "segments": [
            {
                "type": "regex",
                "value": ".*",
                "split": true
            }
        ]
    }
}
```

In the above example, no filtering is being done, and instead, a regular expression to include everything is set as the value. This will result in all the continents, countries and US states (where applicable) to be included in the hierarchy.

Special Segmentation Format: gpu

The gpu value is made up of 2 granularity levels - `vendor` and `model`. These granularity levels are explained in Special Filter: gpu. A particular level needs to be selected, and this is to be included in the property name, namely gpu.vendor or gpu.model. Different granularity levels can be requested for different segmentation levels. Therefore, it is possible to generate a 2-level hierarchical tree in which level 1 would show the vendor, and level 2 would show the model number.

The following example shows the levels object requesting 2 granularity levels as described above:

```
{
    "level1": {
        "property": "gpu.vendor",
        "segments": [
            {
                "type": "regex",
                "value": ".*",
                "split": true
            }
        ]
    },
    "level2": {
        "property": "gpu.model",
        "segments": [
            {
                "type": "regex",
                "value": ".*",
                "split": true
            }
        ]
    }
}
```

In the above example, no filtering is being done, and instead, a regular expression to include everything is set as the value. This will result in all the GPU vendors and models to be included in the hierarchy.

Special Segmentation Format: optOut and backOff

Both `backOff` and `optOut` values are made up of 2 boolean sub-values: `historical` and `current`. Any number of segments can be defined, and each segment can contain any subset of the 2 sub-values. These values are ANDed together. A `segmentLabel` value is required.

In the following example, 2 segments are specified - the first one showing historical AND not current and the second one showing not historical (i.e. never opted-out):

```
[
    {
        "segmentLabel": "HISTORICAL and not CURRENT",
        "historical": true,
        "current": false
    },
    {
        "segmentLabel": "Never opted-out",
        "historical": false
    }
]
```

<NULL> Values in Segmentation and Levels (Current Reports)

Null values in segmentation are to be requested in a similar way to null values in filters, as described in <NULL> Values in Global Filters (Current Reports). The same properties that support null in filtering also support null in segmentation.

By default, when segmenting, null values are not included within the segments, since only the values that have been specified in each segment are included. Null values don't match any regular expression, so the only way to request null values to be included is to specify "includeNull" as true in a similar way to filtering. In segmentation, null values are returned as "<NULL>". The API considers all cases where the data has never been set from the SDK, set as an empty string, or set as a string containing "<NULL>" to be the same.

The following example requests all values of `prodBuild` including null:

```
{
    "level1": {
        "property": "prodBuild",
        "segments": [
            {
                "type": "regex",
                "value": ".*",
                "includeNull": true
            }
        ]
    }
}
```

In the case of segmentation properties that use sub-properties (os, geography, and gpu), the `includeNull` value is to be included in the sub-property and applies to that specific sub-property only. In order to be able to include the `includeNull` property, instead of providing the value as a string or an array of strings, the value of the sub-property must be a JSON object that contains a property named "value", and another named "includeNull". Each of these properties is optional, but at least one of them must be present. The same rules that apply for filtering these types of properties for null values also apply to segmentation.

In the following example, we are requesting segmentation by continent and are also requesting the number of clients where we could not detect the geographical location:

```
{
    "level1": {
        "property": "geography",
        "segments": [
            {
                "type": "regex",
                "continent": {
                                "value": ".*",
                                "includeNull": true
                            }
            }
        ]
    }
}
```

Results Format

The results object contains a hierarchy of sub-objects depending on how many segmentation levels were requested.

If at least 1 level was requested, the results object will contain only 1 member named level1. The value of `level1` is a JSON object with a member for each segment of level 1. Therefore, assuming the level 1 property is `prodVersion`, the keys contain product versions or the segment labels that were specified. The value for each of these members will be an object containing an object named totals and if 2 or more levels were requested, another object named `level2`.

The `totals` object will contain up to 3 members - new, active, and lost, based on what was specified in the `clientStatus` property in the request. If the level2 object is present, it will be in the same format as level1. Similarly, the `level2` object may contain an object named level3.

Refer to the following example for a sample request and response.

- Full Example Request/Response of Daily Timeline Report with Segmentation
- Results Format for Reports with No Segmentation Levels

Full Example Request/Response of Daily Timeline Report with Segmentation

The following example request and response show a report with the data being segmented by selected product versions in level 1 and all product editions in level 2. In level 1, product versions 1 and 1.5 are being combined into a single segment, while product version 2 is in a segment on its own. In level 2, each known product edition will be shown in a segment on its own.

Example Request

```
POST /reporting/generic/current  HTTP/1.1
Host: api.revulytics.com
Content-Type: application/json
Accept: application/json

{
    "user": "testuser@test.com",
    "sessionId": "VSB8E2BzSC2eZSJm4QmTpA",
    "productId": 12345678901,
    "startDate": "2018-08-01",
    "clientStatus": [
        "new",
        "active",
        "lost"
    ],
    "daysUntilDeclaredLost": 30,
    "dateReportedLost": "dateDeclaredLost",
    "globalFilters": {
        "prodLanguage": {
            "type": "stringList",
            "value": ["English", "French"]
        },
        "ram": {
            "type": "numberRange",
            "min": 512
        },
        "os": {
            "type": "string",
            "version": "Microsoft Windows 7"
        }
    },
    "levels": {
        "level1": {
            "property": "prodVersion",
            "segments": [
                {
```

```
                    "type": "stringArray",
                    "value": [
                        "1",
                        "1.5"
                    ],
                    "segmentLabel": "1 and 1.5",
                    "split": false
                },
                {
                    "type": "string",
                    "value": "2"
                }
            ]
        },
        "level2": {
            "property": "prodEdition",
            "segments": [
                {
                    "type": "regex",
                    "value": ".*",
                    "split": true
                }
            ]
        }
    }
}
```

Example Server Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "status": "OK",
    "levelLabels": {
        "level1": "prodVersion",
        "level2": "prodEdition"
    },
    "results": {
        "level1": {
            "2": {
                "totals": {
                    "new": 40,
                    "active": 138,
                    "lost": 0
                },
                "level2": {
                    "Lite": {
                        "totals": {
                            "new": 5,
                            "active": 30,
                            "lost": 0
                        }
                    },
                    "Premium": {
                        "total": {
```

```
                        "new": 5,
                        "active": 17,
                        "lost": 0
                    }
                },
                "Standard": {
                    "totals": {
                        "new": 30,
                        "active": 91,
                        "lost": 0
                    }
                }
            },
            "1 and 1.5": {
                "totals": {
                    "new": 0,
                    "active": 174,
                    "lost": 10
                },
                "level2": {
                    "Lite": {
                        "total": {
                            "new": 0,
                            "active": 48,
                            "lost": 5
                        }
                    },
                    "Premium": {
                        "totals": {
                            "new": 0,
                            "active": 13,
                            "lost": 1
                        }
                    },
                    "Standard": {
                        "totals": {
                            "new": 0,
                            "active": 113,
                            "lost": 4
                        }
                    }
                }
            }
        }
    }
}
```


Results Format for Reports with No Segmentation Levels

If no segmentation levels are specified, the results object will be a simple JSON object with up to 3 members - new, active, and lost, depending on what was requested in the `clientStatus` property in the request. The following is an example of a complete response:


```
{
```

```
    "status": "OK",
    "results": {
        "active": 1734,
        "new": 729,
        "lost": 387
    }
}
```

9

User Engagement Histograms

The User Engagement Histogram Report consists of 3 histograms which show the following metrics:

- **Active Days**—The number of days clients were active within the specified date range
- **Sessions**—The number of times users launched your application
- **Runtime**—The total amount of time in hours users spent interacting with your application

For more information see:

- Request/Response Parameters Summary
- Global Filters
- Results Format

Request/Response Parameters Summary

```
POST /reporting/engagement/usageDistribution
```

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 9-1 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. Required only for non-cookie authentication. • **sessionId (string)**—The `sessionId` obtained via POST /auth/login. Required only for non-cookie authentication. • **productId (integer)**—The product ID on which this request is being done • **startDate (string)**—The first date of the date range on which to base the report. This is to be formatted as YYYY-MM-DD. • **globalFilters (object)**—JSON object containing the filters to be applied to the available data. Details about these filters can be found in the Global Filters section.
Response JSON Object	• status (string)—Contains OK if successful or SYNTAX ERROR or AUTH ERROR. • reason (string)—Present only if status is not OK. Contains error message (reason). • **results (object)**—Contains the results as requested represented as a JSON object. The result format is described below.

Global Filters

Most of the available filter properties are string-based. This means that when applying a filter, the requested field can be represented as a `string`, `stringArray` or regex. There are also some filters which are numeric. These filters should be represented as `number` or `numberRange`.

- String-Based Filters
- Numeric Filters
- Date Range Filters
- Boolean Filters
- Special Filters
- <NULL> Values in Global Filters

String-Based Filters

The following properties are stored as strings:

```
machineId *
clientId *
prodVersion
prodEdition
prodBuild
```

```
prodLanguage
licenseType
formFactor *
osLanguage
osWordLength *
cpuType *
dotNetVersion *
javaVersion *
javaVendor *
javaRuntime *
javaGraphics *
javaVmVersion *
javaVmName *
vm *
C01 .. C20 (Custom properties)
licenseKey *
```

Note • *`licenseKey` requires a special user permission to be used as a filter.*

Note • *Properties marked with an asterisk (*) are based on the current (latest known) values.*

The **type** field in the above filters needs to be `string`, `stringArray` or `regex`. A **value** field is always required. The contents of this field should be according to the specified type.

- If `string` is specified, then the value field must contain a single string that needs to be matched precisely with the stored data.
- If `stringArray` is specified, then the value field must contain an array of strings where one of which needs to match precisely with the stored data.
- If specifying a `regex`, the value field should contain a string which is treated as a regular expression and the stored data will be matched against it using regular expression rules.

Example Filter Using a String Value

In this example, the product build value needs to be exactly “3014.int-12214”:

```
{
      "prodBuild":
          {
              "type": "string",
              "value": "3014.int-12214"
          }
}
```

Example Filter Using a String Array

In this example, the product build value needs to be either “3014.int-12214”, “3017.enx-57718”, or “4180.vrx-81059”. Note that since the type is declared as stringArray, the value field needs to contain an array. Consider all elements in the array to have an OR logical expression between them.:

```
{
    "prodBuild":
```

```
        {
            "type": "stringArray",
            "value": ["3014.int-12214", "3017.enx-57718", "4180.vrx-81059"]
        }
}
```

Example Filter Using a Regular Expression

In this example, the product build value needs to start with “30” and end with “18” whilst having 10 characters in between:

```
{
     "prodBuild":
         {
             "type": "regex",
             "value": "^30.{10}18$"
         }
}
```

Numeric Filters

The following properties are stored as numeric values:

```
cpuCores *
displayCount *
ram *
resolutionWidth *
resolutionHeight *
lifetimeRuntimeMinutes *
lifetimeSessionCount *
screenPpi *
javaVmRam *
```

Note • *Properties marked with an asterisk (*) are based on the current (latest known) values.*

The type field in the above filters needs to be number or numberRange.

- If `number` is specified, then a value field must also be present. The value field should contain a number, which may contain a decimal point if required.
- If `numberRange` is specified, then the value field should NOT be used. Instead, the properties min and max are to be used. These refer to the minimum and maximum number to be included in the report. If only one limit needs to be set, the other property is to be left out. Therefore, if you want to include installations with up to 2 display devices, you would not specify a min value, but instead specify only a max and set it as 2.

Example Filter Using a Number Value

In this example, the number of display devices needs to be exactly 3:

```
{
      "displayCount":
          {
              "type": "number",
              "value": 3
          }
```

```
}
```

Example Filter Using a Number Range Value

In this example, the RAM needs to be between 1025MB and 4096MB (both included):

```
{
      "ram":
          {
              "type": "numberRange",
              "min": 1025,
              "max": 4096
          }
}
```

Date Range Filters

The following properties are stored as dates:

```
dateInstalled
dateLastSeen
```

The type field in the above filters needs to be `date` or `dateRange`.

- If `date` is specified, then a value field must also be present. The value field should contain a date.
- If `dateRange` is specified, then the value field should NOT be used. Instead, the properties min and max are to be used. These refer to the minimum and maximum dates to be included in the report. If only one limit needs to be set, the other property is to be left out.

In the following example, users installed after January 1st 2018 are to be shown:

```
{
    "dateInstalled":
        {
            "type": "dateRange",
            "min": "2018-01-01"
        }
}
```

Note • *All dates must be in ISO 8601 format.*

Boolean Filters

The following property is stored as boolean:

```
touchScreen
```

The type field in the above filters needs to be boolean. The value must be true or false. In the following filter, clients with a touch screen are being requested.

```
{
    "touchScreen":
        {
```

```
            "value": true
        }
}
```

Special Filters

Some filters need to be represented in a special format due to their unique requirements. These special filters are:

- Special Filter: licenseStatus
- Special Filter: os
- Special Filter: geography
- Special Filter: gpu
- Special Filters: optOut and backOff
- Special Filter: lifetimeEventUsage
- Special Filter: reachOutDeliveries

Special Filter: licenseStatus

The `licenseStatus` filter is made up of 4 sub-values: `activated`, `blacklisted`, `expired` and `whitelisted`. These are presented as boolean values.

Unlike other filters, this filter is presented as an array of JSON objects. Each object can contain a subset (or all) of these 4 boolean values.

Consider the following example. In this example, for a client to be included, the license has to either be activated AND whitelisted, or else it can be not whitelisted AND expired. In other words, ((activated AND whitelisted) OR ((NOT)whitelisted AND expired)).

```
{
    "licenseStatus":
        [
            {
                "activated": true,
                "whitelisted": true
            },
            {
                "whitelisted": false,
                "expired": true
            }
        ]
}
```

Special Filter: os

The os filter is made up of 3 granularity levels. These are `platform`, `version`, and `edition`. These are meant to split the OS name into levels of detail as required by the user. Consider the following:

- **platform**: Microsoft Windows

- **version**: Microsoft Windows 7
- **edition**: Microsoft Windows 7 Professional

If a filter is set on the version “Microsoft Windows 7”, the result would include all editions of Windows 7. One or more of these granularity levels may be specified. If more than 1 granularity level is specified, the values are ORed together.

In the following example, all editions of “Microsoft Windows 7” are included, and also “Microsoft Windows Vista Home Premium”:

```
{
    "type": "string",
    "version": "Microsoft Windows 7",
    "edition": "Microsoft Windows Vista Home Premium"
}
```

In the following example, the type is `stringArray`. Note that an array needs to be passed if the type is set as such, even if it is to contain only 1 element. In this case, the version can be either “Microsoft Windows 7” or “Microsoft Windows 8” (which are ORed together). Also, clients running on “Microsoft Windows XP Professional” are to be included.

```
{
    "type": "stringArray",
    "version": ["Microsoft Windows 7", "Microsoft Windows 8"],
    "edition": ["Microsoft Windows XP Professional"]
}
```

Special Filter: geography

The geography filter is made up of 3 granularity levels. These are continent, country, and usState. The usState value applies only to United States. Continents and countries are presented in 2-letter codes. Countries follow ISO standard 3166-1 alpha-2. US states are presented in ISO 3166-2:US format.

In the following example, the clients have to be either:

- In the continents Asia or Oceania
- In the country Germany
- In the US states New York, New Jersey, or Kansas

```
{
    "type": "stringArray",
    "continent": ["AS", "OC"],
    "country": ["DE"],
    "usState": ["US-NY", "US-NJ", "US-KS"]
}
```

Important • *In this filter, the type can be `string` or `stringArray`. Regular expressions are not supported in geography filters.*

Special Filter: gpu

The gpu filter is made up of 2 granularity levels. These are `vendor` and `model`. Both are represented as string values.

In the following example, the clients have to have a GPU:

- From the vendors NVIDIA or Intel
- With the model AMD Radeon HD 4600

```
{
    "type": "stringArray",
    "vendor": ["NVIDIA", "Intel"],
    "model": ["AMD Radeon HD 4600"]
}
```

Special Filters: optOut and backOff

The opt-out mechanism was introduced in SDK version 5.1.0. With this feature, vendors can have their application report to the Usage Intelligence servers that a user does not want to be tracked. Using this property, vendors can filter out installations that were opted-out.

Similarly, backoff filtering was introduced with version 5.0.0. Backoff is when a product account runs over-quota and the server starts rejecting data. Although filtering for backed-off installations was introduced with version 5, it was also backported to previous versions. However, when a new installation with an SDK prior to version 5 tries to register with the server and is rejected, it is not marked as being once backed-off when it is eventually accepted by the server. With version 5 onwards, the server flags an installation as being historically backed-off in such cases.

Both `backOff` and `optOut` filters are made up of 2 boolean sub-values: `historical` and `current`.

- The `historical` value refers to installations that were once backed-off or opted-out. These may include installations that are still currently backed-off or opted-out.
- The `current` value refers to the status during the last time that the client called the server. Therefore, if an installation was opted-out yesterday but got opted-in today, it will be marked as historically opted-out but not currently opted-out.

In the following example, for a client to be included, the `optOut` status has to either be `historical` AND not `current`, or else it can be not `historical` (i.e. users have to be currently opted-in but used to be opted-out at some point or never opted out).

```
{
    "optOut":
        [
            {
                "historical": true,
                "current": false
            },
            {
                "historical": false
            }
        ]
}
```

Special Filter: lifetimeEventUsage

Using lifetime event usage filters, clients can be filtered based on whether a particular event or set of events occurred or not within the client's lifetime. Alternatively, one can set a filter based on the number of times an event has occurred.

In the following example, clients that are included must have done the “File Operations - Open” event at least 5 times to be counted.

```
{
     "category": "File Operations",
     "name": "Open",
     "min": 5
}
```

In the following example, clients must have done either “File Operations - Open” or “File Operations - Save” for a combined total of between 10 to 50 times.

```
{
    "combiArray": [
                    {
                      "categoryType": "string",
                      "nameType": "string",
                      "category": "File Operations",
                      "name": "Open"
                    },
                    {
                      "categoryType": "string",
                      "nameType": "string",
                      "category": "File Operations",
                      "name": "Save",
                    }
                ],
    "min": 10,
    "max": 50
}
```

In the following example, clients must have done any event within the “File Operations” category for a combined total of not more than 100 times. This is done using a regular expression in the name.

```
{
    "combiArray":  [
                     {
                       "categoryType": "string",
                       "nameType": "regex",
                       "category": "File Operations",
                       "name": ".*"
                     }
                  ],
    "max": 100
}
```

Special Filter: reachOutDeliveries

Using ReachOut delivery filters, clients can be filtered based on whether a particular ReachOut message or a combination of ReachOut messages were delivered or not within the client’s lifetime.

The filter consists of a JSON array that includes one or more objects. Each object is a combination of delivered and undelivered campaigns, and the different combinations are ORed together. Therefore, it is possible to show users that either received ReachOut message 1 but not 2, or else received 3 but not 4 as in the following example:

In the following example, we are looking for clients who either received campaign 1 but not 2, OR received campaign 2 but not 3.

```
[
     {"auto": {"delivered": ["1"], "undelivered":["2"]}},
     {"auto": {"delivered": ["2"], "undelivered":["3"]}}
]
```

The above example contains only “auto” ReachOut campaigns. Manual campaigns can be specified using “manual” instead of “auto” as in the above example. Each object can contain a mix of “auto” and “manual” campaigns.

<NULL> Values in Global Filters

Most of the available properties can include null values. There are different reasons why a value would be null. When these are properties that are set by the application, the possible reasons why a value would be null are cases where the value has not been set by the application (such as `prodBuild` never being set), and cases where values are set to an empty string (“”) or to a string containing “<NULL>”.

One other reason is that although these values have been set, the SDK has not yet had time to sync with the servers to provide this new information. In cases where the properties are set automatically such as hardware or OS related information, the values would be null if the SDK failed to retrieve that value from the OS or if the server failed to identify the value retrieved by the SDK.

Other reasons include cases where Java version is requested from an application that does not use the Java SDK, US state is requested for users who are not running within the US, etc.

The following are the properties that support null values:

```
prodVersion
prodEdition
prodBuild
prodLanguage
machineId
formFactor
vm
cpuType
cpuCores
ram
resolutionWidth
resolutionHeight
javaVersion
javaVmVersion
javaVmName
javaVendor
javaRuntime
javaGraphics
osLanguage
licenseKey
C01 .. C20 (Custom properties)
os
geography
gpu
```

Null values can be requested either on their own or as part of a filter containing other values.

The following example would return only cases where the `prodVersion` is null:

```
{
      "prodVersion":
          {
              "includeNull": true
          }
}
```

The following example would return cases where the prodVersion is either 1.1, 1.2 or null:

```
{
      "prodVersion":
          {
              "type": "stringArray",
              "value": ["1.1", "1.2"],
              "includeNull": true
          }
}
```

By default, when specifying a filter, null values would not be included. Therefore, in the following example, only clients with prodVersion set to 1.1 or 1.2 will be included, while null values are excluded:

```
{
      "prodVersion":
          {
              "type": "stringArray",
              "value": ["1.1", "1.2"]
          }
}
```

However, if no filter is specified, then nulls are included by default. Therefore, if you want to include any value of prodVersion as long as it is not null, a prodVersion filter needs to be included as follows:

```
{
      "prodVersion":
          {
              "type": "regex",
              "value": ".*",
              "includeNull": false
          }
}
```

In the case of filters that use sub-properties (os, geography, and gpu), the includeNull filter is to be included in the sub-property and applies to that specific sub-property only. In order to be able to include the includeNull property, instead of providing the value as a string or an array of strings, the value of the sub-property must be a JSON object that contains a property named “value”, and another named “includeNull”. Each of these properties is optional, but at least one of them must be present.

In the case of geography, this has a very particular meaning. Requesting for null “country” value does not return all cases where the country could not be retrieved, but only cases where the continent could be retrieved but the country could not. Similarly, requesting null “usState” returns cases where the continent and country could be retrieved but the US state could not. This does not include clients that are not situated in the US. If you are interested in finding clients where we could not detect any geographical data, the includeNull filter needs to be applied in the continent sub-property.

In the following example, we are requesting cases where we know that the client is within the US but the state could not be identified:

```
{
      "geography":
```

```
            {
                "type": "string",
                "country": "US",
                "usState":
                    {
                        "includeNull": true
                    }
            }
}
```

In the following example, we are requesting cases where the GPU is either “NVIDIA”, “AMD” or null (unidentified):

```
{
      "gpu":
          {
              "type": "stringArray",
              "vendor":
                  {
                      "value": ["NVIDIA", "AMD"],
                      "includeNull": true
                  }
          }
}
```

Results Format

The results object consist of 3 properties each containing an object. These properties are `activeDays`, `sessions`, and `runtime`. Each one of these properties contains histogram data for the number of active days per client, number of sessions, and total runtime hours, all within the selected date range.

Example Request

```
POST /reporting/engagement/usageDistribution  HTTP/1.1
Host: api.revulytics.com
Content-Type: application/json
Accept: application/json

{
    "user": "testuser@test.com",
    "sessionId": "VSB8E2BzSC2eZSJm4QmTpA",
    "productId": 12345678901,
    "startDate": "2018-05-01",
    "stopDate": "2018-06-30"
}
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "status": "OK",
    "results": {
        "activeDays": {
            "1": {
```

```
        "users": 152,
        "sessions": {
            "\u2264 1": 35,
            "2": 34,
            "3": 43,
            "4 - 5": 16,
            "6 - 8": 16,
            "9 - 12": 8,
            "\u2265 13": 0
        },
        "runtime": {
            "\u2264 0:05": 0,
            "0:06 - 0:15": 4,
            "0:16 - 0:30": 148,
            "\u2265 0:31": 0
        }
    },
    "2": {
        "users": 180,
        "sessions": {
            "\u2264 1": 2,
            "2": 15,
            "3": 19,
            "4 - 5": 70,
            "6 - 8": 36,
            "9 - 12": 23,
            "13 - 20": 15,
            "\u2265 21": 0
        },
        "runtime": {
            "\u2264 0:15": 0,
            "0:16 - 0:30": 12,
            "0:31 - 0:45": 20,
            "0:46 - 1:00": 148,
            "\u2265 1:01": 0
        }
    },
    "3": {
        "users": 160,
        "sessions": {
            "\u2264 2": 0,
            "3": 4,
            "4 - 5": 31,
            "6 - 8": 58,
            "9 - 12": 22,
            "13 - 20": 39,
            "21 - 30": 6,
            "\u2265 31": 0
        },
        "runtime": {
            "\u2264 0:45": 0,
            "0:46 - 1:00": 15,
            "1:01 - 1:30": 145,
            "\u2265 1:31": 0
        }
    },
```

```
"4": {
    "users": 154,
    "sessions": {
        "\u2264 3": 0,
        "4 - 5": 4,
        "6 - 8": 44,
        "9 - 12": 56,
        "13 - 20": 28,
        "21 - 30": 21,
        "31 - 50": 1,
        "\u2265 51": 0
    },
    "runtime": {
        "\u2264 0:45": 0,
        "0:46 - 1:00": 5,
        "1:01 - 1:30": 32,
        "1:31 - 2:00": 117,
        "\u2265 2:01": 0
    }
},
"5": {
    "users": 152,
    "sessions": {
        "\u2264 5": 0,
        "6 - 8": 13,
        "9 - 12": 63,
        "13 - 20": 37,
        "21 - 30": 29,
        "31 - 50": 10,
        "\u2265 51": 0
    },
    "runtime": {
        "\u2264 1:00": 0,
        "1:01 - 1:30": 2,
        "1:31 - 2:00": 150,
        "\u2265 2:01": 0
    }
},
"6": {
    "users": 168,
    "sessions": {
        "\u2264 5": 0,
        "6 - 8": 4,
        "9 - 12": 52,
        "13 - 20": 42,
        "21 - 30": 44,
        "31 - 50": 26,
        "\u2265 51": 0
    },
    "runtime": {
        "\u2264 1:30": 0,
        "1:31 - 2:00": 9,
        "2:01 - 3:00": 159,
        "\u2265 3:01": 0
    }
},
```

```
"7 - 9": {
    "users": 419,
    "sessions": {
        "\u2264 8": 0,
        "9 - 12": 26,
        "13 - 20": 191,
        "21 - 30": 51,
        "31 - 50": 131,
        "51 - 75": 20,
        "\u2265 76": 0
    },
    "runtime": {
        "\u2264 1:30": 0,
        "1:31 - 2:00": 1,
        "2:01 - 3:00": 168,
        "3:01 - 4:00": 250,
        "\u2265 4:01": 0
    }
},
"10 - 14": {
    "users": 604,
    "sessions": {
        "\u2264 12": 0,
        "13 - 20": 65,
        "21 - 30": 216,
        "31 - 50": 127,
        "51 - 75": 171,
        "76 - 100": 25,
        "\u2265 101": 0
    },
    "runtime": {
        "\u2264 2:00": 0,
        "2:01 - 3:00": 2,
        "3:01 - 4:00": 150,
        "4:01 - 6:00": 452,
        "\u2265 6:01": 0
    }
},
"15 - 19": {
    "users": 464,
    "sessions": {
        "\u2264 20": 0,
        "21 - 30": 41,
        "31 - 50": 173,
        "51 - 75": 91,
        "76 - 100": 136,
        "101 - 150": 23,
        "\u2265 151": 0
    },
    "runtime": {
        "\u2264 4:00": 0,
        "4:01 - 6:00": 157,
        "6:01 - 9:00": 307,
        "\u2265 9:01": 0
    }
},
```

```
"20 - 24": {
    "users": 426,
    "sessions": {
        "\u2264 30": 0,
        "31 - 50": 118,
        "51 - 75": 60,
        "76 - 100": 90,
        "101 - 150": 158,
        "\u2265 151": 0
    },
    "runtime": {
        "\u2264 6:00": 0,
        "6:01 - 9:00": 336,
        "9:01 - 15:00": 90,
        "\u2265 15:01": 0
    }
},
"25 - 29": {
    "users": 384,
    "sessions": {
        "\u2264 30": 0,
        "31 - 50": 34,
        "51 - 75": 123,
        "76 - 100": 34,
        "101 - 150": 168,
        "151 - 250": 25,
        "\u2265 251": 0
    },
    "runtime": {
        "\u2264 6:00": 0,
        "6:01 - 9:00": 24,
        "9:01 - 15:00": 360,
        "\u2265 15:01": 0
    }
},
"30 - 44": {
    "users": 811,
    "sessions": {
        "\u2264 30": 0,
        "31 - 50": 1,
        "51 - 75": 152,
        "76 - 100": 120,
        "101 - 150": 165,
        "151 - 250": 368,
        "251 - 500": 5,
        "\u2265 501": 0
    },
    "runtime": {
        "\u2264 9:00": 0,
        "9:01 - 15:00": 581,
        "15:01 - 24:00": 230,
        "\u2265 24:01": 0
    }
},
"45 - 59": {
    "users": 388,
```

```
        "sessions": {
            "\u2264 75": 0,
            "76 - 100": 32,
            "101 - 150": 89,
            "151 - 250": 129,
            "251 - 500": 138,
            "\u2265 501": 0
        },
        "runtime": {
            "\u2264 9:00": 0,
            "9:01 - 15:00": 1,
            "15:01 - 24:00": 387,
            "\u2265 24:01": 0
        }
    },
    "60 - 89": {
        "users": 329,
        "sessions": {
            "\u2264 100": 0,
            "101 - 150": 55,
            "151 - 250": 80,
            "251 - 500": 193,
            "\u2265 501": 1
        },
        "runtime": {
            "\u2264 15:00": 0,
            "15:01 - 24:00": 70,
            "24:01 - 48:00": 259,
            "\u2265 48:01": 0
        }
    },
    "90 - 119": {
        "users": 179,
        "sessions": {
            "\u2264 150": 0,
            "151 - 250": 21,
            "251 - 500": 97,
            "\u2265 501": 61
        },
        "runtime": {
            "\u2264 24:00": 0,
            "24:01 - 48:00": 179,
            "\u2265 48:01": 0
        }
    },
    "120 - 179": {
        "users": 101,
        "sessions": {
            "\u2264 250": 0,
            "251 - 500": 18,
            "\u2265 501": 83
        },
        "runtime": {
            "\u2264 24:00": 0,
            "24:01 - 48:00": 12,
            "48:01 - 96:00": 89,
```

```
                "\u2265 96:01": 0
            }
        },
        "\u2265 180": {
            "users": 0,
            "sessions": {},
            "runtime": {}
        }
    },
    "sessions": {
        "\u2264 1": {
            "users": 37,
            "activeDays": {
                "1": 35,
                "2": 2,
                "\u2265 3": 0
            },
            "runtime": {
                "\u2264 0:05": 0,
                "0:06 - 0:15": 1,
                "0:16 - 0:30": 36,
                "\u2265 0:31": 0
            }
        },
        "2": {
            "users": 49,
            "activeDays": {
                "1": 34,
                "2": 15,
                "\u2265 3": 0
            },
            "runtime": {
                "\u2264 0:05": 0,
                "0:06 - 0:15": 2,
                "0:16 - 0:30": 33,
                "0:31 - 0:45": 5,
                "0:46 - 1:00": 9,
                "\u2265 1:01": 0
            }
        },
        "3": {
            "users": 66,
            "activeDays": {
                "1": 43,
                "2": 19,
                "3": 4,
                "\u2265 4": 0
            },
            "runtime": {
                "\u2264 0:05": 0,
                "0:06 - 0:15": 1,
                "0:16 - 0:30": 44,
                "0:31 - 0:45": 1,
                "0:46 - 1:00": 17,
                "1:01 - 1:30": 3,
                "\u2265 1:31": 0
```

```
        }
    },
    "4 - 5": {
        "users": 121,
        "activeDays": {
            "1": 16,
            "2": 70,
            "3": 31,
            "4": 4,
            "\u2265 5": 0
        },
        "runtime": {
            "\u2264 0:15": 0,
            "0:16 - 0:30": 19,
            "0:31 - 0:45": 8,
            "0:46 - 1:00": 61,
            "1:01 - 1:30": 30,
            "1:31 - 2:00": 3,
            "\u2265 2:01": 0
        }
    },
    "6 - 8": {
        "users": 171,
        "activeDays": {
            "1": 16,
            "2": 36,
            "3": 58,
            "4": 44,
            "5": 13,
            "6": 4,
            "\u2265 7": 0
        },
        "runtime": {
            "\u2264 0:15": 0,
            "0:16 - 0:30": 18,
            "0:31 - 0:45": 3,
            "0:46 - 1:00": 41,
            "1:01 - 1:30": 60,
            "1:31 - 2:00": 45,
            "2:01 - 3:00": 4,
            "\u2265 3:01": 0
        }
    },
    "9 - 12": {
        "users": 250,
        "activeDays": {
            "1": 8,
            "2": 23,
            "3": 22,
            "4": 56,
            "5": 63,
            "6": 52,
            "7 - 9": 26,
            "\u2265 10": 0
        },
        "runtime": {
```

```
            "\u2264 0:15": 0,
            "0:16 - 0:30": 10,
            "0:31 - 0:45": 3,
            "0:46 - 1:00": 22,
            "1:01 - 1:30": 35,
            "1:31 - 2:00": 105,
            "2:01 - 3:00": 73,
            "3:01 - 4:00": 2,
            "\u2265 4:01": 0
        }
    },
    "13 - 20": {
        "users": 417,
        "activeDays": {
            "1": 0,
            "2": 15,
            "3": 39,
            "4": 28,
            "5": 37,
            "6": 42,
            "7 - 9": 191,
            "10 - 14": 65,
            "\u2265 15": 0
        },
        "runtime": {
            "\u2264 0:45": 0,
            "0:46 - 1:00": 18,
            "1:01 - 1:30": 42,
            "1:31 - 2:00": 61,
            "2:01 - 3:00": 122,
            "3:01 - 4:00": 154,
            "4:01 - 6:00": 20,
            "\u2265 6:01": 0
        }
    },
    "21 - 30": {
        "users": 408,
        "activeDays": {
            "1 - 2": 0,
            "3": 6,
            "4": 21,
            "5": 29,
            "6": 44,
            "7 - 9": 51,
            "10 - 14": 216,
            "15 - 19": 41,
            "\u2265 20": 0
        },
        "runtime": {
            "\u2264 1:00": 0,
            "1:01 - 1:30": 9,
            "1:31 - 2:00": 51,
            "2:01 - 3:00": 60,
            "3:01 - 4:00": 74,
            "4:01 - 6:00": 203,
            "6:01 - 9:00": 11,
```

```
            "\u2265 9:01": 0
        }
    },
    "31 - 50": {
        "users": 621,
        "activeDays": {
            "1 - 3": 0,
            "4": 1,
            "5": 10,
            "6": 26,
            "7 - 9": 131,
            "10 - 14": 127,
            "15 - 19": 173,
            "20 - 24": 118,
            "25 - 29": 34,
            "30 - 44": 1,
            "\u2265 45": 0
        },
        "runtime": {
            "\u2264 1:30": 0,
            "1:31 - 2:00": 12,
            "2:01 - 3:00": 70,
            "3:01 - 4:00": 116,
            "4:01 - 6:00": 160,
            "6:01 - 9:00": 230,
            "9:01 - 15:00": 33,
            "\u2265 15:01": 0
        }
    },
    "51 - 75": {
        "users": 617,
        "activeDays": {
            "1 - 6": 0,
            "7 - 9": 20,
            "10 - 14": 171,
            "15 - 19": 91,
            "20 - 24": 60,
            "25 - 29": 123,
            "30 - 44": 152,
            "\u2265 45": 0
        },
        "runtime": {
            "\u2264 3:00": 0,
            "3:01 - 4:00": 54,
            "4:01 - 6:00": 175,
            "6:01 - 9:00": 105,
            "9:01 - 15:00": 278,
            "15:01 - 24:00": 5,
            "\u2265 24:01": 0
        }
    },
    "76 - 100": {
        "users": 437,
        "activeDays": {
            "1 - 9": 0,
            "10 - 14": 25,
```

```
            "15 - 19": 136,
            "20 - 24": 90,
            "25 - 29": 34,
            "30 - 44": 120,
            "45 - 59": 32,
            "\u2265 60": 0
        },
        "runtime": {
            "\u2264 4:00": 0,
            "4:01 - 6:00": 50,
            "6:01 - 9:00": 191,
            "9:01 - 15:00": 147,
            "15:01 - 24:00": 49,
            "\u2265 24:01": 0
        }
    },
    "101 - 150": {
        "users": 658,
        "activeDays": {
            "1 - 14": 0,
            "15 - 19": 23,
            "20 - 24": 158,
            "25 - 29": 168,
            "30 - 44": 165,
            "45 - 59": 89,
            "60 - 89": 55,
            "\u2265 90": 0
        },
        "runtime": {
            "\u2264 4:00": 0,
            "4:01 - 6:00": 1,
            "6:01 - 9:00": 130,
            "9:01 - 15:00": 319,
            "15:01 - 24:00": 191,
            "24:01 - 48:00": 17,
            "\u2265 48:01": 0
        }
    },
    "151 - 250": {
        "users": 623,
        "activeDays": {
            "1 - 24": 0,
            "25 - 29": 25,
            "30 - 44": 368,
            "45 - 59": 129,
            "60 - 89": 80,
            "90 - 119": 21,
            "\u2265 120": 0
        },
        "runtime": {
            "\u2264 9:00": 0,
            "9:01 - 15:00": 255,
            "15:01 - 24:00": 273,
            "24:01 - 48:00": 95,
            "\u2265 48:01": 0
        }
```

```
        },
        "251 - 500": {
            "users": 451,
            "activeDays": {
                "1 - 29": 0,
                "30 - 44": 5,
                "45 - 59": 138,
                "60 - 89": 193,
                "90 - 119": 97,
                "120 - 179": 18,
                "\u2265 180": 0
            },
            "runtime": {
                "\u2264 15:00": 0,
                "15:01 - 24:00": 169,
                "24:01 - 48:00": 265,
                "48:01 - 96:00": 17,
                "\u2265 96:01": 0
            }
        },
        "\u2265 501": {
            "users": 145,
            "activeDays": {
                "1 - 59": 0,
                "60 - 89": 1,
                "90 - 119": 61,
                "120 - 179": 83,
                "\u2265 180": 0
            },
            "runtime": {
                "\u2264 24:00": 0,
                "24:01 - 48:00": 73,
                "48:01 - 96:00": 72,
                "\u2265 96:01": 0
            }
        }
    },
    "runtime": {
        "\u2264 0:05": {
            "users": 0,
            "activeDays": {},
            "sessions": {}
        },
        "0:06 - 0:15": {
            "users": 4,
            "activeDays": {
                "1": 4,
                "\u2265 2": 0
            },
            "sessions": {
                "\u2264 1": 1,
                "2": 2,
                "3": 1,
                "\u2265 4": 0
            }
        },
```

```
"0:16 - 0:30": {
    "users": 160,
    "activeDays": {
        "1": 148,
        "2": 12,
        "\u2265 3": 0
    },
    "sessions": {
        "\u2264 1": 36,
        "2": 33,
        "3": 44,
        "4 - 5": 19,
        "6 - 8": 18,
        "9 - 12": 10,
        "\u2265 13": 0
    }
},
"0:31 - 0:45": {
    "users": 20,
    "activeDays": {
        "1": 0,
        "2": 20,
        "\u2265 3": 0
    },
    "sessions": {
        "\u2264 1": 0,
        "2": 5,
        "3": 1,
        "4 - 5": 8,
        "6 - 8": 3,
        "9 - 12": 3,
        "\u2265 13": 0
    }
},
"0:46 - 1:00": {
    "users": 168,
    "activeDays": {
        "1": 0,
        "2": 148,
        "3": 15,
        "4": 5,
        "\u2265 5": 0
    },
    "sessions": {
        "\u2264 1": 0,
        "2": 9,
        "3": 17,
        "4 - 5": 61,
        "6 - 8": 41,
        "9 - 12": 22,
        "13 - 20": 18,
        "\u2265 21": 0
    }
},
"1:01 - 1:30": {
    "users": 179,
```

```
        "activeDays": {
            "1 - 2": 0,
            "3": 145,
            "4": 32,
            "5": 2,
            "\u2265 6": 0
        },
        "sessions": {
            "\u2264 2": 0,
            "3": 3,
            "4 - 5": 30,
            "6 - 8": 60,
            "9 - 12": 35,
            "13 - 20": 42,
            "21 - 30": 9,
            "\u2265 31": 0
        }
    },
    "1:31 - 2:00": {
        "users": 277,
        "activeDays": {
            "1 - 3": 0,
            "4": 117,
            "5": 150,
            "6": 9,
            "7 - 9": 1,
            "\u2265 10": 0
        },
        "sessions": {
            "\u2264 3": 0,
            "4 - 5": 3,
            "6 - 8": 45,
            "9 - 12": 105,
            "13 - 20": 61,
            "21 - 30": 51,
            "31 - 50": 12,
            "\u2265 51": 0
        }
    },
    "2:01 - 3:00": {
        "users": 329,
        "activeDays": {
            "1 - 5": 0,
            "6": 159,
            "7 - 9": 168,
            "10 - 14": 2,
            "\u2265 15": 0
        },
        "sessions": {
            "\u2264 5": 0,
            "6 - 8": 4,
            "9 - 12": 73,
            "13 - 20": 122,
            "21 - 30": 60,
            "31 - 50": 70,
            "\u2265 51": 0
```

```
        }
    },
    "3:01 - 4:00": {
        "users": 400,
        "activeDays": {
            "1 - 6": 0,
            "7 - 9": 250,
            "10 - 14": 150,
            "\u2265 15": 0
        },
        "sessions": {
            "\u2264 8": 0,
            "9 - 12": 2,
            "13 - 20": 154,
            "21 - 30": 74,
            "31 - 50": 116,
            "51 - 75": 54,
            "\u2265 76": 0
        }
    },
    "4:01 - 6:00": {
        "users": 609,
        "activeDays": {
            "1 - 9": 0,
            "10 - 14": 452,
            "15 - 19": 157,
            "\u2265 20": 0
        },
        "sessions": {
            "\u2264 12": 0,
            "13 - 20": 20,
            "21 - 30": 203,
            "31 - 50": 160,
            "51 - 75": 175,
            "76 - 100": 50,
            "101 - 150": 1,
            "\u2265 151": 0
        }
    },
    "6:01 - 9:00": {
        "users": 667,
        "activeDays": {
            "1 - 14": 0,
            "15 - 19": 307,
            "20 - 24": 336,
            "25 - 29": 24,
            "\u2265 30": 0
        },
        "sessions": {
            "\u2264 20": 0,
            "21 - 30": 11,
            "31 - 50": 230,
            "51 - 75": 105,
            "76 - 100": 191,
            "101 - 150": 130,
            "\u2265 151": 0
```

```
        }
    },
    "9:01 - 15:00": {
        "users": 1032,
        "activeDays": {
            "1 - 19": 0,
            "20 - 24": 90,
            "25 - 29": 360,
            "30 - 44": 581,
            "45 - 59": 1,
            "\u2265 60": 0
        },
        "sessions": {
            "\u2264 30": 0,
            "31 - 50": 33,
            "51 - 75": 278,
            "76 - 100": 147,
            "101 - 150": 319,
            "151 - 250": 255,
            "\u2265 251": 0
        }
    },
    "15:01 - 24:00": {
        "users": 687,
        "activeDays": {
            "1 - 29": 0,
            "30 - 44": 230,
            "45 - 59": 387,
            "60 - 89": 70,
            "\u2265 90": 0
        },
        "sessions": {
            "\u2264 50": 0,
            "51 - 75": 5,
            "76 - 100": 49,
            "101 - 150": 191,
            "151 - 250": 273,
            "251 - 500": 169,
            "\u2265 501": 0
        }
    },
    "24:01 - 48:00": {
        "users": 450,
        "activeDays": {
            "1 - 59": 0,
            "60 - 89": 259,
            "90 - 119": 179,
            "120 - 179": 12,
            "\u2265 180": 0
        },
        "sessions": {
            "\u2264 100": 0,
            "101 - 150": 17,
            "151 - 250": 95,
            "251 - 500": 265,
            "\u2265 501": 73
```

```
                    }
                },
                "48:01 - 96:00": {
                    "users": 89,
                    "activeDays": {
                        "1 - 119": 0,
                        "120 - 179": 89,
                        "\u2265 180": 0
                    },
                    "sessions": {
                        "\u2264 250": 0,
                        "251 - 500": 17,
                        "\u2265 501": 72
                    }
                },
                "\u2265 96:01": {
                    "users": 0,
                    "activeDays": {},
                    "sessions": {}
                }
            }
        }
    }
```

10

Event Tracking Reports

These reports are meant to provide insight of what features in your product are most popular and how they are used. Due to the different needs of each individual software product, Usage Intelligence offers a range of different ways how to track events.

- Lifetime Event Tracking Reports
- Basic Event Tracking Reports
- Advanced Event Tracking Reports

Lifetime Event Tracking Reports

The aim of these reports is to show how events occur throughout the clients' lifetime. The data can be presented either as a paged table which shows a list of all events and how many times each occurred, or else as a histogram showing only a subset of events as specified. The histogram shows how many clients performed an event throughout their lifetime or their average daily/weekly/monthly usage.

- Data Table Report
- Histogram Report

Data Table Report

This report returns data that is to be represented in tabular format. It contains data about each tracked event, how many times it occurred, how many times each user performed each event on average, etc. The events can be presented either as a flat view or categorized hierarchically based on event category and name. Data for each event can then be segmented by any property as described below.

- Request/Response Parameters Summary
- Global Filters
- Segmentation
- Sorting

- Results Format

Request/Response Parameters Summary

This report returns data that is to be represented in tabular format.

`POST /reporting/eventTracking/lifetime/dataTable`

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 10-1 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. Required only for non-cookie authentication. • **sessionId (string)**—The `sessionId` obtained via POST /auth/login. Required only for non-cookie authentication. • **productId (integer)**—The product ID on which this request is being done. • **groupBy (string)**—Optional parameter to specify the property with which to group installations. By default, this value is considered to be `clientId`. Other possible options are `machineId`, `licenseKey` or any custom property of type 3. • **startDate (string)**—The first date of the date range on which to base the report. This is to be formatted as YYYY-MM-DD. • **stopDate (string)**—The last date of the date range during which users must be active in order to be included in the report. This is to be formatted as YYYY-MM-DD. • **globalFilters (object)**—JSON object containing the filters to be applied to the available data. Details about these filters can be found in the Global Filters section. • **segmentBy (string)**—he field with which to segment the data. Details about segmentation can be found in Segmentation. • **events (array)**—Optional parameter to specify which events to include. Array of objects specifying which events to include in the result. Supports both single events and event combinations. Details can be found in Events Property. • **segments (string)**—Used to specify how data is to be segmented. Must be used in conjunction with `segmentBy`. Details about segmentation can be found in Segmentation. • **categorizeEvents (boolean)**—Whether to return events hierarchically based on category/event name (true) or return a flattened list (false). • **sorting (object)**—Used to specify the values with which to sort and the direction. Details about this field can be found in Sorting. • **paging (object)**—Optional parameter used to specify how many events to show and the index of the event to start with (starting from 0). These sub-parameters are named `limit` and `startAt`. Therefore, if showing 10 events per page and requesting page 3, `limit` should be set to 10 and `startAt` should be set to 20.

Table 10-1 • Request Properties

Property	Description
Response JSON Object	• status (string)—Contains OK if successful or SYNTAX ERROR or AUTH ERROR. • reason (string)—Present only if status is not OK. Contains error message (reason). • **segmentBy (string)**—The same value that was passed as segmentBy in the request • **categorizeEvents (boolean)**—The same value that was passed as categorizeEvents in the request • **results (object)**—Contains the results as requested represented as a JSON object. The result format is described below in Results Format.

Global Filters

Most of the available filter properties are string-based. This means that when applying a filter, the requested field can be represented as a string, stringArray or regex. There are also some filters which are numeric. These filters should be represented as number or numberRange.

- String-Based Filters
- Numeric Filters
- Date Range Filters
- Boolean Filters
- Special Filters
- <NULL> Values for Global Filters

String-Based Filters

The following properties are stored as strings:

```
machineId *
clientId *
prodVersion
prodEdition
prodBuild
prodLanguage
licenseType
formFactor *
osLanguage
osWordLength *
cpuType *
dotNetVersion *
javaVersion *
javaVendor *
javaRuntime *
javaGraphics *
javaVmVersion *
javaVmName *
```

```
vm *
C01 .. C20 (Custom properties)
licenseKey *
```

Note • *licenseKey requires a special user permission to be used as a filter.*

Note • *Properties marked with an asterisk (*) are based on the current (latest known) values.*

The **type** field in the above filters needs to be `string`, `stringArray` or `regex`. A **value** field is always required. The contents of this field should be according to the specified type.

- If `string` is specified, then the value field must contain a single string that needs to be matched precisely with the stored data.
- If `stringArray` is specified, then the value field must contain an array of strings where one of which needs to match precisely with the stored data.
- If specifying a `regex`, the value field should contain a string which is treated as a regular expression and the stored data will be matched against it using regular expression rules.

Example Filter Using a String Value

In this example, the product build value needs to be exactly "`3014.int-12214`":

```
{
      "prodBuild":
          {
              "type": "string",
              "value": "3014.int-12214"
          }
}
```

Example Filter Using a String Array

In this example, the product build value needs to be either "`3014.int-12214`", "`3017.enx-57718`", or "`4180.vrx-81059`". Note that since the type is declared as `stringArray`, the value field needs to contain an array. Consider all elements in the array to have an OR logical expression between them.

```
{
    "prodBuild":
        {
            "type": "stringArray",
            "value": ["3014.int-12214", "3017.enx-57718", "4180.vrx-81059"]
        }
}
```

Example Filter Using a Regular Expression

In this example, the product build value needs to start with "`30`" and end with "`18`" whilst having 10 characters in between:

```
{
     "prodBuild":
         {
             "type": "regex",
```

```
            "value": "^30.{10}18$"
        }
}
```

Numeric Filters

The following properties are stored as numeric values:

```
cpuCores *
displayCount *
ram *
resolutionWidth *
resolutionHeight *
lifetimeRuntimeMinutes *
lifetimeSessionCount *
screenPpi *
javaVmRam *
```

Note • *Properties marked with an asterisk (*) are based on the current (latest known) values.*

The type field in the above filters needs to be `number` or `numberRange`.

- If number is specified, then a value field must also be present. The value field should contain a number, which may contain a decimal point if required.
- If `numberRange` is specified, then the value field should NOT be used. Instead, the properties min and max are to be used. These refer to the minimum and maximum number to be included in the report. If only one limit needs to be set, the other property is to be left out. Therefore, if you want to include installations with up to 2 display devices, you would not specify a min value, but instead specify only a max and set it as 2.

Example Filter Using a Number Value

In this example, the number of display devices needs to be exactly 3:

```
{
      "displayCount":
          {
              "type": "number",
              "value": 3
          }
}
```

Example Filter Using a Number Range Value

In this example, the RAM needs to be between 1025MB and 4096MB (both included):

```
{
      "ram":
          {
              "type": "numberRange",
              "min": 1025,
              "max": 4096
          }
}
```

Date Range Filters

The following properties are stored as dates:

`dateInstalled`
`dateLastSeen`

The type field in the above filters needs to be `date` or `dateRange`.

- If date is specified, then a `value` field must also be present. The `value` field should contain a date.
- If `dateRange` is specified, then the value field should NOT be used. Instead, the properties `min` and `max` are to be used. These refer to the minimum and maximum dates to be included in the report. If only one limit needs to be set, the other property is to be left out.

In the following example, users installed after January 1st 2018 are to be shown:

```
{
    "dateInstalled":
        {
            "type": "dateRange",
            "min": "2018-01-01"
        }
}
```

Note that all dates must be in ISO 8601 format.

Boolean Filters

The following property is stored as boolean:

`touchScreen`

The type field in the above filters needs to be boolean. The value must be true or false.

In the following filter, clients with a touch screen are being requested.

```
{
    "touchScreen":
        {
            "value": true
        }
}
```

Special Filters

Some filters need to be represented in a special format due to their unique requirements. These special filters are:

- Special Filter: licenseStatus
- Special Filter: os
- Special Filter: geography
- Special Filter: gpu
- Special Filters: optOut and backOff

Special Filter: licenseStatus

The `licenseStatus` filter is made up of 4 sub-values: `activated`, `blacklisted`, `expired` and `whitelisted`. These are presented as boolean values.

Unlike other filters, this filter is presented as an array of JSON objects. Each object can contain a subset (or all) of these 4 boolean values.

Consider the following example. In this example, for a client to be included, the license has to either be activated AND whitelisted, or else it can be not whitelisted AND expired. In other words, **(** (activated AND whitelisted) OR ((NOT)whitelisted AND expired) **)**.

```
{
    "licenseStatus":
        [
            {
                "activated": true,
                "whitelisted": true
            },
            {
                "whitelisted": false,
                "expired": true
            }
        ]
}
```

Special Filter: os

The os filter is made up of 3 granularity levels. These are `platform`, `version`, and `edition`. These are meant to split the OS name into levels of detail as required by the user. Consider the following:

- **platform**: Microsoft Windows
- **version**: Microsoft Windows 7
- **edition**: Microsoft Windows 7 Professional

If a filter is set on the version “Microsoft Windows 7”, the result would include all editions of Windows 7. One or more of these granularity levels may be specified. If more than 1 granularity level is specified, the values are ORed together.

In the following example, all editions of “Microsoft Windows 7” are included, and also “Microsoft Windows Vista Home Premium”:

```
{
    "type": "string",
    "version": "Microsoft Windows 7",
    "edition": "Microsoft Windows Vista Home Premium"
}
```

In the following example, the type is `stringArray`. Note that an array needs to be passed if the type is set as such, even if it is to contain only 1 element. In this case, the version can be either “Microsoft Windows 7” or “Microsoft Windows 8” (which are ORed together). Also, clients running on “Microsoft Windows XP Professional” are to be included.

```
{
```

```
    "type": "stringArray",
    "version": ["Microsoft Windows 7", "Microsoft Windows 8"],
    "edition": ["Microsoft Windows XP Professional"]
}
```

Special Filter: geography

The geography filter is made up of 3 granularity levels. These are `continent`, `country`, and `usState`.

The `usState` value applies only to United States. Continents and countries are presented in 2-letter codes. Countries follow ISO standard 3166-1 alpha-2. US states are presented in ISO 3166-2:US format.

In the following example, the clients have to be either:

- In the continents *`Asia`* or *`Oceania`*
- In the country *`Germany`*
- In the US states *`New York`*, *`New Jersey`*, or *`Kansas`*

```
{
    "type": "stringArray",
    "continent": ["AS", "OC"],
    "country": ["DE"],
    "usState": ["US-NY", "US-NJ", "US-KS"]
}
```

Important • *In this filter, the type can be string or `stringArray`. Regular expressions are not supported in geography filters.*

Special Filter: gpu

The gpu filter is made up of 2 granularity levels. These are `vendor` and `model`. Both are represented as string values.

In the following example, the clients have to have a GPU:

- From the vendors NVIDIA or Intel
- With the model AMD Radeon HD 4600

```
{
    "type": "stringArray",
    "vendor": ["NVIDIA", "Intel"],
    "model": ["AMD Radeon HD 4600"]
}
```

Special Filters: optOut and backOff

The opt-out mechanism was introduced in SDK version 5.1.0. With this feature, vendors can have their application report to the Usage Intelligence servers that a user does not want to be tracked. Using this property, vendors can filter out installations that were opted-out.

Similarly, backoff filtering was introduced with version 5.0.0. Backoff is when a product account runs over-quota and the server starts rejecting data. Although filtering for backed-off installations was introduced with version 5, it was also backported to previous versions. However, when a new installation with an SDK prior to version 5 tries to register with the server and is rejected, it is not marked as being once backed-off when it is eventually accepted by the server. With version 5 onwards, the server flags an installation as being historically backed-off in such cases.

Both backOff and optOut filters are made up of 2 boolean sub-values: historical and current.

- The historical value refers to installations that were once backed-off or opted-out. These may include installations that are still currently backed-off or opted-out.
- The current value refers to the status during the last time that the client called the server.

Therefore, if an installation was opted-out yesterday but got opted-in today, it will be marked as historically opted-out but not currently opted-out.

In the following example, for a client to be included, the optOut status has to either be historical AND not current, or else it can be not historical (i.e. users have to be currently opted-in but used to be opted-out at some point or never opted out).

```
{
    "optOut":
        [
            {
                "historical": true,
                "current": false
            },
            {
                "historical": false
            }
        ]
}
```

Special Filter: lifetimeEventUsage

Using lifetime event usage filters, clients can be filtered based on whether a particular event or set of events occurred or not within the client's lifetime. Alternatively, one can set a filter based on the number of times an event has occurred.

In the following example, clients that are included must have done the "File Operations - Open" event at least 5 times to be counted.

```
{
     "category": "File Operations",
     "name": "Open",
     "min": 5
}
```

In the following example, clients must have done either "File Operations - Open" or "File Operations - Save" for a combined total of between 10 to 50 times.

```
{
    "combiArray": [
                    {
                      "categoryType": "string",
                      "nameType": "string",
                      "category": "File Operations",
                      "name": "Open"
                    },
                    {
                      "categoryType": "string",
                      "nameType": "string",
                      "category": "File Operations",
                      "name": "Save",
                    }
```

```
                ],
    "min": 10,
    "max": 50
}
```

In the following example, clients must have done any event within the “File Operations” category for a combined total of not more than 100 times. This is done using a regular expression in the name.

```
{
    "combiArray":  [
                      {
                        "categoryType": "string",
                        "nameType": "regex",
                        "category": "File Operations",
                        "name": ".*"
                      }
                   ],
    "max": 100
}
```

Special Filter: reachOutDeliveries

Using ReachOut delivery filters, clients can be filtered based on whether a particular ReachOut message or a combination of ReachOut messages were delivered or not within the client’s lifetime.

The filter consists of a JSON array that includes one or more objects. Each object is a combination of delivered and undelivered campaigns, and the different combinations are ORed together. Therefore, it is possible to show users that either received ReachOut message 1 but not 2, or else received 3 but not 4 as in the following example.

In the following example, we are looking for clients who either received campaign 1 but not 2, OR received campaign 2 but not 3.

```
[
     {"auto": {"delivered": ["1"], "undelivered":["2"]}},
     {"auto": {"delivered": ["2"], "undelivered":["3"]}}
]
```

The above example contains only “auto” ReachOut campaigns. Manual campaigns can be specified using “manual” instead of “auto” as in the above example. Each object can contain a mix of “auto” and “manual” campaigns.

<NULL> Values for Global Filters

Most of the available properties can include null values. There are different reasons why a value would be null. When these are properties that are set by the application, the possible reasons why a value would be null are: cases where the value has not been set by the application (such as `prodBuild` never being set), and cases where values are set to an empty string (“”) or to a string containing “<NULL>”.

One other reason is that although these values have been set, the SDK has not yet had time to sync with the servers to provide this new information.

In cases where the properties are set automatically such as hardware or OS related information, the values would be null if the SDK failed to retrieve that value from the OS or if the server failed to identify the value retrieved by the SDK.

Other reasons include cases where Java version is requested from an application that does not use the Java SDK, US state is requested for users who are not running within the US, etc.

The following are the properties that support null values:

```
prodVersion
prodEdition
prodBuild
prodLanguage
machineId
formFactor
vm
cpuType
cpuCores
ram
resolutionWidth
resolutionHeight
javaVersion
javaVmVersion
javaVmName
javaVendor
javaRuntime
javaGraphics
osLanguage
licenseKey
C01 .. C20 (Custom properties)
os
geography
gpu
```

Null values can be requested either on their own or as part of a filter containing other values.

The following example would return only cases where the `prodVersion` is null:

```
{
      "prodVersion":
          {
              "includeNull": true
          }
}
```

The following example would return cases where the `prodVersion` is either 1.1, 1.2 or null:

```
{
      "prodVersion":
          {
              "type": "stringArray",
              "value": ["1.1", "1.2"],
              "includeNull": true
          }
}
```

By default, when specifying a filter, null values would not be included. Therefore, in the following example, only clients with `prodVersion` set to 1.1 or 1.2 will be included, while null values are excluded:

```
{
      "prodVersion":
          {
              "type": "stringArray",
              "value": ["1.1", "1.2"]
          }
}
```

However, if no filter is specified, then nulls are included by default. Therefore, if you want to include any value of `prodVersion` as long as it is not null, a `prodVersion` filter needs to be included as follows:

```
{
      "prodVersion":
          {
              "type": "regex",
              "value": ".*",
              "includeNull": false
          }
}
```

In the case of filters that use sub-properties (`os`, `geography`, and `gpu`), the `includeNull` filter is to be included in the sub-property and applies to that specific sub-property only. In order to be able to include the `includeNull` property, instead of providing the value as a string or an array of strings, the value of the sub-property must be a JSON object that contains a property named "`value`", and another named "`includeNull`". Each of these properties is optional, but at least one of them must be present.

In the case of geography, this has a very particular meaning. Requesting for null "country" value does not return all cases where the country could not be retrieved, but only cases where the continent could be retrieved but the country could not. Similarly, requesting null "`usState`" returns cases where the `continent` and `country` could be retrieved but the US state could not. This does not include clients that are not situated in the US. If you are interested in finding clients where we could not detect any geographical data, the `includeNull` filter needs to be applied in the continent sub-property.

In the following example, we are requesting cases where we know that the client is within the US but the state could not be identified:

```
{
      "geography":
          {
              "type": "string",
              "country": "US",
              "usState":
                  {
                      "includeNull": true
                  }
          }
}
```

In the following example, we are requesting cases where the GPU is either "NVIDIA", "AMD" or null (unidentified):

```
{
      "gpu":
          {
              "type": "stringArray",
              "vendor":
                  {
                      "value": ["NVIDIA", "AMD"],
                      "includeNull": true
                  }
          }
}
```

Segmentation

The data in this report is segmented based on the specified property. The property used for segmentation is to be specified in the `segmentBy` field. The segments field should specify how the report is to be segmented.

- String-Based Segmentation Properties
- Numeric Segmentation Properties
- Boolean Segmentation Properties
- Special Segmentation Properties
- <NULL> Values for Segmentation

String-Based Segmentation Properties

The following properties are stored as strings:

```
machineId *
clientId *
prodVersion
prodEdition
prodBuild
prodLanguage
licenseType
formFactor *
osLanguage
osWordLength *
cpuType *
javaVersion *
javaVendor *
javaRuntime *
javaGraphics *
javaVmVersion *
javaVmName *
vm *
C01 .. C20 (Custom properties)
licenseKey *
```

Note • *`licenseKey` requires a special user permission to be used for segmentation.*

Note • *Properties marked with an asterisk (*) are based on the current (latest known) values.*

The **type** field when using one of the above properties needs to be `string`, `stringArray` or `regex`. A **value** field is always required. The contents of this field should be according to the specified type.

- If `string` is specified, then the value field must contain a single string that needs to be matched precisely with the stored data.
- If `stringArray` is specified, then the value field must contain an array of strings where one of which needs to match precisely with the stored data.

- If specifying a regex, the value field should contain a string which is treated as a regular expression and the stored data will be matched against it using regular expression rules.

Example Using Segmentation by string, stringArray, and regex Values

```
{
    "segmentBy": "prodVersion",
    "segments": [
        {
            "type": "string",
            "value": "1.0"
        },
        {
            "type": "stringArray",
            "value": ["2.0", "2.1", "3.1"],
            "split": false,
            "segmentLabel": "Versions 2 and 3"
        },
        {
            "type": "regex",
            "value": "^4\..*",
            "split": false,
            "segmentLabel": "All version 4"
        },
        {
            "type": "regex",
            "value": "^5\..*",
            "split": true
        }
    ]
}
```

In the above example, we are requesting a report with multiple segments. The first segment contains installations running version 1.0. Notice how this does not require a “split” property since there is only 1 value and therefore no further splitting is possible. The second segment contains versions 2.0, 2.1 and 3.1. In this case, the “split” property is required, and since we are requesting the API to combine these 3 versions, we must provide a “segmentLabel” value so that the returned data can be identified. The third segment is similar, although in this case the request is built using a regular expression. In this case, all versions starting with “4.” are to be included into one combined segment.

The last segment is different from the rest because we are requesting the API to split the data (split is set to true). Therefore, this can produce much more than 1 segment. In this case, we could see segments such as “5.1”, “5.2”, etc. Notice how since we are splitting, we should not provide a `segmentLabel` value since the labels are built using the different values that are found in the data.

Numeric Segmentation Properties

The following properties are stored as numeric values:

```
cpuCores
displayCoun
ram
resolutionWidth
resolutionHeight
lifetimeRuntimeMinutes
lifetimeSessionCount
```

`screenPpi`
`javaVmRam`

The **type** field in the above properties needs to be number or `numberRange`. If number is specified, then a value field must also be present. The value field should contain a number, which may contain a decimal point if required. If `numberRange` is specified, then the value field should NOT be used. Instead, the properties min and max are to be used. These refer to the minimum and maximum number to be included in the report. If only one limit needs to be set, the other property is to be left out. Therefore, if you want to include installations with up to 2 display devices, you would not specify a min value, but instead specify only a max and set it as 2.

Example Using Segmentation by number, and numberRange Values

```
{
    "segmentBy": "cpuCores",
    "segments": [
        {
            "type": "number",
            "value": 1,
        },
        {
            "type": "numberRange",
            "min": 2,
            "max": 4,
            "segmentLabel": "2 - 4"
        },
        {
            "type": "numberRange",
            "min": 5,
            "segmentLabel": "5 +"
        }
    ]
}
```

In the above example, we are requesting a report with 3 segments. The first segment contains only installations running on 1 CPU core, the second segments contains installations running on 2, 3, or 4 cores (range 2 - 4), while the last segment contains all installations which are running on a machine with 5 or more CPU cores. Note how when the type was `numberRange`, we had do specify a `segmentLabel` which is a free string that will be used by the user to identify what is being included in that specific segment.

Boolean Segmentation Properties

The following properties are stored as boolean values:

`touchScreen`

The `type` field needs to be boolean, and the value must be true or false. A `segmentLabel` field is also required

The following example requests data segmented by `touchScreen`:

```
{
    "segmentBy": "touchScreen",
    "segments": [
        {
            "type": "boolean",
            "value": true,
            "segmentLabel": "Yes"
```

```
        },
        {
            "type": "boolean",
            "value": false,
            "segmentLabel": "No"
        },
        {
            "includeNull": true,
            "segmentLabel": "Unknown"
        }
    ]
}
```

In the above example, we are requesting a report with 3 segments. The first segment contains installations on which a touch screen was detected, the second one where no touch screen has been detected, while the last one is where we could not detect whether a touch screen is present due to the client using an old SDK which did not have touch screen detection support.

Special Segmentation Properties

Some properties need to be represented in a special format due to their unique requirements. These special properties are:

- Special Segmentation Format: licenseStatus
- Special Segmentation Format: os
- Special Segmentation Format: geography
- Special Segmentation Format: gpu
- Special Segmentation Format: optOut and backOff

Special Segmentation Format: licenseStatus

The `licenseStatus` value is made up of 4 sub-values: `activated`, `blacklisted`, `expired` and `whitelisted`. These are presented as boolean values. Any number of segments can be defined, and each segment can contain any subset of the 4 sub-values. These values are ANDed together. A `segmentLabel` value is required.

In the following example, 2 segments are specified - the first one showing blacklisted AND not expired and the second one showing whitelisted AND activated:

```
[
    {
        "segmentLabel": "BL and not EXP",
        "blacklisted": true,
        "expired": false
    },
    {
        "segmentLabel": "WL and ACT",
        "whitelisted": true,
        "expired": true
    }
]
```

Special Segmentation Format: os

The os value is made up of 3 granularity levels - `platform`, `version`, and `edition`. A particular level needs to be selected, and this is to be included in the property name such as `os.version` or `os.edition`. For a description of the differences between the 3 granularity levels, refer to Special Filter: os.

The following example requests data segmented by all OS versions:

```
{
    "segmentBy": "os.version",
    "segments": [
        {
            "type": "regex",
            "value": ".*",
            "split": true
        }
    ]
}
```

In the above example, no filtering is being done, and instead, a regular expression to include everything is set as the value. This will result in all OS versions to be returned.

Special Segmentation Format: geography

The geography value is made up of 3 granularity levels - `continent`, `country`, and `usState`. These granularity levels are explained in Special Filter: geography. A particular level needs to be selected, and this is to be included in the property name such as `geography.continent` or `geography.country`.

The following example requests data segmented by all countries:

```
{
    "segmentBy": "geography.country",
    "segments": [
        {
            "type": "regex",
            "value": ".*",
            "split": true
        }
    ]
```

}In the above example, no filtering is being done, and instead, a regular expression to include everything is set as the value. This will result in all countries to be returned.

Special Segmentation Format: gpu

The gpu value is made up of 2 granularity levels - `vendor` and `model`. These granularity levels are explained in Special Filter: gpu. A particular level needs to be selected, and this is to be included in the property name, namely `gpu.vendor` or `gpu.model`.

The following example requests data segmented by all GPU vendor:

```
{
    "segmentBy": "gpu.vendor",
    "segments": [
        {
            "type": "regex",
            "value": ".*",
```

```
            "split": true
        }
    ]
}
```

In the above example, no filtering is being done, and instead, a regular expression to include everything is set as the value. This will result in all the GPU vendors to be returned.

Special Segmentation Format: optOut and backOff

Both `backOff` and `optOut` values are made up of 2 boolean sub-values: `historical` and `current`. Any number of segments can be defined, and each segment can contain any subset of the 2 sub-values. These values are ANDed together. A `segmentLabel` value is required.

In the following example, 2 segments are specified - the first one showing historical AND not current and the second one showing not historical (i.e. never opted-out):

```
[
    {
        "segmentLabel": "HISTORICAL and not CURRENT",
        "historical": true,
        "current": false
    },
    {
        "segmentLabel": "Never opted-out",
        "historical": false
    }
]
```

<NULL> Values for Segmentation

Null values in segmentation are to be requested in a similar way to null values in filters (<NULL> Values for Global Filters). The same properties that support null in filtering also support null in segmentation.

By default, when segmenting, null values are not included within the segments, since only the values that have been specified in each segment are included. Null values don't match any regular expression, so the only way to request null values to be included is to specify "includeNull" as true in a similar way to filtering. In segmentation, null values are returned as "<NULL>". The API considers all cases where the data has never been set from the SDK, set as an empty string, or set as a string containing "<NULL>" to be the same.

The following example requests all values of `prodBuild` including null:

```
{
    "level1": {
        "property": "prodBuild",
        "segments": [
            {
                "type": "regex",
                "value": ".*",
                "includeNull": true
            }
        ]
    }
}
```

In the case of segmentation properties that use sub-properties (os, geography, and gpu), the includeNull value is to be included in the sub-property and applies to that specific sub-property only. In order to be able to include the includeNull property, instead of providing the value as a string or an array of strings, the value of the sub-property must be a JSON object that cointains a property named “value”, and another named “includeNull”. Each of these properties is optional, but at least one of them must be present. The same rules that apply for filtering these types of properties for null values also apply to segmentation.

In the following example, we are requesting segmentation by continent and are also requesting the number of clients where we could not detect the geographical location:

```
{
    "level1": {
        "property": "geography",
        "segments": [
            {
                "type": "regex",
                "continent": {
                                "value": ".*",
                                "includeNull": true
                            }
            }
        ]
    }
}
```

Sorting

The sorting parameter expects a JSON object which is made up of the following:

- **events (string)**—The value with which the event names are to be sorted. Possible values are `alpha` (alphabetical sorting), `eventCounts`, or `uniqueUsersUsedAtLeastOnce`.
- **eventsDirection (string)**—Whether to sort event names in ascending or descending order. Possible values are `asc` or `desc`.
- **segments (string)**—The value with which the segments are to be sorted. Possible values are `alpha` (alphabetical sorting), `active` (sorting by count of active users), `eventCounts`, `uniqueUsersUsedAtLeastOnce`, or `percentUsedAtLeastOnce`.
- **segmentsDirection (string)**—Whether to sort segments in ascending or descending order. Possible values are `asc` or `desc`.

Results Format

The structure of the results depends on whether the `categorizeEvents` property is set to true or false, and whether segmentation is being used.

The simplest form is when events are not categorized and segmentation is disabled. In this case, the results property contains an array of objects each containing details about each event (event category and name), how many times the event occurred in total, how many times each event occurred per user on average, etc. When setting `categorizeEvents` to `true`, the results are presented into 2 levels. The results are still an array of objects, but each object consists of only 2

properties - eventCategory and categoryData. The categoryData value is an array of objects, and each object contains the data for one event within that particular category. The only difference is that this sub-object does not contain the eventCategory property since that property is already present in the upper level.

Segmentation adds another level in the results. When using segmentation, each event object contains a property named segments. The segments value is an array of objects, with each object containing data about one segment. The segment name can be found in the property value named segmentLabel. The rest of the properties are common with the properties inside the event objects.

The following are the properties contained in the data objects:

Table 10-2 • Data Object Properties

Property	Description
eventCategory (string)	Present only in event objects when categorizeEvents is set to false. Contains the event category.
eventName (string)	Present in event objects. Contains the event name.
segmentLabel (string)	Present in segment objects. Contains the name/label of the segment.
eventCount (integer)	The total number of times that the event occurred.
eventCountPerUserUsedAtLeastOnce (float)	The average number of times that this event occurred for each client that performed this event at least once.
eventCountPerUserUsedAtLeastOnceActiveDay (float)	The average number of times that this event occurred per day for each client that performed this event at least once.
eventCountPerUserUsedAtLeastOnceActiveWeek (float)	The average number of times that this event occurred per week for each client that performed this event at least once.
eventCountPerUserUsedAtLeastOnceActiveMonth (float)	The average number of times that this event occurred per month for each client that performed this event at least once.
usersUsedAtLeastOnce (integer)	The number of clients which performed this event at least once.
percentUsedAtLeastOnce (float)	The percentage of clients which performed this event at least once.
usersNeverUsed (integer)	The number of clients which never performed this event.
percentNeverUsed (float)	The percentage of clients which never performed this event.
segments (object)	Present only in event objects when segmentation is being used. Contains an array of objects - one object for each segment.

For more information, see:

- Example Response with No Event Categorization and No Segmentation

- Example Response with Event Categorization and Segmentation by prodVersion

Example Response with No Event Categorization and No Segmentation

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "status": "OK",
    "segmentBy": null,
    "categorizeEvents": false,
    "results": [
                {
                    "eventCategory": "File Menu",
                    "eventName": "Open",
                    "eventCount": 5000,
                    "eventCountPerUserUsedAtLeastOnce": 20.0,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 2.62,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 9.84,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 29.96,
                    "usersUsedAtLeastOnce": 250,
                    "percentUsedAtLeastOnce": 50.0,
                    "usersNeverUsed": 250,
                    "percentNeverUsed": 50.0
                },
                {
                    "eventCategory": "File Menu",
                    "eventName": "Save",
                    "eventCount": 2000,
                    "eventCountPerUserUsedAtLeastOnce": 20.0,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 2.62,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 9.84,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 29.96,
                    "usersUsedAtLeastOnce": 100,
                    "percentUsedAtLeastOnce": 20.0,
                    "usersNeverUsed": 400,
                    "percentNeverUsed": 80.0
                },
                {
                    "eventCategory": "Edit Menu",
                    "eventName": "Enlarge",
                    "eventCountPerUser": 1.0,
                    "eventCountPerUserUsedAtLeastOnce": 0.3,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 2.62,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 9.84,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 29.96,
                    "usersUsedAtLeastOnce": 150,
                    "percentUsedAtLeastOnce": 30.0,
                    "usersNeverUsed": 350,
                    "percentNeverUsed": 70.0
                }
            ]
```

Example Response with Event Categorization and Segmentation by prodVersion


```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "status": "OK",
    "segmentBy": "prodVersion",
    "categorizeEvents": true,
    "results": [
                {
                    "eventCategory": "File Menu",
                    "categoryData":
                        [
                            {
                                "eventName": "Open",
                                "eventCount": 5000,
                                "eventCountPerUserUsedAtLeastOnce": 20.0,
                                "eventCountPerUserUsedAtLeastOnceActiveDay": 2.62,
                                "eventCountPerUserUsedAtLeastOnceActiveDay": 9.84,
                                "eventCountPerUserUsedAtLeastOnceActiveDay": 29.96,
                                "usersUsedAtLeastOnce": 250,
                                "percentUsedAtLeastOnce": 50.0,
                                "usersNeverUsed": 250,
                                "percentNeverUsed": 50.0
                                "segments":
                                    [
                                        {
                                            "segmentLabel": "V1.1",
                                            "eventCount": 3000,
                                            "eventCountPerUserUsedAtLeastOnce": 20.0,
                                            "eventCountPerUserUsedAtLeastOnceActiveDay": 2.62,
                                            "eventCountPerUserUsedAtLeastOnceActiveDay": 9.84,
                                            "eventCountPerUserUsedAtLeastOnceActiveDay": 29.96,
                                            "usersUsedAtLeastOnce": 150,
                                            "percentUsedAtLeastOnce": 33.33,
                                            "usersNeverUsed": 300,
                                            "percentNeverUsed": 66.66
                                        },
                                        {
                                            "segmentLabel": "V1.5",
                                            "eventCount": 2000,
                                            "eventCountPerUserUsedAtLeastOnce": 10.0,
                                            "eventCountPerUserUsedAtLeastOnceActiveDay": 1.85,
                                            "eventCountPerUserUsedAtLeastOnceActiveDay": 4.27,
                                            "eventCountPerUserUsedAtLeastOnceActiveDay": 14.92,
                                            "usersUsedAtLeastOnce": 100,
                                            "percentUsedAtLeastOnce": 33.33,
                                            "usersNeverUsed": 200,
                                            "percentNeverUsed": 66.66
                                        }
                                    ]
                            },
                            {
                                "eventName": "Save",
                                "eventCount": 2000,
```

```
                    "eventCountPerUserUsedAtLeastOnce": 20.0,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 2.62,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 9.84,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 29.96,
                    "usersUsedAtLeastOnce": 100,
                    "percentUsedAtLeastOnce": 20.0,
                    "usersNeverUsed": 400,
                    "percentNeverUsed": 80.0,
                    "segments":
                        [
                            {
                                "segmentLabel": "V1.1",
                                "eventCount": 1300,
                                "eventCountPerUserUsedAtLeastOnce": 13.0,
                                "eventCountPerUserUsedAtLeastOnceActiveDay": 4.98,
                                "eventCountPerUserUsedAtLeastOnceActiveDay": 11.31,
                                "eventCountPerUserUsedAtLeastOnceActiveDay": 24.18,
                                "usersUsedAtLeastOnce": 150,
                                "percentUsedAtLeastOnce": 40.0,
                                "usersNeverUsed": 100,
                                "percentNeverUsed": 60.0
                            },
                            {
                                "segmentLabel": "V1.5",
                                "eventCount": 700,
                                "eventCountPerUserUsedAtLeastOnce": 7.0,
                                "eventCountPerUserUsedAtLeastOnceActiveDay": 3.1,
                                "eventCountPerUserUsedAtLeastOnceActiveDay": 7.27,
                                "eventCountPerUserUsedAtLeastOnceActiveDay": 11.82,
                                "usersUsedAtLeastOnce": 100,
                                "percentUsedAtLeastOnce": 25.0,
                                "usersNeverUsed": 300,
                                "percentNeverUsed": 75.0
                            }
                        ]
                }
            ]
    },
    {
        "eventCategory": "Edit Menu",
        "categoryData":
            [
                {
                    "eventName": "Enlarge",
                    "eventCount": 500,
                    "eventCountPerUserUsedAtLeastOnce": 0.3,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 2.62,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 9.84,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 29.96,
                    "usersUsedAtLeastOnce": 150,
                    "percentUsedAtLeastOnce": 30.0,
                    "usersNeverUsed": 350,
                    "percentNeverUsed": 70.0,
                    "segments":
                        [
                            {
```

```
                "segmentLabel": "V1.1",
                "eventCount": 200,
                "eventCountPerUserUsedAtLeastOnce": 3.33,
                "eventCountPerUserUsedAtLeastOnceActiveDay": 4.32,
                "eventCountPerUserUsedAtLeastOnceActiveDay": 12.47,
                "eventCountPerUserUsedAtLeastOnceActiveDay": 25.51,
                "usersUsedAtLeastOnce": 60,
                "percentUsedAtLeastOnce": 28.57,
                "usersNeverUsed": 150,
                "percentNeverUsed": 71.43
            },
            {
                "segmentLabel": "V1.5",
                "eventCount": 300,
                "eventCountPerUserUsedAtLeastOnce": 3.33,
                "eventCountPerUserUsedAtLeastOnceActiveDay": 5.33,
                "eventCountPerUserUsedAtLeastOnceActiveDay": 6.42,
                "eventCountPerUserUsedAtLeastOnceActiveDay": 9.18,
                "usersUsedAtLeastOnce": 90,
                "percentUsedAtLeastOnce": 31.03,
                "usersNeverUsed": 200,
                "percentNeverUsed": 68.97
            }
        ]
    }
  ]
 }
]
```


Histogram Report

This report returns data that is to be represented in chart format. The result consists of histogram-style data for each different event or event combination that has been requested. The results consist of 4 histograms based on different values: event counts, average event count per day, average event count per week, and average event count per month.

- Request/Response Parameters Summary
- Events Property
- lowerBounds and binUpperBounds Properties
- Results Summary
- Results Histograms

Request/Response Parameters Summary

```
POST /reporting/eventTracking/lifetime/histogram
```

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 10-3 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. Required only for non-cookie authentication. • **sessionId (string)**—The sessionId obtained via POST /auth/login. Required only for non-cookie authentication. • **productId (integer)**—The product ID on which this request is being done. • **groupBy (string)**—Optional parameter to specify the property with which to group installations. By default, this value is considered to be `clientId`. Other possible options are `machineId`, `licenseKey` or any custom property of type 3. • **startDate (string)**—The first date of the date range during which users must be active in order to be included in the report. This is to be formatted as YYYY-MM-DD. • **stopDate (string)**—The last date of the date range during which users must be active in order to be included in the report. This is to be formatted as YYYY-MM-DD. • **globalFilters (object)**—JSON object containing the filters to be applied to the available data. These work exactly the same as in the data table report. Details can be found in Global Filters. • **events (array)**—Array of objects specifying which events to include in the result. Supports both single events and event combinations. Details can be found in Events Property. • **lowerBounds (object)**—Optional parameter to specify the lower bounds of each histogram. Must be used in conjunction with `binUpperBounds`. Details can be found in lowerBounds and binUpperBounds Properties. • **binUpperBounds (object)**—Optional parameter to specify the histogram bin bounds. Must be used in conjunction with `lowerBounds`. Details can be found in lowerBounds and binUpperBounds Properties.
Response JSON Object	• status (string)—Contains OK if successful or `SYNTAX ERROR` or `AUTH ERROR`. • reason (string)—Present only if status is not OK. Contains error message (reason). • **summary (object)**—Contains the number of clients who performed each event at least once and the number of clients which never performed each event. The summary data format is described in Results Histograms. • **histograms (object)**—Contains the histogram data as requested represented as a JSON object. The histogram data format is described in Results Histograms.

Events Property

The events property consists of an array of objects each representing a single event or a combination of events which are to be grouped together and represented as if they were a single event.

To specify a single event, the events array would contain an object similar to the following example:

```
{
     "category": "File Operations",
     "name":"Open"
}
```

To specify a combination of events, the events array would contain an object that looks like the following:

```
{
     "combiLabel": "Open and Save",
     "combiArray":  [
                       {
                         "categoryType": "string",
                         "nameType": "string",
                         "category": "File Operations",
                         "name": "Open"
                       },
                       {
                         "categoryType": "string",
                         "nameType": "string",
                         "category": "File Operations",
                         "name": "Save",
                       }
                    ]
}
```

In the above example, we are combining the data from the Open and Save events, both under the `File Operations` category. In this example, “string” is being used both as `categoryType` and `nameType`. The other possible value is “`regex`”.

The following example shows a case where all events under the `File Operations` category are being combined:

```
{
     "combiLabel": "All File Operations",
     "combiArray":  [
                       {
                         "categoryType": "string",
                         "nameType": "regex",
                         "category": "File Operations",
                         "name": ".*"
                       }
                    ]
}
```

In order to allow more advanced reporting, it is also possible to apply different filters for each event using a property named `eventFilters`. This way, it is possible to compare usage of a property or group of properties between different user groups. The format for `eventFilters` is exactly the same as `globalFilters`.

In the following example, we are comparing how the clients that are on version 1.1 have used the “Open” event vs. those clients that are on version 2.0:

```
[
    {
         "category": "File Operations",
         "name":"Open",
         "eventFilters":
             {
                 "prodVersion":
```

```
                    {
                        "type": "string",
                        "value": "1.1"
                    }
            }
    },
    {
         "category": "File Operations",
         "name":"Open",
         "eventFilters":
             {
                 "prodVersion":
                     {
                         "type": "string",
                         "value": "2.0"
                     }
             }
    }
]
```

It is also possible to apply different filters to different events within the same combiArray. Note that the use cases for this kind of filtering are rather limited, and in most cases, this method of filtering is not advised unless you have very specific requirements. In the following example, we are requesting the “Open” event for version 1.1 combined with the “Save” event for version 2.0:

```
{
     "combiLabel": "Open and Save in v1.1 and v2.0 respectively",
     "combiArray":  [
                       {
                         "categoryType": "string",
                         "nameType": "string",
                         "category": "File Operations",
                         "name": "Open",
                         "eventFilters":
                             {
                                 "prodVersion":
                                     {
                                         "type": "string",
                                         "value": "1.1"
                                     }
                             }
                       },
                       {
                         "categoryType": "string",
                         "nameType": "string",
                         "category": "File Operations",
                         "name": "Save",
                         "eventFilters":
                             {
                                 "prodVersion":
                                     {
                                         "type": "string",
                                         "value": "2.0"
                                     }
                             }
                       }
```

```
                    ]
}
```

lowerBounds and binUpperBounds Properties

By default, this report generates the histogram bins (ranges) automatically using a proprietary algorithm to reduce the effect of outliers. The `lowerBounds` and `binUpperBounds` are optional properties to allow the user to manually specify the bins instead of using this algorithm.

Both properties consist of a JSON object which contains 4 properties - `eventCount`, `averageCountPerActiveDay`, `averageCountPerActiveWeek`, and `averageCountPerActiveMonth`.

The `lowerBounds` object contains the lowest boundary in the leftmost bin in the histogram. The expected values inside the `lowerBounds` object are numeric. The `eventCount` value is an integer greater or equal to 1, while the other values are floating point numbers greater or equal to 0. Floating point numbers are expected to contain up to 2 decimal places, otherwise they are rounded.

The following is an example `lowerBounds` object:

```
{
    "eventCount": 1,
    "averageCountPerActiveDay": 0.1,
    "averageCountPerActiveWeek": 0.1,
    "averageCountPerActiveMonth": 0.1
}
```

The `binUpperBounds` property contains the upper boundaries of each bin in the histogram. The expected values inside the `binUpperBounds` object are arrays containing numeric values or a string containing “inf”. Each boundary must be higher than the one before it, and the first boundary must be greater than or equal to its corresponding value in `lowerBounds`. Similar to `lowerBounds`, the `eventCount` values are expected to be integers, while other values are expected to be floating point numbers with up to 2 decimal places.

The following is an example of the `binUpperBounds` object:

```
{
    "eventCount": [1, 2, 4, 8, 16, 32, "inf"],
    "averageCountPerActiveDay": [0.1, 0.2, 0.4, 0.6, 0.8, 1, 2, 4, 6, 10, 50, 100, "inf"],
    "averageCountPerActiveWeek": [0.1, 0.2, 0.4, 0.6, 0.8, 1, 2, 4, 6, 10, 50, 100, "inf"],
    "averageCountPerActiveMonth": [0.1, 0.2, 0.4, 0.6, 0.8, 1, 2, 4, 6, 10, 50, 100, "inf"]
}
```

Results Summary

The summary section is intended to show how many clients performed each event vs. the number of clients which never performed each event.

```
{
    "usedAtLeastOnce": [{
        "combiLabel": "Save or Delete",
        "value": 5
    }, {
        "category": "File Menu",
        "name": "Open",
        "value": 20
    }],
```

```
    "neverUsed": [{
        "combiLabel": "Save or Delete",
        "value": 37
    }, {
        "category": "File Menu",
        "name": "Open",
        "value": 112
    }]
}
```


Results Histograms

The histograms section contains the actual histogram results as requested.


```
{
    "eventCount":
      {
        "1 - 2":
          [
            {
                "combiLabel": "Save or Delete",
                "value": 10
            },
            {
                "category": "File Menu",
                "name": "Open",
                "value": 10
            }
          ],
        "3 - 5":
          [
            {
                "combiLabel": "Save or Delete",
                "value": 10
            },
            {
                "category": "File Menu",
                "name": "Open",
                "value": 10
            }
          ],
        "6 - 8":
          [
            {
                "combiLabel": "Save or Delete",
                "value": 1
            },
            {
                "category": "File Menu",
                "name": "Open",
                "value": 1
            }
          ],
        "9 - 11":
          [
```

```
        {
            "combiLabel": "Save or Delete",
            "value": 0
        },
        {
            "category": "File Menu",
            "name": "Open",
            "value": 0
        }
      ]
  },
"averageCountPerActiveDay":
  {
    "0.0 - 0.1":
      [
        {
            "combiLabel": "Save or Delete",
            "value": 0
        },
        {
            "category": "File Menu",
            "name": "Open",
            "value": 0
        }
      ],
    "0.11 - 0.2":
      [
        {
            "combiLabel": "Save or Delete",
            "value": 4
        },
        {
            "category": "File Menu",
            "name": "Open",
            "value": 4
        }
      ],
    "0.21 - 3.0":
      [
        {
            "combiLabel": "Save or Delete",
            "value": 20
        },
        {
            "category": "File Menu",
            "name": "Open",
            "value": 20
        }
      ],
    "3.01 - 4.0":
      [
        {
            "combiLabel": "Save or Delete",
            "value": 1
        },
        {
```

```
                "category": "File Menu",
                "name": "Open",
                "value": 1
            }
          ],
        "4.01 - 5.0":
          [
            {
                "combiLabel": "Save or Delete",
                "value": 0
            },
            {
                "category": "File Menu",
                "name": "Open",
                "value": 0
            }
          ]
      },
    "averageCountPerActiveWeek":
      {
        "0.5 - 1.0":
          [
            {
                "combiLabel": "Save or Delete",
                "value": 18
            },
            {
                "category": "File Menu",
                "name": "Open",
                "value": 18
            }
          ],
        "1.01 - 2.0":
          [
            {
                "combiLabel": "Save or Delete",
                "value": 2
            },
            {
                "category": "File Menu",
                "name": "Open",
                "value": 2
            }
          ],
        "2.01 - 3.0":
          [
            {
                "combiLabel": "Save or Delete",
                "value": 3
            },
            {
                "category": "File Menu",
                "name": "Open",
                "value": 3
            }
          ],
```

```
    "3.00 - 4.0":
      [
        {
            "combiLabel": "Save or Delete",
            "value": 2
        },
        {
            "category": "File Menu",
            "name": "Open",
            "value": 2
        }
      ],
    "4.01 - 5.0":
      [
        {
            "combiLabel": "Save or Delete",
            "value": 0
        },
        {
            "category": "File Menu",
            "name": "Open",
            "value": 0
        }
      ]
  },
"averageCountPerActiveMonth":
  {
    "0.7 - 1.0":
      [
        {
            "combiLabel": "Save or Delete",
            "value": 16
        },
        {
            "category": "File Menu",
            "name": "Open",
            "value": 16
        }
      ],
    "1.01 - 3.0":
      [
        {
            "combiLabel": "Save or Delete",
            "value": 3
        },
        {
            "category": "File Menu",
            "name": "Open",
            "value": 3
        }
      ],
    "3.01 - 5.0":
      [
        {
            "combiLabel": "Save or Delete",
            "value": 4
```

```
            },
            {
                "category": "File Menu",
                "name": "Open",
                "value": 4
            }
          ],
        "6.01 - 7.0":
          [
            {
                "combiLabel": "Save or Delete",
                "value": 1
            },
            {
                "category": "File Menu",
                "name": "Open",
                "value": 1
            }
          ]
      }
}
```


Basic Event Tracking Reports

Basic event tracking reports are presented in either a data table format which is meant to show a list of all known events within a date range and how they occurred, and also in a timeline format which is meant to show the daily usage of a smaller subset. Normally, the timeline chart is used to drill-down on a selection of events that can be seen on the data table.

- Data Table Report
- Timeline Report

Data Table Report

This report returns data that is to be represented in tabular format. It contains data about each tracked event, how many times it occurred, how many times each user performed each event on average, etc.

- Request/Response Parameters Summary
- Results Format

Request/Response Parameters Summary

```
POST /reporting/eventTracking/basic/dataTable
```

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 10-4 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. Required only for non-cookie authentication. • **sessionId (string)**—The `sessionId` obtained via POST /auth/login. Required only for non-cookie authentication. • **productId (integer)**—The product ID on which this request is being done/ • **startDate (string)**—The first date of the date range on which to base the report. This is to be formatted as YYYY-MM-DD. • **stopDate (string)**—The last date of the date range on which to base the report. This is to be formatted as YYYY-MM-DD.
Response JSON Object	• status (string)—Contains OK if successful or `SYNTAX ERROR` or `AUTH ERROR`. • reason (string)—Present only if status is not OK. Contains error message (reason). • **results (object)**—Contains the results as requested represented as a JSON object. The result format is described below.

Results Format

The results are formatted as an array which contains a JSON object for each category. Each of these objects has a property named “category” which refers to the category names of the events. If there are events that were not assigned a category, they can be found under a category named null.

Each of these objects contains 3 other keys - `categoryUsageSummary`, `categoryCustomValuesSummary`, and `categoryEventsData`. The first two contain objects that contain values regarding the totals for each category.

In the below example, the events in the File Operations category occurred for a total of 554 times. This is displayed in the `cumulativeUsageCount` property inside `categoryUsageSummary`.

The `categoryEventsData` object, contains data about each individual event type in that particular category. This object contains a number of sub-objects, each referring to a particular event name. The keys refer to the event names. Each of these sub-objects contain 2 keys - `eventUsage` and `eventCustomValues`. These contain the data values for individual events. For example, the event Menu Launched inside the Print Operations category occurred only 1 time, and the total numeric custom value collected by this event was 1.29.

Example Request

```
POST /reporting/eventTracking/basic/dataTable HTTP/1.1
Host: api.revulytics.com
Content-Type: application/json
Accept: application/json

{
    "user": "testuser@test.com",
```

```
    "sessionId": "VSB8E2BzSC2eZSJm4QmTpA",
    "productId": 12345678901,
    "startDate": "2018-08-01",
    "stopDate": "2018-08-03"
}
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "status": "OK",
    "results": [
        {
            "category": "File Operations",
            "categoryUsageSummary": {
                "cumulativeUsageCount": 554,
                "averageCountPerUser": 1.01,
                "averageCountPerSession": 0.21,
                "averageCountPerRuntimeHour": 0.01
            },
            "categoryCustomValuesSummary": {
                "cumulativeCustomValueCount": 2495.03,
                "averageCustomValuePerEvent": 4.66,
                "averageCustomValuePerUser": 4.71,
                "averageCustomValuePerSession": 0.99,
                "averageCustomValuePerRuntimeHour": 0.03
            },
            "categoryEventsData": {
                "Open": {
                    "eventUsage": {
                        "cumulativeUsageCount": 455,
                        "averageCountPerUser": 0.71,
                        "averageCountPerSession": 0.15,
                        "averageCountPerRuntimeHour": 0.24
                    },
                    "eventCustomValues": {
                        "cumulativeCustomValueCount": 2118.31,
                        "averageCustomValuePerEvent": 4.66,
                        "averageCustomValuePerUser": 3.3,
                        "averageCustomValuePerSession": 0.69,
                        "averageCustomValuePerRuntimeHour": 1.11
                    }
                },
                "Save": {
                    "eventUsage": {
                        "cumulativeUsageCount": 99,
                        "averageCountPerUser": 0.15,
                        "averageCountPerSession": 0.03,
                        "averageCountPerRuntimeHour": 0.05
                    },
                    "eventCustomValues": {
                        "cumulativeCustomValueCount": 376.72,
                        "averageCustomValuePerEvent": 3.81,
                        "averageCustomValuePerUser": 0.59,
```

```
                "averageCustomValuePerSession": 0.12,
                "averageCustomValuePerRuntimeHour": 0.2
            }
        }
    }
},
{
    "category": "Print Operations",
    "categoryUsageSummary": {
        "cumulativeUsageCount": 418,
        "averageCountPerUser": 0.65,
        "averageCountPerSession": 0.14,
        "averageCountPerRuntimeHour": 0
    },
    "categoryCustomValuesSummary": {
        "cumulativeCustomValueCount": 563.37,
        "averageCustomValuePerEvent": 1.35,
        "averageCustomValuePerUser": 0.88,
        "averageCustomValuePerSession": 0.18,
        "averageCustomValuePerRuntimeHour": 0
    },
    "categoryEventsData": {
        "Edit Mode": {
            "eventUsage": {
                "cumulativeUsageCount": 224,
                "averageCountPerUser": 0.35,
                "averageCountPerSession": 0.07,
                "averageCountPerRuntimeHour": 0.12
            },
            "eventCustomValues": {
                "cumulativeCustomValueCount": 149,
                "averageCustomValuePerEvent": 0.67,
                "averageCustomValuePerUser": 0.23,
                "averageCustomValuePerSession": 0.05,
                "averageCustomValuePerRuntimeHour": 0.08
            }
        },
        "Preview Mode": {
            "eventUsage": {
                "cumulativeUsageCount": 141,
                "averageCountPerUser": 0.22,
                "averageCountPerSession": 0.05,
                "averageCountPerRuntimeHour": 0.07
            },
            "eventCustomValues": {
                "cumulativeCustomValueCount": 274,
                "averageCustomValuePerEvent": 1.94,
                "averageCustomValuePerUser": 0.43,
                "averageCustomValuePerSession": 0.09,
                "averageCustomValuePerRuntimeHour": 0.14
            }
        },
        "Button Clicked": {
             "eventUsage": {
                "cumulativeUsageCount": 52,
                "averageCountPerUser": 0.08,
```

```
                    "averageCountPerSession": 0.02,
                    "averageCountPerRuntimeHour": 0.03
                },
                "eventCustomValues": {
                    "cumulativeCustomValueCount": 139.08,
                    "averageCustomValuePerEvent": 2.67,
                    "averageCustomValuePerUser": 0.22,
                    "averageCustomValuePerSession": 0.05,
                    "averageCustomValuePerRuntimeHour": 0.07
                }
            },
            "Menu Launched": {
                "eventUsage": {
                    "cumulativeUsageCount": 1,
                    "averageCountPerUser": 0,
                    "averageCountPerSession": 0,
                    "averageCountPerRuntimeHour": 0
                },
                "eventCustomValues": {
                    "cumulativeCustomValueCount": 1.29,
                    "averageCustomValuePerEvent": 1.29,
                    "averageCustomValuePerUser": 0,
                    "averageCustomValuePerSession": 0,
                    "averageCustomValuePerRuntimeHour": 0
                }
             }
        }
    }
]
}
```


Timeline Report

This report returns a daily/weekly/monthly timeline of how many times each of the requested events occurred. The event occurrence counts can optionally be divided by the number of users on each day/week/month, the number of user sessions, runtime hours, etc.

- Request/Response Parameters Summary
- Results Format

Request/Response Parameters Summary

```
POST /reporting/eventTracking/basic/timeline
```

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 10-5 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. Required only for non-cookie authentication. • **sessionId (string)**—The `sessionId` obtained via POST /auth/login. Required only for non-cookie authentication. • **productId (integer)**—The product ID on which this request is being done/ • **startDate (string)**—The first date of the date range on which to base the report. This is to be formatted as YYYY-MM-DD. • **stopDate (string)**—The last date of the date range on which to base the report. This is to be formatted as YYYY-MM-DD. • **dateSplit (string)**—Whether to present results by day, week, or month. • **dataView (string)**—Whether to show the counts of the number of times each event occurred (`usageCount`) or the total of the numeric custom values passed with each event (`customValues`). • **divisor (string)**—Optional parameter to divide the data by a related value. The possible values are `users`, `sessions`, `runtime`, `usageCounts`. Note that `usageCounts` cannot be selected if `dataView` is set to `usageCount`. To get the raw values with no divisors, you may leave the divisor property out or set its value to null. • **events (array)**—An array containing JSON objects, each containing a 2 string values - `category` and `name`.
Response JSON Object	• status (string)—Contains OK if successful or `SYNTAX ERROR` or `AUTH ERROR`. • reason (string)—Present only if status is not OK. Contains error message (reason). • **results (object)**—Contains the results as requested represented as a JSON object. The result format is described below.

Results Format

The results object contains a number of members having a date as the key and an object as the value. The date is formatted as YYYY-MM-DD regardless of whether the report is being split by day, week, or month. If the report is being split by week or by month, the first date of the week or month is used.

Each sub-object contains a sub-object for each event. These contain the event category and name, and the number of times it occurred, or the custom numeric value that was sent with it, depending on whether the `dataView` value is set as `usageCount` or `customValues`.

Example Request

```
POST /reporting/eventTracking/basic/timeline  HTTP/1.1
Host: api.revulytics.com
```

```
Content-Type: application/json
Accept: application/json

{
    "user": "testuser@test.com",
    "sessionId": "VSB8E2BzSC2eZSJm4QmTpA",
    "productId": 12345678901,
    "startDate": "2018-08-01",
    "stopDate": "2018-08-03",
    "dateSplit": "day",
    "dataView": "usageCounts",
    "divisor": null,
    "events": [
                {
                    "category": "File Operations",
                    "name": "Open"
                },
                {
                    "category": "File Operations",
                    "name": "Save"
                }

            ]
}
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "status": "OK",
    "results": {
        "2018-08-01": [
            {
                "category": "File Operations",
                "name": "Open",
                "value": 295
            },
            {
                "category": "File Operations",
                "name": "Save",
                "value": 59
            }
        ],
        "2018-08-02": [
            {
                "category": "File Operations",
                "name": "Open",
                "value": 132
            },
            {
                "category": "File Operations",
                "name": "Save",
                "value": 24
            }
```

```
        ],
        "2018-08-03": [
            {
                "category": "File Operations",
                "name": "Open",
                "value": 14
            },
            {
                "category": "File Operations",
                "name": "Save",
                "value": 2
            }
        ]
    }
}
```


Advanced Event Tracking Reports

These reports are meant to provide more insight about how events are used by your users. For events to be available in these reports, they must be selected as advanced events in the events management page.

- Event Usage Frequency Report

Event Usage Frequency Report

This report provides a detailed view into how users are using your tracked events. It is available only for events tracked as advanced. This report contains data about how many users used each event, many times each event occurred in total, how many times each user performed each event on average, etc. It also returns a histogram for each event showing the distribution of how many times each user performed each event.

- Request/Response Parameters Summary
- Global Filters
- Data Segmentation
- Events Property
- Results Format

Request/Response Parameters Summary

```
POST /reporting/eventTracking/advanced/fullReport
```

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 10-6 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. Required only for non-cookie authentication. • **sessionId (string)**—The sessionId obtained via POST /auth/login. Required only for non-cookie authentication. • **productId (integer)**—The product ID on which this request is being done. • **startDate (string)**—The first date of the date range on which to base the report. This is to be formatted as YYYY-MM-DD. • **stopDate (string)**—The last date of the date range on which to base the report. This is to be formatted as YYYY-MM-DD. • **globalFilters (object)**—JSON object containing the filters to be applied to the available data. Details about these filters can be found in Global Filters. • **segmentBy (string)**—The field with which to segment the data. Details about segmentation can be found in the Data Segmentation. • **segments (array)**— Optional JSON array that describes how data is to be split into segments. Refer to Data Segmentation. • **sort (string)**—Optional property to specify how the segments are to be sorted. Possible values are alpha or `eventCounts`. If this property is not included, the data is sorted alphabetically by default. • **sortDirection (string)**—Optional property to specify whether to sort in ascending or descending order. Possible values are `asc` and `desc`. If not specified, data is sorted in ascending order by default. • **events (array)**—Array of objects specifying which events or combination of events to base the report on. Please refer to Events Property.
Response JSON Object	• status (string)—Contains OK if successful or `SYNTAX ERROR` or `AUTH ERROR`. • reason (string)—Present only if status is not OK. Contains error message (reason). • **results (array)**—Present only if status is OK. Contains the list of available properties as described below.

Global Filters

Most of the available filter properties are string-based. This means that when applying a filter, the requested field can be represented as a string, `stringArray` or `regex`. There are also some filters which are numeric. These filters should be represented as `number` or `numberRange`.

- String-Based Filters

- Numeric Filters
- Date Range Filters
- Boolean Filters
- Special Filters
- <NULL> Values in Global Filters

String-Based Filters

The following properties are stored as strings:

```
machineId *
clientId *
prodVersion
prodEdition
prodBuild
prodLanguage
licenseType
formFactor *
osLanguage
osWordLength *
cpuType *
dotNetVersion *
javaVersion *
javaVendor *
javaRuntime *
javaGraphics *
javaVmVersion *
javaVmName *
vm *
C01 .. C20 (Custom properties)
licenseKey *
```

Important • *licenseKey requires a special user permission to filter by license key.*

Note • *Properties marked with an asterisk (*) are based on the current (latest known) values.*

The `type` field in the above filters needs to be `string`, `stringArray` or `regex`. A value field is always required. The contents of this field should be according to the specified type.

- If string is specified, then the value field must contain a single string that needs to be matched precisely with the stored data.
- If `stringArray` is specified, then the value field must contain an array of strings where one of which needs to match precisely with the stored data.
- If specifying a `regex`, the value field should contain a string which is treated as a regular expression and the stored data will be matched against it using regular expression rules.

Example Filter Using a String Value

In this example, the product build value needs to be exactly “3014.int-12214”:

```
{
      "prodBuild":
          {
              "type": "string",
              "value": "3014.int-12214"
          }
 }
```

Example Filter Using a String Array

In this example, the product build value needs to be either “3014.int-12214”, “3017.enx-57718”, or “4180.vrx-81059”. Note that since the type is declared as stringArray, the value field needs to contain an array. Consider all elements in the array to have an OR logical expression between them.:

```
{
    "prodBuild":
        {
            "type": "stringArray",
            "value": ["3014.int-12214", "3017.enx-57718", "4180.vrx-81059"]
        }
}
```

Example Filter Using a Regular Expression

In this example, the product build value needs to start with “30” and end with “18” whilst having 10 characters in between:

```
{
     "prodBuild":
         {
             "type": "regex",
             "value": "^30.{10}18$"
         }
}
```

Numeric Filters

The following properties are stored as numeric values:

```
cpuCores *
displayCount *
ram *
resolutionWidth *
resolutionHeight *
lifetimeRuntimeMinutes *
lifetimeSessionCount *
screenPpi *
javaVmRam *
```

Note • *Properties marked with an asterisk (*) are based on the current (latest known) values.*

The type field in the above filters needs to be `number` or `numberRange`.

- If `number` is specified, then a value field must also be present. The value field should contain a number, which may contain a decimal point if required.
- If `numberRange` is specified, then the value field should NOT be used. Instead, the properties `min` and `max` are to be used. These refer to the minimum and maximum number to be included in the report. If only one limit needs to be set, the other property is to be left out. Therefore, if you want to include installations with up to 2 display devices, you would not specify a min value, but instead specify only a max and set it as 2.

Example Filter Using a Number Value

In this example, the number of display devices needs to be exactly 3:

```
{
      "displayCount":
          {
              "type": "number",
              "value": 3
          }
 }
```

Example Filter Using a Number Value

In this example, the RAM needs to be between 1025MB and 4096MB (both included):

```
{
      "ram":
          {
              "type": "numberRange",
              "min": 1024,
              "max": 4096
          }
 }
```

Date Range Filters

The following properties are stored as dates:

```
dateInstalled
dateLastSeen
```

The type field in the above filters needs to be `date` or `dateRange`.

- If `date` is specified, then a value field must also be present. The value field should contain a date.
- If `dateRange` is specified, then the value field should NOT be used. Instead, the properties min and max are to be used. These refer to the minimum and maximum dates to be included in the report. If only one limit needs to be set, the other property is to be left out. In the following example, users installed after January 1st 2018 are to be shown:

```
{
    "dateInstalled":
        {
            "min": "2018-01-01"
        }
}
```

Note that all dates must be in ISO 8601 format.

Boolean Filters

The following property is stored as boolean:

`touchScreen`

The type field in the above filters needs to be boolean. The value must be true or false. In the following filter, clients with a touch screen are being requested.

```
{
    "touchScreen":
        {
            "value": true
        }
}
```

Special Filters

Some filters need to be represented in a special format due to their unique requirements. These special filters are:

- Special Filter: licenseStatus
- Special Filter: os
- Special Filter: geography
- Special Filter: gpu
- Special filters: optOut and backOff
- Special Filter: lifetimeEventUsage
- Special Filter: reachOutDeliveries

Special Filter: licenseStatus

The `licenseStatus` filter is made up of 4 sub-values: `activated`, `blacklisted`, `expired` and `whitelisted`. These are presented as boolean values. Unlike other filters, this filter is presented as an array of JSON objects. Each object can contain a subset (or all) of these 4 boolean values.

Consider the following example. In this example, for a client to be included, the license has to either be activated AND whitelisted, or else it can be not whitelisted AND expired. In other words, **(** (activated AND whitelisted) OR ((NOT)whitelisted AND expired) **)**.

```
{
    "licenseStatus":
        [
            {
                "activated": true,
                "whitelisted": true
            },
            {
                "whitelisted": false,
                "expired": true
            }
        ]
}
```

Special Filter: os

The os filter is made up of 3 granularity levels. These are `platform`, `version`, and `edition`. These are meant to split the OS name into levels of detail as required by the user. Consider the following:

- **platform**: Microsoft Windows
- **version**: Microsoft Windows 7
- **edition**: Microsoft Windows 7 Professional

If a filter is set on the version “Microsoft Windows 7”, the result would include all editions of Windows 7. One or more of these granularity levels may be specified. If more than 1 granularity level is specified, the values are ORed together.

In the following example, all editions of “Microsoft Windows 7” are included, and also “Microsoft Windows Vista Home Premium”:

```
{
    "type": "string",
    "version": "Microsoft Windows 7",
    "edition": "Microsoft Windows Vista Home Premium"
}
```

In the following example, the type is `stringArray`. Note that an array needs to be passed if the type is set as such, even if it is to contain only 1 element. In this case, the version can be either “Microsoft Windows 7” or “Microsoft Windows 8” (which are ORed together). Also, clients running on “Microsoft Windows XP Professional” are to be included.

```
{
    "type": "stringArray",
    "version": ["Microsoft Windows 7", "Microsoft Windows 8"],
    "edition": ["Microsoft Windows XP Professional"]
}
```

Special Filter: geography

The geography filter is made up of 3 granularity levels. These are `continent`, `country`, and `usState`.

The `usState` value applies only to United States. Continents and countries are presented in 2-letter codes. Countries follow ISO standard 3166-1 alpha-2. US states are presented in ISO 3166-2:US format.

In the following example, the clients have to be either:

- In the continents *Asia* or *Oceania*
- In the country *Germany*
- In the US states *New York*, *New Jersey*, or *Kansas*

```
{
    "type": "stringArray",
    "continent": ["AS", "OC"],
    "country": ["DE"],
    "usState": ["US-NY", "US-NJ", "US-KS"]
}
```

Important • *In this filter, the type can be string or `stringArray`. Regular expressions are not supported in geography filters.*

Special Filter: gpu

The gpu filter is made up of 2 granularity levels. These are `vendor` and `model`. Both are represented as string values.

In the following example, the clients have to have a GPU:

- From the vendors *NVIDIA* or *Intel*
- With the model *AMD Radeon HD 4600*

```
{
    "type": "stringArray",
    "vendor": ["NVIDIA", "Intel"],
    "model": ["AMD Radeon HD 4600"]
}
```

Special filters: optOut and backOff

The opt-out mechanism was introduced in SDK version 5.1.0. With this feature, vendors can have their application report to the Usage Intelligence servers that a user does not want to be tracked. Using this property, vendors can filter out installations that were opted-out.

Similarly, backoff filtering was introduced with version 5.0.0. Backoff is when a product account runs over-quota and the server starts rejecting data. Although filtering for backed-off installations was introduced with version 5, it was also backported to previous versions. However, when a new installation with an SDK prior to version 5 tries to register with the server and is rejected, it is not marked as being once backed-off when it is eventually accepted by the server. With version 5 onwards, the server flags an installation as being historically backed-off in such cases.

Both `backOff` and **optOut** filters are made up of 2 boolean sub-values: `historical` and `current`.

- The `historical` value refers to installations that were once backed-off or opted-out. These may include installations that are still currently backed-off or opted-out.
- The `current` value refers to the status during the last time that the client called the server. Therefore, if an installation was opted-out yesterday but got opted-in today, it will be marked as historically opted-out but not currently opted-out.

In the following example, for a client to be included, the `optOut` status has to either be `historical` AND not `current`, or else it can be not `historical` (i.e. users have to be currently opted-in but used to be opted-out at some point or never opted out).

```
{
    "optOut":
        [
            {
                "historical": true,
                "current": false
            },
            {
                "historical": false
            }
        ]
}
```

Special Filter: lifetimeEventUsage

Using lifetime event usage filters, clients can be filtered based on whether a particular event or set of events occurred or not within the client's lifetime. Alternatively, one can set a filter based on the number of times an event has occurred.

In the following example, clients that are included must have done the "File Operations - Open" event at least 5 times to be counted.

```
{
     "category": "File Operations",
     "name": "Open",
     "min": 5
}
```

In the following example, clients must have done either "File Operations - Open" or "File Operations - Save" for a combined total of between 10 to 50 times.

```
{
    "combiArray": [
                    {
                      "categoryType": "string",
                      "nameType": "string",
                      "category": "File Operations",
                      "name": "Open"
                    },
                    {
                      "categoryType": "string",
                      "nameType": "string",
                      "category": "File Operations",
                      "name": "Save",
                    }
                ],
    "min": 10,
    "max": 50
}
```

In the following example, clients must have done any event within the "File Operations" category for a combined total of not more than 100 times. This is done using a regular expression in the name.

```
{
    "combiArray":  [
                     {
                       "categoryType": "string",
                       "nameType": "regex",
                       "category": "File Operations",
                       "name": ".*"
                     }
                  ],
    "max": 100
}
```

Special Filter: reachOutDeliveries

Using ReachOut delivery filters, clients can be filtered based on whether a particular ReachOut message or a combination of ReachOut messages were delivered or not within the client's lifetime.

The filter consists of a JSON array that includes one or more objects. Each object is a combination of delivered and undelivered campaigns, and the different combinations are ORed together. Therefore, it is possible to show users that either received ReachOut message 1 but not 2, or else received 3 but not 4 as in the following example:

In the following example, we are looking for clients who either received campaign 1 but not 2, OR received campaign 2 but not 3.

```
[
     {"auto": {"delivered": ["1"], "undelivered":["2"]}},
     {"auto": {"delivered": ["2"], "undelivered":["3"]}}
]
```

The above example contains only “auto” ReachOut campaigns. Manual campaigns can be specified using “manual” instead of “auto” as in the above example. Each object can contain a mix of “auto” and “manual” campaigns.

<NULL> Values in Global Filters

Most of the available properties can include null values. There are different reasons why a value would be null. When these are properties that are set by the application, the possible reasons why a value would be null are cases where the value has not been set by the application (such as `prodBuild` never being set), and cases where values are set to an empty string (“”) or to a string containing “<NULL>”.

One other reason is that although these values have been set, the SDK has not yet had time to sync with the servers to provide this new information. In cases where the properties are set automatically such as hardware or OS related information, the values would be null if the SDK failed to retrieve that value from the OS or if the server failed to identify the value retrieved by the SDK. Other reasons include cases where Java version is requested from an application that does not use the Java SDK, US state is requested for users who are not running within the US, etc.

The following are the properties that support null values:

```
prodVersion
prodEdition
prodBuild
prodLanguage
machineId
formFactor
vm
cpuType
cpuCores
ram
resolutionWidth
resolutionHeight
javaVersion
javaVmVersion
javaVmName
javaVendor
javaRuntime
javaGraphics
osLanguage
licenseKey
C01 .. C20 (Custom properties)
os
geography
gpu
```

Null values can be requested either on their own or as part of a filter containing other values.

The following example would return only cases where the prodVersion is null:

```
{
      "prodVersion":
          {
              "includeNull": true
          }
}
```

The following example would return cases where the prodVersion is either 1.1, 1.2 or null:

```
{
      "prodVersion":
          {
              "type": "stringArray",
              "value": ["1.1", "1.2"],
              "includeNull": true
          }
}
```

By default, when specifying a filter, null values would not be included. Therefore, in the following example, only clients with prodVersion set to 1.1 or 1.2 will be included, while null values are excluded:

```
{
      "prodVersion":
          {
              "type": "stringArray",
              "value": ["1.1", "1.2"]
          }
}
```

However, if no filter is specified, then nulls are included by default. Therefore, if you want to include any value of prodVersion as long as it is not null, a prodVersion filter needs to be included as follows:

```
{
      "prodVersion":
          {
              "type": "regex",
              "value": ".*",
              "includeNull": false
          }
}
```

In the case of filters that use sub-properties (os, geography, and gpu), the includeNull filter is to be included in the sub-property and applies to that specific sub-property only. In order to be able to include the includeNull property, instead of providing the value as a string or an array of strings, the value of the sub-property must be a JSON object that contains a property named “value”, and another named “includeNull”. Each of these properties is optional, but at least one of them must be present.

In the case of geography, this has a very particular meaning. Requesting for null “country” value does not return all cases where the country could not be retrieved, but only cases where the continent could be retrieved but the country could not. Similarly, requesting null “usState” returns cases where the continent and country could be retrieved but the US state could not. This does not include clients that are not situated in the US. If you are interested in finding clients where we could not detect any geographical data, the includeNull filter needs to be applied in the continent sub-property.

In the following example, we are requesting cases where we know that the client is within the US but the state could not be identified:

```
{
      "geography":
          {
              "type": "string",
              "country": "US",
              "usState":
                  {
                      "includeNull": true
                  }
          }
}
```

In the following example, we are requesting cases where the GPU is either “NVIDIA”, “AMD” or null (unidentified):

```
{
      "gpu":
          {
              "type": "stringArray",
              "vendor":
                  {
                      "value": ["NVIDIA", "AMD"],
                      "includeNull": true
                  }
          }
}
```

Data Segmentation

Optionally, data may be segmented by 1 level. When using segmentation, events data is split based on the segment property requested. In order to select a segmentation property, specify any metadata property in the segmentBy field. The way segments are to be generated must be specified in the segments field. The segments field should contain a JSON array containing objects with the following properties:

Table 10-7 • Data Segmentation Properties

Property	Description
`type (string)`	The data type of the value. Can be `string`, `stringArray`, `regex`, `number` or `numberRange` based on whether the property is string-based or numeric.
`value (string/array/number)`	An exact string, an array of strings, a regular expression or a numeric value. This property should not be used if `type` is `numberRange`. Format is based on whether the property is string-based or numeric.
`min (number)`	Used only if the `type` is `numberRange`. Contains the minimum numeric value to include in this segment. May be combined with max.
`max (number)`	Used only if the `type` is `numberRange`. Contains the maximum numeric value to include in this segment. May be combined with min.

Table 10-7 • Data Segmentation Properties

Property	Description
`split (boolean)`	Used only if the `type` is `stringArray` or `regex`. This specifies whether to split the returned data based on each different value matched by the regular expression or array (true), or to join all the clients that match the value as 1 table value or series (false).
`segmentLabel (string)`	Used only if split is false or if `type` is `numberRange`. This is required to give a name to a series when not splitting by value. It is important that the name given is unique.
`limit (integer)`	Optional property to set the limit on the maximum number of table values or series that should be produced by this set of values. To be used only if split is true.

Information about Data Segmentation are presented in the following topics:

- String-Based Segmentation Properties
- Numeric Segmentation Properties
- Boolean Segmentation Properties
- Special Segmentation Properties
- <NULL> Values for Data Segmentation

String-Based Segmentation Properties

The following properties are stored as strings:

```
machineId *
clientId *
prodVersion
prodEdition
prodBuild
prodLanguage
licenseType
formFactor *
osLanguage
osWordLength *
cpuType *
javaVersion *
javaVendor *
javaRuntime *
javaGraphics *
javaVmVersion *
javaVmName *
vm
C01 .. C20 (Custom properties)
licenseKey * **
```

Important • *licenseKey requires a special user permission to be used for segmentation.*

Note • *Properties marked with an asterisk (*) are based on the current (latest known) values.*

The `type` field when using one of the above properties needs to be `string`, `stringArray` or `regex`. A value field is always required. The contents of this field should be according to the specified type. If string is specified, then the value field must contain a single string that needs to be matched precisely with the stored data. If `stringArray` is specified, then the value field must contain an array of strings where one of which needs to match precisely with the stored data. If specifying a `regex`, the value field should contain a string which is treated as a regular expression and the stored data will be matched against it using regular expression rules.

Example using segmentation by `string`, `stringArray`, and `regex` values

```
{
    "segmentBy": "prodVersion",
    "segments": [
        {
            "type": "string",
            "value": "1.0"
        },
        {
            "type": "stringArray",
            "value": ["2.0", "2.1", "3.1"],
            "split": false,
            "segmentLabel": "Versions 2 and 3"
        },
        {
            "type": "regex",
            "value": "^4\..*",
            "split": false,
            "segmentLabel": "All version 4"
        },
        {
            "type": "regex",
            "value": "^5\..*",
            "split": true
        }
    ]
}
```

In the above example, we are requesting a report with multiple segments. The first segment contains installations running version 1.0. Notice how this does not require a “split” property since there is only 1 value and therefore no further splitting is possible. The second segment contains versions 2.0, 2.1 and 3.1. In this case, the “split” property is required, and since we are requesting the API to combine these 3 versions, we must provide a “`segmentLabel`” value so that the returned data can be identified. The third segment is similar, although in this case the request is built using a regular expression. In this case, all versions starting with “4.” are to be included into one combined segment.

The last segment is different from the rest because we are requesting the API to split the data (split is set to true). Therefore, this can produce much more than 1 segment. In this case, we could see segments such as “5.1”, “5.2”, etc. Notice how since we are splitting, we should not provide a `segmentLabel` value since the labels are built using the different values that are found in the data.

Numeric Segmentation Properties

The following properties are stored as numeric values:

```
cpuCores
displayCount
ram
resolutionWidth
resolutionHeight
lifetimeRuntimeMinutes
lifetimeSessionCount
screenPpi
javaVmRam
```

The type field in the above properties needs to be number or `numberRange`. If number is specified, then a value field must also be present. The value field should contain a number, which may contain a decimal point if required. If `numberRange` is specified, then the value field should NOT be used. Instead, the properties min and max are to be used. These refer to the minimum and maximum number to be included in the report. If only one limit needs to be set, the other property is to be left out. Therefore, if you want to include installations with up to 2 display devices, you would not specify a min value, but instead specify only a max and set it as 2.

Example using segmentation by `number`, and `numberRange` values

```
{
    "segmentBy": "cpuCores",
    "segments": [
        {
            "type": "number",
            "value": 1,
        },
        {
            "type": "numberRange",
            "min": 2,
            "max": 4,
            "segmentLabel": "2 - 4"
        },
        {
            "type": "numberRange",
            "min": 5,
            "segmentLabel": "5 +"
        }
    ]
}
```

In the above example, we are requesting a report with 3 segments. The first segment contains only installations running on 1 CPU core, the second segments contains installations running on 2, 3, or 4 cores (range 2 - 4), while the last segment contains all installations which are running on a machine with 5 or more CPU cores. Note how when the type was `numberRange`, we had do specify a `segmentLabel` which is a free string that will be used by the user to identify what is being included in that specific segment.

Boolean Segmentation Properties

The following properties are stored ad boolean values:

```
touchScreen
```

The `type` field needs to be boolean, and the value must be true or false. A `segmentLabel` field is also required

The following example requests data segmented by `touchScreen`:

```
{
    "segmentBy": "touchScreen",
    "segments": [
        {
            "type": "boolean",
            "value": true,
            "segmentLabel": "Yes"
        },
        {
            "type": "boolean",
            "value": false,
            "segmentLabel": "No"
        },
        {
            "includeNull": true,
            "segmentLabel": "Unknown"
        }
    ]
}
```

In the above example, we are requesting a report with 3 segments. The first segment contains installations on which a touch screen was detected, the second one where no touch screen has been detected, while the last one is where we could not detect whether a touch screen is present due to the client using an old SDK which did not have touch screen detection support.

Special Segmentation Properties

Some properties need to be represented in a special format due to their unique requirements. These special properties are:

- Special Segmentation Format: licenseStatus
- Special Segmentation Format: os
- Special Segmentation Format: geography
- Special Segmentation Format: gpu
- Special Segmentation Format: optOut and backOff

Special Segmentation Format: licenseStatus

The `licenseStatus` value is made up of 4 sub-values: activated, blacklisted, expired and whitelisted. These are presented as boolean values. Any number of segments can be defined, and each segment can contain any subset of the 4 sub-values. These values are ANDed together. A `segmentLabel` value is required.

In the following example, 2 segments are specified - the first one showing blacklisted AND not expired and the second one showing whitelisted AND activated:

```
[
    {
        "segmentLabel": "BL and not EXP",
        "blacklisted": true,
        "expired": false
```

```
    },
    {
        "segmentLabel": "WL and ACT",
        "whitelisted": true,
        "expired": true
    }
]
```

Special Segmentation Format: os

The os value is made up of 3 granularity levels - `platform`, `version`, and `edition`. A particular level needs to be selected, and this is to be included in the property name such as os.version or os.edition. For a description of the differences between the 3 granularity levels, refer to the os special filter section.

The following example requests data segmented by all OS versions:

```
{
    "segmentBy": "os.version",
    "segments": [
        {
            "type": "regex",
            "value": ".*",
            "split": true
        }
    ]

}
```

In the above example, no filtering is being done, and instead, a regular expression to include everything is set as the value. This will result in all OS versions to be returned.

Special Segmentation Format: geography

The geography value is made up of 3 granularity levels - `continent`, `country`, and `usState`. These granularity levels are explained in the geography special filter section. A particular level needs to be selected, and this is to be included in the property name such as geography.continent or geography.country.

The following example requests data segmented by all countries:

```
{
    "segmentBy": "geography.country",
    "segments": [
        {
            "type": "regex",
            "value": ".*",
            "split": true
        }
    ]
}
```

In the above example, no filtering is being done, and instead, a regular expression to include everything is set as the value. This will result in all countries to be returned.

Special Segmentation Format: gpu

The gpu value is made up of 2 granularity levels - vendor and model. These granularity levels are explained in Special Filter: gpu. A particular level needs to be selected, and this is to be included in the property name, namely gpu.vendor or gpu.model.

The following example requests data segmented by all GPU vendor:

```
{
    "segmentBy": "gpu.vendor",
    "segments": [
        {
            "type": "regex",
            "value": ".*",
            "split": true
        }
    ]
}
```

In the above example, no filtering is being done, and instead, a regular expression to include everything is set as the value. This will result in all the GPU vendors to be returned.

Special Segmentation Format: optOut and backOff

Both backOff and optOut values are made up of 2 boolean sub-values: historical and current. Any number of segments can be defined, and each segment can contain any subset of the 2 sub-values. These values are ANDed together. A segmentLabel value is required.

In the following example, 2 segments are specified - the first one showing historical AND not current and the second one showing not historical (i.e. never opted-out):

```
[
    {
        "segmentLabel": "HISTORICAL and not CURRENT",
        "historical": true,
        "current": false
    },
    {
        "segmentLabel": "Never opted-out",
        "historical": false
    }
]
```

<NULL> Values for Data Segmentation

Null values in segmentation are to be requested in a similar way to null values in filters. The same properties that support null in filtering also support null in segmentation.

By default, when segmenting, null values are not included within the segments, since only the values that have been specified in each segment are included. Null values don't match any regular expression, so the only way to request null values to be included is to specify "includeNull" as true in a similar way to filtering. In segmentation, null values are returned as "<NULL>". The API considers all cases where the data has never been set from the SDK, set as an empty string, or set as a string containing "<NULL>" to be the same.

The following example requests all values of prodBuild including null:

```
{
    "level1": {
        "property": "prodBuild",
        "segments": [
            {
                "type": "regex",
                "value": ".*",
                "includeNull": true
            }
        ]
    }
}
```

In the case of segmentation properties that use sub-properties (os, geography, and gpu), the `includeNull` value is to be included in the sub-property and applies to that specific sub-property only. In order to be able to include the `includeNull` property, instead of providing the value as a string or an array of strings, the value of the sub-property must be a JSON object that contains a property named "`value`", and another named "`includeNull`". Each of these properties is optional, but at least one of them must be present. The same rules that apply for filtering these types of properties for null values also apply to segmentation.

In the following example, we are requesting segmentation by continent and are also requesting the number of clients where we could not detect the geographical location:

```
{
    "level1": {
        "property": "geography",
        "segments": [
            {
                "type": "regex",
                "continent": {
                                "value": ".*",
                                "includeNull": true
                            }
            }
        ]
    }
}
```

Events Property

This property should be formatted as an array of objects. In each object, one must specify either a single event or a combination of events.

To specify a single event, the inner object should look like the following example:

```
{
     "category": "File Operations",
     "name":"Open"
}
```

To specify a combination of events, the inner object should look like the following:

```
{
     "combiLabel":  "Open and Save",
     "combiArray":  [
                       {
```

```
                    "categoryType": "string",
                    "nameType": "string",
                    "category": "File Operations",
                    "name": "Open"
                  },
                  {
                    "categoryType": "string",
                    "nameType": "string",
                    "category": "File Operations",
                    "name": "Save",
                  }
                ]
}
```

In the above example, we are combining the data from the Open and Save events, both under the File Operations category. In this example, “string” is being used both as categoryType and nameType. The other possible value is “regex”. The following example shows a case where all events under the File Operations category are being combined:

```
{
     "combiLabel":  "All File Operations",
     "combiArray":  [
                    {
                      "categoryType": "string",
                      "nameType": "regex",
                      "category": "File Operations",
                      "name": ".*"
                    }
                  ]
}
```

The `combiLabel` property should contain any user-friendly unique string value to identify each events combination.

Results Format

If segmentation is not being used, the results are presented in an array of objects. Each object inside the array represents an event or a combination of events as requested. If segmentation is used, the data is presented as a JSON object. This object would contain an element for each segment, where the key would be the segment name, and the value would be an array containing event data as described above.

The inner objects inside the array/s contain the following elements:

- **eventCounts**—The total number of recorded occurrences of this event
- **averageEventCountPerSessionUsedAtLeastOnce**—The average number of event occurrences per runtime session from installations which performed this event at least once within the specified date range
- **averageEventCountPerSessionAll**—The average number of event occurrences per runtime session from all installations (including installations who did not perform this event)
- **averageEventCountPerUserAtLeastOnce**—The average number of event occurrences per installations which performed this event at least once within the specified date range
- **averageEventCountPerUserAll** —The average number of event occurrences per installation (including installations who did not perform this event)

- **uniqueUsersUsedAtLeastOnce**—The number of unique installations/users who used this event at least once within the specified date range
- **uniqueUsersNeverUsed** —The number of unique installations/users who never used this event within the specified date range
- **usageFrequency**—Histogram data showing the number of times the event occurred by how many users.

In the example below, no segmentation is being used. As described above, when segmentation is used, the results are presented in an object which contains an element for each segment having an array in the same format as below as the value.

Example Request

```
POST /reporting/eventTracking/advanced/fullReport  HTTP/1.1
Host: api.revulytics.com
Content-Type: application/json
Accept: application/json

{
    "user": "testuser@test.com",
    "sessionId": "VSB8E2BzSC2eZSJm4QmTpA",
    "productId": 12345678901,
    "startDate": "2018-10-01",
    "stopDate": "2018-12-31",
    "events": [
                {
                    "category": "File Operations",
                    "name":"Open"
                },
                {
                    "category": "File Operations",
                    "name":"Save"
                }
            ]
}
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "status": "OK",
    "segmentBy": null,
    "results": {
           "detailedReport": [{
                  "category": "File Operations",
                  "name": "Save",
                  "data": {
                         "eventCounts": 81223,
                         "averageEventCountPerSessionUsedAtLeastOnce": 0.35,
                         "averageEventCountPerSessionAll": 0.28,
                         "averageEventCountPerUserAtLeastOnce": 27.46,
                         "averageEventCountPerUserAll": 22.85,
                         "uniqueUsersUsedAtLeastOnce": 2958,
                         "uniqueUsersNeverUsed": 596,
```

```
                        "usageFrequency": {
                                "1 - 5": 314,
                                "6 - 10": 352,
                                "11 - 15": 334,
                                "16 - 20": 315,
                                "21 - 25": 292,
                                "26 - 30": 316,
                                "31 - 35": 257,
                                "36 - 40": 214,
                                "41 - 45": 142,
                                "46 - 50": 116,
                                "51 - 60": 98,
                                "61 - 70": 51,
                                "\u2265 71": 157
                        }
                }
        }, {
                "category": "File Operations",
                "name": "Open",
                "data": {
                        "eventCounts": 211161,
                        "averageEventCountPerSessionUsedAtLeastOnce": 0.73,
                        "averageEventCountPerSessionAll": 0.73,
                        "averageEventCountPerUserAtLeastOnce": 59.7,
                        "averageEventCountPerUserAll": 59.42,
                        "uniqueUsersUsedAtLeastOnce": 3537,
                        "uniqueUsersNeverUsed": 17,
                        "usageFrequency": {
                                "1 - 5": 192,
                                "6 - 10": 232,
                                "11 - 15": 211,
                                "16 - 20": 221,
                                "21 - 25": 210,
                                "26 - 30": 177,
                                "31 - 35": 178,
                                "36 - 40": 169,
                                "41 - 45": 153,
                                "46 - 50": 120,
                                "51 - 60": 252,
                                "61 - 70": 211,
                                "71 - 80": 173,
                                "81 - 90": 184,
                                "91 - 100": 156,
                                "101 - 110": 135,
                                "111 - 120": 107,
                                "121 - 130": 94,
                                "131 - 140": 69,
                                "141 - 150": 79,
                                "\u2265 151": 214
                        }
                }
        }]
    }
}
```

11

Churn-Related Reports

This collection of reports is meant to provide insight on the lifetime of lost users. These reports are meant to report what happened during the whole lifetime of these installations rather than what happened during a defined date range.

- Churn and Engagement Report
- Runtime Activity Reports for Lost Installations
- Churned User Activity Reports

Churn and Engagement Report

This report shows how new installations during the specified date range were engaged or lost during their lifetime. This report does not only show users who were reported lost, but instead, all users who installed during the specified date range are included. By viewing this report, one can then see how many users were still active/engaged after x days/weeks/months.

- Request/Response Parameters Summary
- Global Filters
- Segmentation
- Results Format

Request/Response Parameters Summary

```
POST /reporting/engagement/churnAndEngagement
```

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 11-1 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. Required only for non-cookie authentication. • **sessionId (string)**—The `sessionId` obtained via POST /auth/login. Required only for non-cookie authentication. • **productId (integer)**—The product ID on which this request is being done • **startDate (string)**—The first date of the date range on which to base the report. This is to be formatted as YYYY-MM-DD. • **stopDate (string)**—The last date of the date range on which to base the report. This is to be formatted as YYYY-MM-DD. • **daysUntilDeclaredLost (integer)**—This specifies the number of consecutive days of inactivity that have to pass until a client installation is declared lost. • **dateSplit (string)** – Whether to present results by day, week, or month. • **globalFilters (object)**—JSON object containing the filters to be applied to the available data. Details about these filters can be found in the Global Filters section. • **segmentBy (string)**—The field with which to segment the data. Details about segmentation can be found in the Segmentation section. • **segments (string)**—Used to specify how data is to be segmented. Must be used in conjunction with `segmentBy`. Details about segmentation can be found in the Segmentation section.
Response JSON Object	• status (string)—Contains OK if successful or `SYNTAX ERROR` or `AUTH ERROR`. • reason (string)—Present only if status is not OK. Contains error message (reason). • **results (object)**—Contains the results as requested represented as a JSON object. The result format is described below.

Global Filters

Most of the available filter properties are string-based. This means that when applying a filter, the requested field can be represented as a `string`, `stringArray` or `regex`. There are also some filters which are numeric. These filters should be represented as `number` or `numberRange`.

The standard filters are based on the value recorded on the install date. Therefore, if a user installed version 1 of the application and then switched to version 2, that particular installation is considered as version 1. However, `currentData` filters (marked with *) are based on the current (last known) values.

- String-Based Filters
- Numeric Filters

- Date Range Filters
- Boolean Filters
- Special Filters
- <NULL> Values in Global Filters

String-Based Filters

The following properties are stored as strings:

```
machineId *
clientId *
prodVersion
prodEdition
prodBuild
prodLanguage
licenseType
formFactor *
osLanguage
osWordLength *
cpuType *
dotNetVersion *
javaVersion *
javaVendor *
javaRuntime *
javaGraphics *
javaVmVersion *
javaVmName *
vm *
C01 .. C20 (Custom properties)
licenseKey *
```

Note • *`licenseKey` requires a special user permission to be used as a filter.*

Note • *Properties marked with an asterisk (*) are based on the current (latest known) values.*

The **type** field in the above filters needs to be `string`, `stringArray` or `regex`. A **value** field is always required. The contents of this field should be according to the specified type.

- If `string` is specified, then the value field must contain a single string that needs to be matched precisely with the stored data.
- If `stringArray` is specified, then the value field must contain an array of strings where one of which needs to match precisely with the stored data.
- If specifying a `regex`, the value field should contain a string which is treated as a regular expression and the stored data will be matched against it using regular expression rules.

Example Filter Using a String Value

In this example, the product build value needs to be exactly “3014.int-12214”:

```
{
      "prodBuild":
          {
              "type": "string",
              "value": "3014.int-12214"
          }
}
```

Example Filter Using a String Array

In this example, the product build value needs to be either “3014.int-12214”, “3017.enx-57718”, or “4180.vrx-81059”. Note that since the type is declared as `stringArray`, the value field needs to contain an array. Consider all elements in the array to have an OR logical expression between them.:

```
{
    "prodBuild":
        {
            "type": "stringArray",
            "value": ["3014.int-12214", "3017.enx-57718", "4180.vrx-81059"]
        }
}
```

Example Filter Using a Regular Expression

In this example, the product build value needs to start with “30” and end with “18” whilst having 10 characters in between:

```
{
     "prodBuild":
         {
             "type": "regex",
             "value": "^30.{10}18$"
         }
}
```

Numeric Filters

The following properties are stored as numeric values:

```
cpuCores *
displayCount *
ram *
resolutionWidth *
resolutionHeight *
lifetimeRuntimeMinutes *
lifetimeSessionCount *
screenPpi *
javaVmRam *
```

Note • *Properties marked with an asterisk (*) are based on the current (latest known) values.*

The type field in the above filters needs to be `number` or `numberRange`.

- If `number` is specified, then a value field must also be present. The value field should contain a number, which may contain a decimal point if required.

- If numberRange is specified, then the value field should NOT be used. Instead, the properties min and max are to be used. These refer to the minimum and maximum number to be included in the report. If only one limit needs to be set, the other property is to be left out. Therefore, if you want to include installations with up to 2 display devices, you would not specify a min value, but instead specify only a max and set it as 2.

Example Filter Using a Number Value

In this example, the number of display devices needs to be exactly 3:

```
{
      "displayCount":
          {
              "type": "number",
              "value": 3
          }
}
```

Example Filter Using a Number Range Value

In this example, the RAM needs to be between 1025MB and 4096MB (both included):

```
{
      "ram":
          {
              "type": "numberRange",
              "min": 1025,
              "max": 4096
          }
}
```

Date Range Filters

The following properties are stored as dates:

```
dateInstalled
dateLastSeen
```

The type field in the above filters needs to be date or dateRange.

- If date is specified, then a value field must also be present. The value field should contain a date.
- If dateRange is specified, then the value field should NOT be used. Instead, the properties min and max are to be used. These refer to the minimum and maximum dates to be included in the report. If only one limit needs to be set, the other property is to be left out.

In the following example, users installed after January 1st 2018 are to be shown:

```
{
    "dateInstalled":
        {
            "type": "dateRange",
            "min": "2018-01-01"
        }
}
```

Note • *All dates must be in ISO 8601 format.*

Boolean Filters

The following property is stored as boolean:

`touchScreen`

The type field in the above filters needs to be boolean. The value must be true or false. In the following filter, clients with a touch screen are being requested.

```
{
    "touchScreen":
        {
            "value": true
        }
}
```

Special Filters

Some filters need to be represented in a special format due to their unique requirements. These special filters are:

- Special Filter: licenseStatus
- Special Filter: os
- Special Filter: geography
- Special Filter: gpu
- Special Filters: optOut and backOff
- Special Filter: lifetimeEventUsage
- Special Filter: reachOutDeliveries

Special Filter: licenseStatus

The `licenseStatus` filter is made up of 4 sub-values: `activated`, `blacklisted`, `expired` and `whitelisted`. These are presented as boolean values.

Unlike other filters, this filter is presented as an array of JSON objects. Each object can contain a subset (or all) of these 4 boolean values.

Consider the following example. In this example, for a client to be included, the license has to either be activated AND whitelisted, or else it can be not whitelisted AND expired. In other words, **(** (activated AND whitelisted) OR ((NOT)whitelisted AND expired) **)**.

```
{
    "licenseStatus":
        [
            {
                "activated": true,
```

```
                "whitelisted": true
            },
            {
                "whitelisted": false,
                "expired": true
            }
        ]
}
```

Special Filter: os

The os filter is made up of 3 granularity levels. These are `platform`, `version`, and `edition`. These are meant to split the OS name into levels of detail as required by the user. Consider the following:

- **platform**: Microsoft Windows
- **version**: Microsoft Windows 7
- **edition**: Microsoft Windows 7 Professional

If a filter is set on the version “Microsoft Windows 7”, the result would include all editions of Windows 7. One or more of these granularity levels may be specified. If more than 1 granularity level is specified, the values are ORed together.

In the following example, all editions of “Microsoft Windows 7” are included, and also “Microsoft Windows Vista Home Premium”:

```
{
    "type": "string",
    "version": "Microsoft Windows 7",
    "edition": "Microsoft Windows Vista Home Premium"
}
```

In the following example, the type is `stringArray`. Note that an array needs to be passed if the type is set as such, even if it is to contain only 1 element. In this case, the version can be either “Microsoft Windows 7” or “Microsoft Windows 8” (which are ORed together). Also, clients running on “Microsoft Windows XP Professional” are to be included.

```
{
    "type": "stringArray",
    "version": ["Microsoft Windows 7", "Microsoft Windows 8"],
    "edition": ["Microsoft Windows XP Professional"]
}
```

Special Filter: geography

The geography filter is made up of 3 granularity levels. These are `continent`, `country`, and `usState`. The usState value applies only to United States. Continents and countries are presented in 2-letter codes. Countries follow ISO standard 3166-1 alpha-2. US states are presented in ISO 3166-2:US format.

In the following example, the clients have to be either:

- In the continents *Asia* or *Oceania*
- In the country *Germany*
- In the US states *New York*, *New Jersey*, or *Kansas*

```
{
```

```
    "type": "stringArray",
    "continent": ["AS", "OC"],
    "country": ["DE"],
    "usState": ["US-NY", "US-NJ", "US-KS"]
}
```

Important • *In this filter, the type can be* `string` *or* `stringArray`*. Regular expressions are not supported in geography filters.*

Special Filter: gpu

The gpu filter is made up of 2 granularity levels. These are `vendor` and `model`. Both are represented as string values.

In the following example, the clients have to have a GPU:

- From the vendors *NVIDIA* or *Intel*
- With the model *AMD Radeon HD 4600*

```
{
    "type": "stringArray",
    "vendor": ["NVIDIA", "Intel"],
    "model": ["AMD Radeon HD 4600"]
}
```

Special Filters: optOut and backOff

The opt-out mechanism was introduced in SDK version 5.1.0. With this feature, vendors can have their application report to the Usage Intelligence servers that a user does not want to be tracked. Using this property, vendors can filter out installations that were opted-out.

Similarly, backoff filtering was introduced with version 5.0.0. Backoff is when a product account runs over-quota and the server starts rejecting data. Although filtering for backed-off installations was introduced with version 5, it was also backported to previous versions. However, when a new installation with an SDK prior to version 5 tries to register with the server and is rejected, it is not marked as being once backed-off when it is eventually accepted by the server. With version 5 onwards, the server flags an installation as being historically backed-off in such cases.

Both `backOff` and `optOut` filters are made up of 2 boolean sub-values: `historical` and `current`.

- The `historical` value refers to installations that were once backed-off or opted-out. These may include installations that are still currently backed-off or opted-out.
- The `current` value refers to the status during the last time that the client called the server. Therefore, if an installation was opted-out yesterday but got opted-in today, it will be marked as historically opted-out but not currently opted-out.

In the following example, for a client to be included, the `optOut` status has to either be `historical` AND not `current`, or else it can be not `historical` (i.e. users have to be currently opted-in but used to be opted-out at some point or never opted out).

```
{
    "optOut":
        [
            {
```

```
            "historical": true,
            "current": false
        },
        {
            "historical": false
        }
    ]
}
```

Special Filter: lifetimeEventUsage

Using lifetime event usage filters, clients can be filtered based on whether a particular event or set of events occurred or not within the client's lifetime. Alternatively, one can set a filter based on the number of times an event has occurred.

In the following example, clients that are included must have done the "File Operations - Open" event at least 5 times to be counted.

```
{
     "category": "File Operations",
     "name": "Open",
     "min": 5
}
```

In the following example, clients must have done either "File Operations - Open" or "File Operations - Save" for a combined total of between 10 to 50 times.

```
{
    "combiArray": [
                    {
                      "categoryType": "string",
                      "nameType": "string",
                      "category": "File Operations",
                      "name": "Open"
                    },
                    {
                      "categoryType": "string",
                      "nameType": "string",
                      "category": "File Operations",
                      "name": "Save",
                    }
                  ],
    "min": 10,
    "max": 50
}
```

In the following example, clients must have done any event within the "File Operations" category for a combined total of not more than 100 times. This is done using a regular expression in the name.

```
{
    "combiArray":  [
                     {
                       "categoryType": "string",
                       "nameType": "regex",
                       "category": "File Operations",
                       "name": ".*"
                     }
```

```
                ],
    "max": 100
}
```

Special Filter: reachOutDeliveries

Using ReachOut delivery filters, clients can be filtered based on whether a particular ReachOut message or a combination of ReachOut messages were delivered or not within the client's lifetime.

The filter consists of a JSON array that includes one or more objects. Each object is a combination of delivered and undelivered campaigns, and the different combinations are ORed together. Therefore, it is possible to show users that either received ReachOut message 1 but not 2, or else received 3 but not 4 as in the following example:

In the following example, we are looking for clients who either received campaign 1 but not 2, OR received campaign 2 but not 3.

```
[
     {"auto": {"delivered": ["1"], "undelivered":["2"]}},
     {"auto": {"delivered": ["2"], "undelivered":["3"]}}
]
```

The above example contains only "auto" ReachOut campaigns. Manual campaigns can be specified using "manual" instead of "auto" as in the above example. Each object can contain a mix of "auto" and "manual" campaigns.

<NULL> Values in Global Filters

Most of the available properties can include null values. There are different reasons why a value would be null. When these are properties that are set by the application, the possible reasons why a value would be null are cases where the value has not been set by the application (such as `prodBuild` never being set), and cases where values are set to an empty string ("") or to a string containing "<NULL>".

One other reason is that although these values have been set, the SDK has not yet had time to sync with the servers to provide this new information. In cases where the properties are set automatically such as hardware or OS related information, the values would be null if the SDK failed to retrieve that value from the OS or if the server failed to identify the value retrieved by the SDK.

Other reasons include cases where Java version is requested from an application that does not use the Java SDK, US state is requested for users who are not running within the US, etc.

The following are the properties that support null values:

`prodVersion`
`prodEdition`
`prodBuild`
`prodLanguage`
`machineId`
`formFactor`
`vm`
`cpuType`
`cpuCores`
`ram`
`resolutionWidth`
`resolutionHeight`
`javaVersion`

```
javaVmVersion
javaVmName
javaVendor
javaRuntime
javaGraphics
osLanguage
licenseKey
C01 .. C20 (Custom properties)
os
geography
gpu
```

Null values can be requested either on their own or as part of a filter containing other values.

The following example would return only cases where the `prodVersion` is null:

```
{
      "prodVersion":
          {
              "includeNull": true
          }
}
```

The following example would return cases where the `prodVersion` is either 1.1, 1.2 or null:

```
{
      "prodVersion":
          {
              "type": "stringArray",
              "value": ["1.1", "1.2"],
              "includeNull": true
          }
}
```

By default, when specifying a filter, null values would not be included. Therefore, in the following example, only clients with `prodVersion` set to 1.1 or 1.2 will be included, while null values are excluded:

```
{
      "prodVersion":
          {
              "type": "stringArray",
              "value": ["1.1", "1.2"]
          }
}
```

However, if no filter is specified, then nulls are included by default. Therefore, if you want to include any value of `prodVersion` as long as it is not null, a `prodVersion` filter needs to be included as follows:

```
{
      "prodVersion":
          {
              "type": "regex",
              "value": ".*",
              "includeNull": false
          }
}
```

In the case of filters that use sub-properties (os, geography, and gpu), the `includeNull` filter is to be included in the sub-property and applies to that specific sub-property only. In order to be able to include the `includeNull` property, instead of providing the value as a string or an array of strings, the value of the sub-property must be a JSON object that contains a property named “value”, and another named “includeNull”. Each of these properties is optional, but at least one of them must be present.

In the case of geography, this has a very particular meaning. Requesting for null “country” value does not return all cases where the country could not be retrieved, but only cases where the continent could be retrieved but the country could not. Similarly, requesting null “usState” returns cases where the continent and country could be retrieved but the US state could not. This does not include clients that are not situated in the US. If you are interested in finding clients where we could not detect any geographical data, the `includeNull` filter needs to be applied in the continent sub-property.

In the following example, we are requesting cases where we know that the client is within the US but the state could not be identified:

```
{
      "geography":
          {
              "type": "string",
              "country": "US",
              "usState":
                  {
                      "includeNull": true
                  }
          }
}
```

In the following example, we are requesting cases where the GPU is either “NVIDIA”, “AMD” or null (unidentified):

```
{
      "gpu":
          {
              "type": "stringArray",
              "vendor":
                  {
                      "value": ["NVIDIA", "AMD"],
                      "includeNull": true
                  }
          }
}
```

Segmentation

The data in this report is segmented based on the specified property. The property used for segmentation is to be specified in the `segmentBy` field. The segments field should specify how the report is to be segmented.

- Segmentation Based on Installation Period
- String-Based Segmentation Properties
- Numeric Segmentation Properties
- Boolean Segmentation Properties
- Special Segmentation Properties
- <NULL> Values for Segmentation

Segmentation Based on Installation Period

This report supports segmentation based on when each client was first seen by the Usage Intelligence system. When using one of these fields, the segments field cannot be used.

- **installationMonth**—The month during which each installation first reported to Usage Intelligence servers
- **installationWeek**—The week during which each installation first reported to Usage Intelligence servers

String-Based Segmentation Properties

The following properties are stored as strings:

```
machineId *
clientId *
prodVersion
prodEdition
prodBuild
prodLanguage
licenseType
formFactor *
osLanguage
osWordLength *
cpuType *
javaVersion *
javaVendor *
javaRuntime *
javaGraphics *
javaVmVersion *
javaVmName *
vm *
C01 .. C20 (Custom properties)
licenseKey *
```

Note • *`licenseKey` requires a special user permission to be used for segmentation.*

Note • *Properties marked with an asterisk (*) are based on the current (latest known) values.*

The **type** field when using one of the above properties needs to be `string`, `stringArray` or `regex`. A **value** field is always required. The contents of this field should be according to the specified type.

- If `string` is specified, then the value field must contain a single string that needs to be matched precisely with the stored data.
- If `stringArray` is specified, then the value field must contain an array of strings where one of which needs to match precisely with the stored data.
- If specifying a `regex`, the value field should contain a string which is treated as a regular expression and the stored data will be matched against it using regular expression rules.

Example Using Segmentation by string, stringArray, and regex Values

```
{
```

```
    "segmentBy": "prodVersion",
    "segments": [
        {
            "type": "string",
            "value": "1.0"
        },
        {
            "type": "stringArray",
            "value": ["2.0", "2.1", "3.1"],
            "split": false,
            "segmentLabel": "Versions 2 and 3"
        },
        {
            "type": "regex",
            "value": "^4\..*",
            "split": false,
            "segmentLabel": "All version 4"
        },
        {
            "type": "regex",
            "value": "^5\..*",
            "split": true
        }
    ]
}
```

In the above example, we are requesting a report with multiple segments. The first segment contains installations running version 1.0. Notice how this does not require a “split” property since there is only 1 value and therefore no further splitting is possible. The second segment contains versions 2.0, 2.1 and 3.1. In this case, the “split” property is required, and since we are requesting the API to combine these 3 versions, we must provide a “segmentLabel” value so that the returned data can be identified. The third segment is similar, although in this case the request is built using a regular expression. In this case, all versions starting with “4.” are to be included into one combined segment.

The last segment is different from the rest because we are requesting the API to split the data (split is set to true). Therefore, this can produce much more than 1 segment. In this case, we could see segments such as “5.1”, “5.2”, etc. Notice how since we are splitting, we should not provide a `segmentLabel` value since the labels are built using the different values that are found in the data.

Numeric Segmentation Properties

The following properties are stored as numeric values:

```
cpuCores
displayCoun
ram
resolutionWidth
resolutionHeight
lifetimeRuntimeMinutes
lifetimeSessionCount
screenPpi
javaVmRam
```

The **type** field in the above properties needs to be number or `numberRange`. If number is specified, then a value field must also be present. The value field should contain a number, which may contain a decimal point if required. If `numberRange` is specified, then the value field should NOT be used. Instead, the properties min and max are to be used. These refer to the

minimum and maximum number to be included in the report. If only one limit needs to be set, the other property is to be left out. Therefore, if you want to include installations with up to 2 display devices, you would not specify a min value, but instead specify only a max and set it as 2.

Example Using Segmentation by number, and numberRange Values

```
{
    "segmentBy": "cpuCores",
    "segments": [
        {
            "type": "number",
            "value": 1,
        },
        {
            "type": "numberRange",
            "min": 2,
            "max": 4,
            "segmentLabel": "2 - 4"
        },
        {
            "type": "numberRange",
            "min": 5,
            "segmentLabel": "5 +"
        }
    ]
}
```

In the above example, we are requesting a report with 3 segments. The first segment contains only installations running on 1 CPU core, the second segments contains installations running on 2, 3, or 4 cores (range 2 - 4), while the last segment contains all installations which are running on a machine with 5 or more CPU cores. Note how when the type was `numberRange`, we had do specify a `segmentLabel` which is a free string that will be used by the user to identify what is being included in that specific segment.

Boolean Segmentation Properties

The following properties are stored as boolean values:

`touchScreen`

The `type` field needs to be boolean, and the value must be true or false. A `segmentLabel` field is also required

The following example requests data segmented by `touchScreen`:

```
{
    "segmentBy": "touchScreen",
    "segments": [
        {
            "type": "boolean",
            "value": true,
            "segmentLabel": "Yes"
        },
        {
            "type": "boolean",
            "value": false,
            "segmentLabel": "No"
        },
```

```
        {
            "includeNull": true,
            "segmentLabel": "Unknown"
        }
    ]
}
```

In the above example, we are requesting a report with 3 segments. The first segment contains installations on which a touch screen was detected, the second one where no touch screen has been detected, while the last one is where we could not detect whether a touch screen is present due to the client using an old SDK which did not have touch screen detection support.

Special Segmentation Properties

Some properties need to be represented in a special format due to their unique requirements. These special properties are:

- Special Segmentation Format: licenseStatus
- Special Segmentation Format: os
- Special Segmentation Format: geography
- Special Segmentation Format: gpu
- Special Segmentation Format: optOut and backOff

Special Segmentation Format: licenseStatus

The `licenseStatus` value is made up of 4 sub-values: `activated`, `blacklisted`, `expired` and `whitelisted`. These are presented as boolean values. Any number of segments can be defined, and each segment can contain any subset of the 4 sub-values. These values are ANDed together. A `segmentLabel` value is required.

In the following example, 2 segments are specified - the first one showing blacklisted AND not expired and the second one showing whitelisted AND activated:

```
[
    {
        "segmentLabel": "BL and not EXP",
        "blacklisted": true,
        "expired": false
    },
    {
        "segmentLabel": "WL and ACT",
        "whitelisted": true,
        "expired": true
    }
]
```

Special Segmentation Format: os

The `os` value is made up of 3 granularity levels - `platform`, `version`, and `edition`. A particular level needs to be selected, and this is to be included in the property name such as `os.version` or `os.edition`. For a description of the differences between the 3 granularity levels, refer to Special Filter: os.

The following example requests data segmented by all OS versions:

```
{
    "segmentBy": "os.version",
    "segments": [
        {
            "type": "regex",
            "value": ".*",
            "split": true
        }
    ]
}
```

In the above example, no filtering is being done, and instead, a regular expression to include everything is set as the value. This will result in all OS versions to be returned.

Special Segmentation Format: geography

The geography value is made up of 3 granularity levels - `continent`, `country`, and `usState`. These granularity levels are explained in Special Filter: geography. A particular level needs to be selected, and this is to be included in the property name such as `geography.continent` or `geography.country`.

The following example requests data segmented by all countries:

```
{
    "segmentBy": "geography.country",
    "segments": [
        {
            "type": "regex",
            "value": ".*",
            "split": true
        }
    ]
```

}In the above example, no filtering is being done, and instead, a regular expression to include everything is set as the value. This will result in all countries to be returned.

Special Segmentation Format: gpu

The gpu value is made up of 2 granularity levels - `vendor` and `model`. These granularity levels are explained in Special Filter: gpu. A particular level needs to be selected, and this is to be included in the property name, namely `gpu.vendor` or `gpu.model`.

The following example requests data segmented by all GPU vendor:

```
{
    "segmentBy": "gpu.vendor",
    "segments": [
        {
            "type": "regex",
            "value": ".*",
            "split": true
        }
    ]
}
```

In the above example, no filtering is being done, and instead, a regular expression to include everything is set as the value. This will result in all the GPU vendors to be returned.

Special Segmentation Format: optOut and backOff

Both `backOff` and `optOut` values are made up of 2 boolean sub-values: `historical` and `current`. Any number of segments can be defined, and each segment can contain any subset of the 2 sub-values. These values are ANDed together. A `segmentLabel` value is required.

In the following example, 2 segments are specified - the first one showing historical AND not current and the second one showing not historical (i.e. never opted-out):

```
[
    {
        "segmentLabel": "HISTORICAL and not CURRENT",
        "historical": true,
        "current": false
    },
    {
        "segmentLabel": "Never opted-out",
        "historical": false
    }
]
```

<NULL> Values for Segmentation

Null values in segmentation are to be requested in a similar way to null values in filters (<NULL> Values in Global Filters). The same properties that support null in filtering also support null in segmentation.

By default, when segmenting, null values are not included within the segments, since only the values that have been specified in each segment are included. Null values don't match any regular expression, so the only way to request null values to be included is to specify "includeNull" as true in a similar way to filtering. In segmentation, null values are returned as "<NULL>". The API considers all cases where the data has never been set from the SDK, set as an empty string, or set as a string containing "<NULL>" to be the same.

The following example requests all values of `prodBuild` including null:

```
{
    "level1": {
        "property": "prodBuild",
        "segments": [
            {
                "type": "regex",
                "value": ".*",
                "includeNull": true
            }
        ]
    }
}
```

In the case of segmentation properties that use sub-properties (os, geography, and gpu), the includeNull value is to be included in the sub-property and applies to that specific sub-property only. In order to be able to include the includeNull property, instead of providing the value as a string or an array of strings, the value of the sub-property must be a JSON

object that cointains a property named “value”, and another named “includeNull”. Each of these properties is optional, but at least one of them must be present. The same rules that apply for filtering these types of properties for null values also apply to segmentation.

In the following example, we are requesting segmentation by continent and are also requesting the number of clients where we could not detect the geographical location:

```
{
    "level1": {
        "property": "geography",
        "segments": [
            {
                "type": "regex",
                "continent": {
                                "value": ".*",
                                "includeNull": true
                            }
            }
        ]
    }
}
```

Results Format

The results are formatted in a JSON object which contains an element for each segment. Each of these elements is an object which contains an element for each day/week/month after the install date. Each of these sub-elements is an object which contains the following properties:

- **engaged**—The number of users that were still active by this day/week/month
- **engagedPercent**—The percentage of users that were still active by this day/week/month
- **lost**—The number of users that got lost by this day/week/month
- **lostPercent**—The percentage of users that got lost by this day/week/month

Example Request

```
POST /reporting/engagement/churnAndEngagement  HTTP/1.1
Host: api.revulytics.com
Content-Type: application/json
Accept: application/json

{
    "user": "testuser@test.com",
    "sessionId": "VSB8E2BzSC2eZSJm4QmTpA",
    "productId": 12345678901,
    "startDate": "2017-11-01",
    "stopDate": "2018-02-31",
    "daysUntilDeclaredLost": 30,
    "dateSplit": "month",
    "segmentBy": "installationMonth"
}
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "status": "OK",
    "results": {
        "2017-11-01": {
            "0": {
                "engaged": 6,
                "engagedPercent": 100,
                "lost": 0,
                "lostPercent": 0
            },
            "1": {
                "engaged": 6,
                "engagedPercent": 100,
                "lost": 0,
                "lostPercent": 0
            },
            "2": {
                "engaged": 6,
                "engagedPercent": 100,
                "lost": 0,
                "lostPercent": 0
            },
            "3": {
                "engaged": 5,
                "engagedPercent": 83.3,
                "lost": 1,
                "lostPercent": 16.7
            }
        },
        "2017-12-01": {
            "0": {
                "engaged": 29,
                "engagedPercent": 100,
                "lost": 0,
                "lostPercent": 0
            },
            "1": {
                "engaged": 28,
                "engagedPercent": 96.6,
                "lost": 1,
                "lostPercent": 3.4
            },
            "2": {
                "engaged": 28,
                "engagedPercent": 96.6,
                "lost": 1,
                "lostPercent": 3.4
            }
        },
        "2018-01-01": {
            "0": {
                "engaged": 2131,
```

```
                "engagedPercent": 100,
                "lost": 0,
                "lostPercent": 0
            },
            "1": {
                "engaged": 2073,
                "engagedPercent": 97.3,
                "lost": 58,
                "lostPercent": 2.7
            }
        },
        "2018-02-01": {
            "0": {
                "engaged": 1655,
                "engagedPercent": 100,
                "lost": 0,
                "lostPercent": 0
            }
        }
    }
}
```


Runtime Activity Reports for Lost Installations

These reports consist of 3 histograms which show the following metrics:

- **Active Days**—The number of days clients were active within their lifetime.
- **Sessions**—The number of times users launched your application.
- **Runtime**—The total amount of time in hours users spent interacting with your application.

For more information, see the following sections:

- Request/Response Parameters Summary
- Global Filters
- Results Format

Request/Response Parameters Summary

Histograms can be requested for the total number of active days, sessions or runtime hours. This is selectable by requesting one of the 3 URLs listed below.

```
POST /reporting/churn/histogram/days
POST /reporting/churn/histogram/sessions
POST /reporting/churn/histogram/runtime
```

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 11-2 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. Required only for non-cookie authentication. • **sessionId (string)**—The `sessionId` obtained via POST /auth/login. Required only for non-cookie authentication. • **productId (integer)**—The product ID on which this request is being done • **startDate (string)**—The first date of the date range on which to base the report. This is to be formatted as YYYY-MM-DD. • **stopDate (string)**—The last date of the date range on which to base the report. This is to be formatted as YYYY-MM-DD. • **daysUntilDeclaredLost (integer)**—This specifies the number of consecutive days of inactivity that have to pass until a client installation is declared lost. • **dateReportedLost (string)**—When an installation is lost, it can either be shown as lost on the date it last contacted the Usage Intelligence servers (`dateLastSeen`) or else it can be shown as lost when it was declared lost (last date when it contacted Usage Intelligence + the number of days specified in `daysUntilDeclaredLost`) (`dateDeclaredLost`). Therefore, the permitted values are `dateLastSeen` and `dateDeclaredLost`. • **globalFilters (object)**—JSON object containing the filters to be applied to the available data. Details about these filters can be found in the Global Filters section.
Response JSON Object	• status (string)—Contains OK if successful or `SYNTAX ERROR` or `AUTH ERROR`. • reason (string)—Present only if status is not OK. Contains error message (reason). • **results (object)**—Contains the results as requested represented as a JSON object. The result format is described below.

Global Filters

These filters are common between both reports documented above. Most of the available filter properties are string-based. This means that when applying a filter, the requested field can be represented as a `string`, `stringArray` or `regex`. There are also some filters which are numeric. These filters should be represented as `number` or `numberRange`.

The data used for the filters in these reports is the latest-known data, or data when lost. Therefore, if a user started using version 1 and then switched to version 2 before getting lost, that user will show up if filtering for version 2 and not for version 1.

- String-Based Filters
- Numeric Filters
- Date Range Filters

String-Based Filters

The following properties are stored as strings:

```
machineId *
clientId *
prodVersion
prodEdition
prodBuild
prodLanguage
licenseType
formFactor *
osLanguage
osWordLength *
cpuType *
dotNetVersion *
javaVersion *
javaVendor *
javaRuntime *
javaGraphics *
javaVmVersion *
javaVmName *
vm *
C01 .. C20 (Custom properties)
licenseKey *
```

Note • *`licenseKey` requires a special user permission to be used as a filter.*

Note • *Properties marked with an asterisk (*) are based on the current (latest known) values.*

The **type** field in the above filters needs to be `string`, `stringArray` or `regex`. A **value** field is always required. The contents of this field should be according to the specified type.

- If `string` is specified, then the value field must contain a single string that needs to be matched precisely with the stored data.
- If `stringArray` is specified, then the value field must contain an array of strings where one of which needs to match precisely with the stored data.
- If specifying a `regex`, the value field should contain a string which is treated as a regular expression and the stored data will be matched against it using regular expression rules.

Example Filter Using a String Value

In this example, the product build value needs to be exactly “3014.int-12214”:

```
{
```

```
      "prodBuild":
          {
              "type": "string",
              "value": "3014.int-12214"
          }
}
```

Example Filter Using a String Array

In this example, the product build value needs to be either “3014.int-12214”, “3017.enx-57718”, or “4180.vrx-81059”. Note that since the type is declared as `stringArray`, the value field needs to contain an array. Consider all elements in the array to have an OR logical expression between them.:

```
{
    "prodBuild":
        {
            "type": "stringArray",
            "value": ["3014.int-12214", "3017.enx-57718", "4180.vrx-81059"]
        }
}
```

Example Filter Using a Regular Expression

In this example, the product build value needs to start with “30” and end with “18” whilst having 10 characters in between:

```
{
     "prodBuild":
         {
             "type": "regex",
             "value": "^30.{10}18$"
         }
}
```

Numeric Filters

The following properties are stored as numeric values:

```
cpuCores *
displayCount *
ram *
resolutionWidth *
resolutionHeight *
lifetimeRuntimeMinutes *
lifetimeSessionCount *
screenPpi *
javaVmRam *
```

Note • *Properties marked with an asterisk (*) are based on the current (latest known) values.*

The type field in the above filters needs to be `number` or `numberRange`.

- If `number` is specified, then a value field must also be present. The value field should contain a number, which may contain a decimal point if required.

- If numberRange is specified, then the value field should NOT be used. Instead, the properties min and max are to be used. These refer to the minimum and maximum number to be included in the report. If only one limit needs to be set, the other property is to be left out. Therefore, if you want to include installations with up to 2 display devices, you would not specify a min value, but instead specify only a max and set it as 2.

Example Filter Using a Number Value

In this example, the number of display devices needs to be exactly 3:

```
{
      "displayCount":
          {
              "type": "number",
              "value": 3
          }
}
```

Example Filter Using a Number Range Value

In this example, the RAM needs to be between 1025MB and 4096MB (both included):

```
{
      "ram":
          {
              "type": "numberRange",
              "min": 1025,
              "max": 4096
          }
}
```

Date Range Filters

The following properties are stored as dates:

```
dateInstalled
dateLastSeen
```

The type field in the above filters needs to be date or dateRange.

- If date is specified, then a value field must also be present. The value field should contain a date.
- If dateRange is specified, then the value field should NOT be used. Instead, the properties min and max are to be used. These refer to the minimum and maximum dates to be included in the report. If only one limit needs to be set, the other property is to be left out.

In the following example, users installed after January 1st 2018 are to be shown:

```
{
    "dateInstalled":
        {
            "type": "dateRange",
            "min": "2018-01-01"
        }
}
```

Note • *All dates must be in ISO 8601 format.*

Boolean Filters

The following property is stored as boolean:

`touchScreen`

The type field in the above filters needs to be boolean. The value must be true or false. In the following filter, clients with a touch screen are being requested.

```
{
    "touchScreen":
        {
            "value": true
        }
}
```

Special Filters

Some filters need to be represented in a special format due to their unique requirements. These special filters are:

- Special Filter: licenseStatus
- Special Filter: os
- Special Filter: geography
- Special Filter: gpu
- Special Filters: optOut and backOff
- Special Filter: lifetimeEventUsage
- Special Filter: reachOutDeliveries

Special Filter: licenseStatus

The `licenseStatus` filter is made up of 4 sub-values: `activated`, `blacklisted`, `expired` and `whitelisted`. These are presented as boolean values.

Unlike other filters, this filter is presented as an array of JSON objects. Each object can contain a subset (or all) of these 4 boolean values.

Consider the following example. In this example, for a client to be included, the license has to either be activated AND whitelisted, or else it can be not whitelisted AND expired. In other words, **(** (activated AND whitelisted) OR ((NOT)whitelisted AND expired) **)**.

```
{
    "licenseStatus":
        [
            {
                "activated": true,
```

```
                "whitelisted": true
            },
            {
                "whitelisted": false,
                "expired": true
            }
        ]
}
```

Special Filter: os

The os filter is made up of 3 granularity levels. These are `platform`, `version`, and `edition`. These are meant to split the OS name into levels of detail as required by the user. Consider the following:

- **platform**: Microsoft Windows
- **version**: Microsoft Windows 7
- **edition**: Microsoft Windows 7 Professional

If a filter is set on the version “Microsoft Windows 7”, the result would include all editions of Windows 7. One or more of these granularity levels may be specified. If more than 1 granularity level is specified, the values are ORed together.

In the following example, all editions of “Microsoft Windows 7” are included, and also “Microsoft Windows Vista Home Premium”:

```
{
    "type": "string",
    "version": "Microsoft Windows 7",
    "edition": "Microsoft Windows Vista Home Premium"
}
```

In the following example, the type is `stringArray`. Note that an array needs to be passed if the type is set as such, even if it is to contain only 1 element. In this case, the version can be either “Microsoft Windows 7” or “Microsoft Windows 8” (which are ORed together). Also, clients running on “Microsoft Windows XP Professional” are to be included.

```
{
    "type": "stringArray",
    "version": ["Microsoft Windows 7", "Microsoft Windows 8"],
    "edition": ["Microsoft Windows XP Professional"]
}
```

Special Filter: geography

The geography filter is made up of 3 granularity levels. These are `continent`, `country`, and `usState`. The usState value applies only to United States. Continents and countries are presented in 2-letter codes. Countries follow ISO standard 3166-1 alpha-2. US states are presented in ISO 3166-2:US format.

In the following example, the clients have to be either:

- In the continents *Asia* or *Oceania*
- In the country *Germany*
- In the US states *New York*, *New Jersey*, or *Kansas*

```
{
```

```
    "type": "stringArray",
    "continent": ["AS", "OC"],
    "country": ["DE"],
    "usState": ["US-NY", "US-NJ", "US-KS"]
}
```

Important • *In this filter, the type can be* `string` *or* `stringArray`*. Regular expressions are not supported in geography filters.*

Special Filter: gpu

The gpu filter is made up of 2 granularity levels. These are `vendor` and `model`. Both are represented as string values.

In the following example, the clients have to have a GPU:

- From the vendors *NVIDIA* or *Intel*
- With the model *AMD Radeon HD 4600*

```
{
    "type": "stringArray",
    "vendor": ["NVIDIA", "Intel"],
    "model": ["AMD Radeon HD 4600"]
}
```

Special Filters: optOut and backOff

The opt-out mechanism was introduced in SDK version 5.1.0. With this feature, vendors can have their application report to the Usage Intelligence servers that a user does not want to be tracked. Using this property, vendors can filter out installations that were opted-out.

Similarly, backoff filtering was introduced with version 5.0.0. Backoff is when a product account runs over-quota and the server starts rejecting data. Although filtering for backed-off installations was introduced with version 5, it was also backported to previous versions. However, when a new installation with an SDK prior to version 5 tries to register with the server and is rejected, it is not marked as being once backed-off when it is eventually accepted by the server. With version 5 onwards, the server flags an installation as being historically backed-off in such cases.

Both `backOff` and `optOut` filters are made up of 2 boolean sub-values: `historical` and `current`.

- The `historical` value refers to installations that were once backed-off or opted-out. These may include installations that are still currently backed-off or opted-out.
- The `current` value refers to the status during the last time that the client called the server. Therefore, if an installation was opted-out yesterday but got opted-in today, it will be marked as historically opted-out but not currently opted-out.

In the following example, for a client to be included, the `optOut` status has to either be `historical` AND not `current`, or else it can be not `historical` (i.e. users have to be currently opted-in but used to be opted-out at some point or never opted out).

```
{
    "optOut":
        [
            {
```

```
                "historical": true,
                "current": false
            },
            {
                "historical": false
            }
        ]
}
```

Special Filter: lifetimeEventUsage

Using lifetime event usage filters, clients can be filtered based on whether a particular event or set of events occurred or not within the client's lifetime. Alternatively, one can set a filter based on the number of times an event has occurred.

In the following example, clients that are included must have done the "File Operations - Open" event at least 5 times to be counted.

```
{
     "category": "File Operations",
     "name": "Open",
     "min": 5
}
```

In the following example, clients must have done either "File Operations - Open" or "File Operations - Save" for a combined total of between 10 to 50 times.

```
{
    "combiArray": [
                    {
                      "categoryType": "string",
                      "nameType": "string",
                      "category": "File Operations",
                      "name": "Open"
                    },
                    {
                      "categoryType": "string",
                      "nameType": "string",
                      "category": "File Operations",
                      "name": "Save",
                    }
                  ],
    "min": 10,
    "max": 50
}
```

In the following example, clients must have done any event within the "File Operations" category for a combined total of not more than 100 times. This is done using a regular expression in the name.

```
{
    "combiArray":  [
                     {
                       "categoryType": "string",
                       "nameType": "regex",
                       "category": "File Operations",
                       "name": ".*"
                     }
```

```
                ],
    "max": 100
}
```

Special Filter: reachOutDeliveries

Using ReachOut delivery filters, clients can be filtered based on whether a particular ReachOut message or a combination of ReachOut messages were delivered or not within the client's lifetime.

The filter consists of a JSON array that includes one or more objects. Each object is a combination of delivered and undelivered campaigns, and the different combinations are ORed together. Therefore, it is possible to show users that either received ReachOut message 1 but not 2, or else received 3 but not 4 as in the following example:

In the following example, we are looking for clients who either received campaign 1 but not 2, OR received campaign 2 but not 3.

```
[
     {"auto": {"delivered": ["1"], "undelivered":["2"]}},
     {"auto": {"delivered": ["2"], "undelivered":["3"]}}
]
```

The above example contains only "auto" ReachOut campaigns. Manual campaigns can be specified using "manual" instead of "auto" as in the above example. Each object can contain a mix of "auto" and "manual" campaigns.

<NULL> Values in Global Filters

Most of the available properties can include null values. There are different reasons why a value would be null. When these are properties that are set by the application, the possible reasons why a value would be null are cases where the value has not been set by the application (such as `prodBuild` never being set), and cases where values are set to an empty string ("") or to a string containing "<NULL>".

One other reason is that although these values have been set, the SDK has not yet had time to sync with the servers to provide this new information. In cases where the properties are set automatically such as hardware or OS related information, the values would be null if the SDK failed to retrieve that value from the OS or if the server failed to identify the value retrieved by the SDK.

Other reasons include cases where Java version is requested from an application that does not use the Java SDK, US state is requested for users who are not running within the US, etc.

The following are the properties that support null values:

`prodVersion`
`prodEdition`
`prodBuild`
`prodLanguage`
`machineId`
`formFactor`
`vm`
`cpuType`
`cpuCores`
`ram`
`resolutionWidth`
`resolutionHeight`
`javaVersion`

```
javaVmVersion
javaVmName
javaVendor
javaRuntime
javaGraphics
osLanguage
licenseKey
C01 .. C20 (Custom properties)
os
geography
gpu
```

Null values can be requested either on their own or as part of a filter containing other values.

The following example would return only cases where the `prodVersion` is null:

```
{
      "prodVersion":
          {
              "includeNull": true
          }
}
```

The following example would return cases where the `prodVersion` is either 1.1, 1.2 or null:

```
{
      "prodVersion":
          {
              "type": "stringArray",
              "value": ["1.1", "1.2"],
              "includeNull": true
          }
}
```

By default, when specifying a filter, null values would not be included. Therefore, in the following example, only clients with `prodVersion` set to 1.1 or 1.2 will be included, while null values are excluded:

```
{
      "prodVersion":
          {
              "type": "stringArray",
              "value": ["1.1", "1.2"]
          }
}
```

However, if no filter is specified, then nulls are included by default. Therefore, if you want to include any value of `prodVersion` as long as it is not null, a `prodVersion` filter needs to be included as follows:

```
{
      "prodVersion":
          {
              "type": "regex",
              "value": ".*",
              "includeNull": false
          }
}
```

In the case of filters that use sub-properties (os, geography, and gpu), the `includeNull` filter is to be included in the sub-property and applies to that specific sub-property only. In order to be able to include the `includeNull` property, instead of providing the value as a string or an array of strings, the value of the sub-property must be a JSON object that contains a property named “value”, and another named “includeNull”. Each of these properties is optional, but at least one of them must be present.

In the case of geography, this has a very particular meaning. Requesting for null “country” value does not return all cases where the country could not be retrieved, but only cases where the continent could be retrieved but the country could not. Similarly, requesting null “usState” returns cases where the continent and country could be retrieved but the US state could not. This does not include clients that are not situated in the US. If you are interested in finding clients where we could not detect any geographical data, the `includeNull` filter needs to be applied in the continent sub-property.

In the following example, we are requesting cases where we know that the client is within the US but the state could not be identified:

```
{
      "geography":
          {
              "type": "string",
              "country": "US",
              "usState":
                  {
                      "includeNull": true
                  }
          }
}
```

In the following example, we are requesting cases where the GPU is either “NVIDIA”, “AMD” or null (unidentified):

```
{
      "gpu":
          {
              "type": "stringArray",
              "vendor":
                  {
                      "value": ["NVIDIA", "AMD"],
                      "includeNull": true
                  }
          }
}
```

Results Format

The results are formatted in a JSON object which contains an element for each histogram bin. Each of these elements contains a numeric value which represents the number of installations which fall under this bin.

Example Request

```
POST /reporting/churn/histogram/runtime  HTTP/1.1
Host: api.revulytics.com
Content-Type: application/json
Accept: application/json

{
    "user": "testuser@test.com",
```

```
    "sessionId": "VSB8E2BzSC2eZSJm4QmTpA",
    "productId": 12345678901,
    "startDate": "2018-10-01",
    "stopDate": "2018-12-31",
    "daysUntilDeclaredLost": 30,
    "globalFilters": {}
}
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "status": "OK",
    "results": {
            "\u2264 0:50": 90,
            "0:51 - 1:00": 2,
            "1:01 - 1:15": 53,
            "1:16 - 1:30": 11,
            "1:31 - 1:45": 51,
            "1:46 - 2:00": 52,
            "2:01 - 2:15": 12,
            "2:16 - 2:30": 41,
            "2:31 - 2:44": 9,
            "2:45 - 3:00": 35,
            "3:01 - 3:30": 49,
            "3:31 - 4:00": 69,
            "4:01 - 4:30": 40,
            "4:31 - 5:00": 35,
            "5:01 - 5:30": 28,
            "5:31 - 6:00": 42,
            "6:01 - 6:30": 26,
            "6:31 - 7:00": 28,
            "7:01 - 7:30": 21,
            "7:31 - 8:00": 28,
            "8:01 - 8:30": 20,
            "8:31 - 9:00": 14,
            "9:01 - 9:30": 20,
            "9:31 - 10:00": 19,
            "10:01 - 10:30": 9,
            "10:31 - 11:00": 19,
            "11:01 - 11:30": 13,
            "11:31 - 12:00": 16,
            "12:01 - 13:00": 23,
            "13:01 - 14:00": 27,
            "14:01 - 15:00": 19,
            "15:01 - 16:00": 17,
            "16:01 - 17:00": 11,
            "17:01 - 18:00": 14,
            "18:01 - 21:00": 36,
            "21:01 - 24:00": 37,
            "24:01 - 27:00": 30,
            "27:01 - 30:00": 28,
            "30:01 - 33:00": 40,
            "33:01 - 36:00": 33,
```

```
            "36:01 - 39:00": 16,
            "\u2265 39:01": 69
        }
    }
```

Churned User Activity Reports

The aim of these reports is to show how events occurred throughout the churned clients' lifetime before they were lost. The data can be presented either as a paged table which shows a list of all events and how many times each occurred, or else as a histogram showing only a subset of events as specified. The histogram shows how much churned clients performed an event throughout their lifetime or their average daily/weekly/monthly usage.

- Data Table Report
- Histogram Report

Data Table Report

This report returns data that is to be represented in tabular format. It contains data about each tracked event, how many times it occurred, how many times each user performed each event on average, etc. The events can be presented either as a flat view or categorized hierarchically based on event category and name. Data for each event can then be segmented by any property as described below.

- Request/Response Parameters Summary
- Global Filters
- Segmentation
- Sorting
- Results Format

Request/Response Parameters Summary

```
POST /reporting/eventTracking/churn/lifetime/dataTable
```

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 11-3 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. Required only for non-cookie authentication. • **sessionId (string)**—The `sessionId` obtained via POST /auth/login. Required only for non-cookie authentication. • **productId (integer)**—The product ID on which this request is being done • **startDate (string)**—The first date of the date range on which to base the report. This is to be formatted as YYYY-MM-DD. • **stopDate (string)**—The last date of the date range on which to base the report. This is to be formatted as YYYY-MM-DD. • **daysUntilDeclaredLost (integer)**—This specifies the number of consecutive days of inactivity that have to pass until a client installation is declared lost. • **dateReportedLost (string)**—When an installation is lost, it can either be shown as lost on the date it last contacted the Usage Intelligence servers (`dateLastSeen`) or else it can be shown as lost when it was declared lost (last date when it contacted Usage Intelligence + the number of days specified in **daysUntilDeclaredLost**) (`dateDeclaredLost`). Therefore, the permitted values are `dateLastSeen` and `dateDeclaredLost`. • **globalFilters (object)**—JSON object containing the filters to be applied to the available data. Details about these filters can be found in the Global Filters section. • **segmentBy (string)**—The field with which to segment the data. Details about segmentation can be found in the Segmentation section. • **segments (string)**—Used to specify how data is to be segmented. Must be used in conjunction with `segmentBy`. Details about segmentation can be found in the Segmentation section. • **categorizeEvents (boolean)**—Whether to return events hierarchically based on category/event name (true) or return a flattened list (false). • **sorting (object)**—Used to specify the values with which to sort and the direction. Details about this field can be found in the sorting section. • **paging (object)**—Optional parameter used to specify how many events to show and the index of the event to start with (starting from 0). These sub-parameters are named limit and `startAt`. Therefore, if showing 10 events per page and requesting page 3, limit should be set to 10 and `startAt` should be set to 20.

Table 11-3 • Request Properties

Property	Description
Response JSON Object	• status (string)—Contains OK if successful or SYNTAX ERROR or AUTH ERROR. • reason (string)—Present only if status is not OK. Contains error message (reason). • **segmentBy (string)**—The same value that was passed as `segmentBy` in the request • **categorizeEvents (boolean)**—The same value that was passed as `categorizeEvents` in the request • **results (array)**—Contains the results as requested represented as a JSON object. The result format is described below.

Global Filters

Most of the available filter properties are string-based. This means that when applying a filter, the requested field can be represented as a `string`, `stringArray` or regex. There are also some filters which are numeric. These filters should be represented as `number` or `numberRange`.

String-Based Filters

The following properties are stored as strings:

```
machineId
clientId
prodVersion
prodEdition
prodBuild
prodLanguage
licenseType
formFactor
osLanguage
osWordLength
cpuType
dotNetVersion
javaVersion
javaVendor
javaRuntime
javaGraphics
javaVmVersion
javaVmName
```

```
vm
C01 .. C20 (Custom properties)
licenseKey
```

Note • *licenseKey requires a special user permission to be used as a filter.*

The **type** field in the above filters needs to be `string`, `stringArray` or `regex`. A **value** field is always required. The contents of this field should be according to the specified type.

- If `string` is specified, then the value field must contain a single string that needs to be matched precisely with the stored data.
- If `stringArray` is specified, then the value field must contain an array of strings where one of which needs to match precisely with the stored data.
- If specifying a `regex`, the value field should contain a string which is treated as a regular expression and the stored data will be matched against it using regular expression rules.

Example Filter Using a String Value

In this example, the product build value needs to be exactly “3014.int-12214”:

```
{
      "prodBuild":
          {
              "type": "string",
              "value": "3014.int-12214"
          }
}
```

Example Filter Using a String Array

In this example, the product build value needs to be either “3014.int-12214”, “3017.enx-57718”, or “4180.vrx-81059”. Note that since the type is declared as `stringArray`, the value field needs to contain an array. Consider all elements in the array to have an OR logical expression between them.:

```
{
    "prodBuild":
        {
            "type": "stringArray",
            "value": ["3014.int-12214", "3017.enx-57718", "4180.vrx-81059"]
        }
}
```

Example Filter Using a Regular Expression

In this example, the product build value needs to start with “30” and end with “18” whilst having 10 characters in between:

```
{
     "prodBuild":
         {
             "type": "regex",
             "value": "^30.{10}18$"
         }
}
```

Numeric Filters

The following properties are stored as numeric values:

```
cpuCores *
displayCount *
ram *
resolutionWidth *
resolutionHeight *
lifetimeRuntimeMinutes *
lifetimeSessionCount *
screenPpi *
javaVmRam *
```

Note • *Properties marked with an asterisk (*) are based on the current (latest known) values.*

The type field in the above filters needs to be `number` or `numberRange`.

- If `number` is specified, then a value field must also be present. The value field should contain a number, which may contain a decimal point if required.
- If `numberRange` is specified, then the value field should NOT be used. Instead, the properties min and max are to be used. These refer to the minimum and maximum number to be included in the report. If only one limit needs to be set, the other property is to be left out. Therefore, if you want to include installations with up to 2 display devices, you would not specify a min value, but instead specify only a max and set it as 2.

Example Filter Using a Number Value

In this example, the number of display devices needs to be exactly 3:

```
{
      "displayCount":
          {
              "type": "number",
              "value": 3
          }
}
```

Example Filter Using a Number Range Value

In this example, the RAM needs to be between 1025MB and 4096MB (both included):

```
{
      "ram":
          {
              "type": "numberRange",
              "min": 1025,
              "max": 4096
          }
}
```

Date Range Filters

The following properties are stored as dates:

`dateInstalled`
`dateLastSeen`

The type field in the above filters needs to be `date` or `dateRange`.

- If `date` is specified, then a value field must also be present. The value field should contain a date.
- If `dateRange` is specified, then the value field should NOT be used. Instead, the properties min and max are to be used. These refer to the minimum and maximum dates to be included in the report. If only one limit needs to be set, the other property is to be left out.

In the following example, users installed after January 1st 2018 are to be shown:

```
{
    "dateInstalled":
        {
            "type": "dateRange",
            "min": "2018-01-01"
        }
}
```

Note • *All dates must be in ISO 8601 format.*

Boolean Filters

The following property is stored as boolean:

`touchScreen`

The type field in the above filters needs to be boolean. The value must be true or false. In the following filter, clients with a touch screen are being requested.

```
{
    "touchScreen":
        {
            "value": true
        }
}
```

Special Filters

Some filters need to be represented in a special format due to their unique requirements. These special filters are:

- Special Filter: licenseStatus
- Special Filter: os
- Special Filter: geography
- Special Filter: gpu
- Special Filters: optOut and backOff
- Special Filter: lifetimeEventUsage
- Special Filter: reachOutDeliveries

Special Filter: licenseStatus

The `licenseStatus` filter is made up of 4 sub-values: `activated`, `blacklisted`, `expired` and `whitelisted`. These are presented as boolean values.

Unlike other filters, this filter is presented as an array of JSON objects. Each object can contain a subset (or all) of these 4 boolean values.

Consider the following example. In this example, for a client to be included, the license has to either be activated AND whitelisted, or else it can be not whitelisted AND expired. In other words, **(** (activated AND whitelisted) OR ((NOT)whitelisted AND expired) **)**.

```
{
    "licenseStatus":
        [
            {
                "activated": true,
                "whitelisted": true
            },
            {
                "whitelisted": false,
                "expired": true
            }
        ]
}
```

Special Filter: os

The os filter is made up of 3 granularity levels. These are `platform`, `version`, and `edition`. These are meant to split the OS name into levels of detail as required by the user. Consider the following:

- **platform**: Microsoft Windows
- **version**: Microsoft Windows 7
- **edition**: Microsoft Windows 7 Professional

If a filter is set on the version “Microsoft Windows 7”, the result would include all editions of Windows 7. One or more of these granularity levels may be specified. If more than 1 granularity level is specified, the values are ORed together.

In the following example, all editions of “Microsoft Windows 7” are included, and also “Microsoft Windows Vista Home Premium”:

```
{
    "type": "string",
    "version": "Microsoft Windows 7",
    "edition": "Microsoft Windows Vista Home Premium"
}
```

In the following example, the type is `stringArray`. Note that an array needs to be passed if the type is set as such, even if it is to contain only 1 element. In this case, the version can be either “Microsoft Windows 7” or “Microsoft Windows 8” (which are ORed together). Also, clients running on “Microsoft Windows XP Professional” are to be included.

```
{
    "type": "stringArray",
    "version": ["Microsoft Windows 7", "Microsoft Windows 8"],
    "edition": ["Microsoft Windows XP Professional"]
}
```

Special Filter: geography

The geography filter is made up of 3 granularity levels. These are `continent`, `country`, and `usState`. The usState value applies only to United States. Continents and countries are presented in 2-letter codes. Countries follow ISO standard 3166-1 alpha-2. US states are presented in ISO 3166-2:US format.

In the following example, the clients have to be either:

- In the continents *Asia* or *Oceania*
- In the country *Germany*
- In the US states *New York*, *New Jersey*, or *Kansas*

```
{
    "type": "stringArray",
    "continent": ["AS", "OC"],
    "country": ["DE"],
    "usState": ["US-NY", "US-NJ", "US-KS"]
}
```

Important • *In this filter, the type can be `string` or `stringArray`. Regular expressions are not supported in geography filters.*

Special Filter: gpu

The gpu filter is made up of 2 granularity levels. These are `vendor` and `model`. Both are represented as string values.

In the following example, the clients have to have a GPU:

- From the vendors *NVIDIA* or *Intel*
- With the model *AMD Radeon HD 4600*

```
{
    "type": "stringArray",
    "vendor": ["NVIDIA", "Intel"],
    "model": ["AMD Radeon HD 4600"]
}
```

Special Filters: optOut and backOff

The opt-out mechanism was introduced in SDK version 5.1.0. With this feature, vendors can have their application report to the Usage Intelligence servers that a user does not want to be tracked. Using this property, vendors can filter out installations that were opted-out.

Similarly, backoff filtering was introduced with version 5.0.0. Backoff is when a product account runs over-quota and the server starts rejecting data. Although filtering for backed-off installations was introduced with version 5, it was also backported to previous versions. However, when a new installation with an SDK prior to version 5 tries to register with the server and is rejected, it is not marked as being once backed-off when it is eventually accepted by the server. With version 5 onwards, the server flags an installation as being historically backed-off in such cases.

Both `backOff` and `optOut` filters are made up of 2 boolean sub-values: `historical` and `current`.

- The `historical` value refers to installations that were once backed-off or opted-out. These may include installations that are still currently backed-off or opted-out.

- The `current` value refers to the status during the last time that the client called the server. Therefore, if an installation was opted-out yesterday but got opted-in today, it will be marked as historically opted-out but not currently opted-out.

In the following example, for a client to be included, the `optOut` status has to either be `historical` AND not `current`, or else it can be not `historical` (i.e. users have to be currently opted-in but used to be opted-out at some point or never opted out).

```
{
    "optOut":
        [
            {
                "historical": true,
                "current": false
            },
            {
                "historical": false
            }
        ]
}
```

Special Filter: lifetimeEventUsage

Using lifetime event usage filters, clients can be filtered based on whether a particular event or set of events occurred or not within the client's lifetime. Alternatively, one can set a filter based on the number of times an event has occurred.

In the following example, clients that are included must have done the "File Operations - Open" event at least 5 times to be counted.

```
{
     "category": "File Operations",
     "name": "Open",
     "min": 5
}
```

In the following example, clients must have done either "File Operations - Open" or "File Operations - Save" for a combined total of between 10 to 50 times.

```
{
    "combiArray": [
                    {
                      "categoryType": "string",
                      "nameType": "string",
                      "category": "File Operations",
                      "name": "Open"
                    },
                    {
                      "categoryType": "string",
                      "nameType": "string",
                      "category": "File Operations",
                      "name": "Save",
                    }
                  ],
    "min": 10,
    "max": 50
}
```

In the following example, clients must have done any event within the “File Operations” category for a combined total of not more than 100 times. This is done using a regular expression in the name.

```
{
    "combiArray":  [
                      {
                        "categoryType": "string",
                        "nameType": "regex",
                        "category": "File Operations",
                        "name": ".*"
                      }
                    ],
    "max": 100
}
```

Special Filter: reachOutDeliveries

Using ReachOut delivery filters, clients can be filtered based on whether a particular ReachOut message or a combination of ReachOut messages were delivered or not within the client’s lifetime.

The filter consists of a JSON array that includes one or more objects. Each object is a combination of delivered and undelivered campaigns, and the different combinations are ORed together. Therefore, it is possible to show users that either received ReachOut message 1 but not 2, or else received 3 but not 4 as in the following example:

In the following example, we are looking for clients who either received campaign 1 but not 2, OR received campaign 2 but not 3.

```
[
     {"auto": {"delivered": ["1"], "undelivered":["2"]}},
     {"auto": {"delivered": ["2"], "undelivered":["3"]}}
]
```

The above example contains only “auto” ReachOut campaigns. Manual campaigns can be specified using “manual” instead of “auto” as in the above example. Each object can contain a mix of “auto” and “manual” campaigns.

<NULL> Values in Global Filters

Most of the available properties can include null values. There are different reasons why a value would be null. When these are properties that are set by the application, the possible reasons why a value would be null are cases where the value has not been set by the application (such as `prodBuild` never being set), and cases where values are set to an empty string (“”) or to a string containing “<NULL>”.

One other reason is that although these values have been set, the SDK has not yet had time to sync with the servers to provide this new information. In cases where the properties are set automatically such as hardware or OS related information, the values would be null if the SDK failed to retrieve that value from the OS or if the server failed to identify the value retrieved by the SDK.

Other reasons include cases where Java version is requested from an application that does not use the Java SDK, US state is requested for users who are not running within the US, etc.

The following are the properties that support null values:

`prodVersion`
`prodEdition`
`prodBuild`
`prodLanguage`

```
machineId
formFactor
vm
cpuType
cpuCores
ram
resolutionWidth
resolutionHeight
javaVersion
javaVmVersion
javaVmName
javaVendor
javaRuntime
javaGraphics
osLanguage
licenseKey
C01 .. C20 (Custom properties)
os
geography
gpu
```

Null values can be requested either on their own or as part of a filter containing other values.

The following example would return only cases where the `prodVersion` is null:

```
{
      "prodVersion":
          {
              "includeNull": true
          }
}
```

The following example would return cases where the `prodVersion` is either 1.1, 1.2 or null:

```
{
      "prodVersion":
          {
              "type": "stringArray",
              "value": ["1.1", "1.2"],
              "includeNull": true
          }
}
```

By default, when specifying a filter, null values would not be included. Therefore, in the following example, only clients with `prodVersion` set to 1.1 or 1.2 will be included, while null values are excluded:

```
{
      "prodVersion":
          {
              "type": "stringArray",
              "value": ["1.1", "1.2"]
          }
}
```

However, if no filter is specified, then nulls are included by default. Therefore, if you want to include any value of `prodVersion` as long as it is not null, a `prodVersion` filter needs to be included as follows:

```
{
      "prodVersion":
```

```
            {
                "type": "regex",
                "value": ".*",
                "includeNull": false
            }
}
```

In the case of filters that use sub-properties (os, geography, and gpu), the `includeNull` filter is to be included in the sub-property and applies to that specific sub-property only. In order to be able to include the `includeNull` property, instead of providing the value as a string or an array of strings, the value of the sub-property must be a JSON object that contains a property named “value”, and another named “includeNull”. Each of these properties is optional, but at least one of them must be present.

In the case of geography, this has a very particular meaning. Requesting for null “country” value does not return all cases where the country could not be retrieved, but only cases where the continent could be retrieved but the country could not. Similarly, requesting null “usState” returns cases where the continent and country could be retrieved but the US state could not. This does not include clients that are not situated in the US. If you are interested in finding clients where we could not detect any geographical data, the `includeNull` filter needs to be applied in the continent sub-property.

In the following example, we are requesting cases where we know that the client is within the US but the state could not be identified:

```
{
      "geography":
            {
                "type": "string",
                "country": "US",
                "usState":
                    {
                        "includeNull": true
                    }
            }
}
```

In the following example, we are requesting cases where the GPU is either “NVIDIA”, “AMD” or null (unidentified):

```
{
      "gpu":
            {
                "type": "stringArray",
                "vendor":
                    {
                        "value": ["NVIDIA", "AMD"],
                        "includeNull": true
                    }
            }
}
```

Segmentation

The data in this report is segmented based on the specified property. The property used for segmentation is to be specified in the `segmentBy` field. The segments field should specify how the report is to be segmented.

- String-Based Segmentation Properties
- Numeric Segmentation Properties

- Boolean Segmentation Properties
- Special Segmentation Properties
- <NULL> Values for Segmentation

String-Based Segmentation Properties

The following properties are stored as strings:

```
machineId
clientId
prodVersion
prodEdition
prodBuild
prodLanguage
licenseType
formFactor
osLanguage
osWordLength
cpuType
javaVersion
javaVendor
javaRuntime
javaGraphics
javaVmVersion
javaVmName
vm
C01 .. C20 (Custom properties)
licenseKey
```

Note • *`licenseKey` requires a special user permission to be used for segmentation.*

The **type** field when using one of the above properties needs to be `string`, `stringArray` or `regex`. A **value** field is always required. The contents of this field should be according to the specified type.

- If `string` is specified, then the value field must contain a single string that needs to be matched precisely with the stored data.
- If `stringArray` is specified, then the value field must contain an array of strings where one of which needs to match precisely with the stored data.
- If specifying a `regex`, the value field should contain a string which is treated as a regular expression and the stored data will be matched against it using regular expression rules.

Example Using Segmentation by string, stringArray, and regex Values

```
{
    "segmentBy": "prodVersion",
    "segments": [
        {
            "type": "string",
            "value": "1.0"
        },
        {
```

```
            "type": "stringArray",
            "value": ["2.0", "2.1", "3.1"],
            "split": false,
            "segmentLabel": "Versions 2 and 3"
        },
        {
            "type": "regex",
            "value": "^4\..*",
            "split": false,
            "segmentLabel": "All version 4"
        },
        {
            "type": "regex",
            "value": "^5\..*",
            "split": true
        }
    ]
}
```

In the above example, we are requesting a report with multiple segments. The first segment contains installations running version 1.0. Notice how this does not require a “split” property since there is only 1 value and therefore no further splitting is possible. The second segment contains versions 2.0, 2.1 and 3.1. In this case, the “split” property is required, and since we are requesting the API to combine these 3 versions, we must provide a “segmentLabel” value so that the returned data can be identified. The third segment is similar, although in this case the request is built using a regular expression. In this case, all versions starting with “4.” are to be included into one combined segment.

The last segment is different from the rest because we are requesting the API to split the data (split is set to true). Therefore, this can produce much more than 1 segment. In this case, we could see segments such as “5.1”, “5.2”, etc. Notice how since we are splitting, we should not provide a `segmentLabel` value since the labels are built using the different values that are found in the data.

Numeric Segmentation Properties

The following properties are stored as numeric values:

```
cpuCores
displayCoun
ram
resolutionWidth
resolutionHeight
lifetimeRuntimeMinutes
lifetimeSessionCount
screenPpi
javaVmRam
```

The **type** field in the above properties needs to be number or `numberRange`. If number is specified, then a value field must also be present. The value field should contain a number, which may contain a decimal point if required. If `numberRange` is specified, then the value field should NOT be used. Instead, the properties min and max are to be used. These refer to the minimum and maximum number to be included in the report. If only one limit needs to be set, the other property is to be left out. Therefore, if you want to include installations with up to 2 display devices, you would not specify a min value, but instead specify only a max and set it as 2.

Example Using Segmentation by number, and numberRange Values

```
{
```

```
    "segmentBy": "cpuCores",
    "segments": [
        {
            "type": "number",
            "value": 1,
        },
        {
            "type": "numberRange",
            "min": 2,
            "max": 4,
            "segmentLabel": "2 - 4"
        },
        {
            "type": "numberRange",
            "min": 5,
            "segmentLabel": "5 +"
        }
    ]
}
```

In the above example, we are requesting a report with 3 segments. The first segment contains only installations running on 1 CPU core, the second segments contains installations running on 2, 3, or 4 cores (range 2 - 4), while the last segment contains all installations which are running on a machine with 5 or more CPU cores. Note how when the type was `numberRange`, we had do specify a `segmentLabel` which is a free string that will be used by the user to identify what is being included in that specific segment.

Boolean Segmentation Properties

The following properties are stored as boolean values:

`touchScreen`

The `type` field needs to be boolean, and the value must be true or false. A `segmentLabel` field is also required

The following example requests data segmented by `touchScreen`:

```
{
    "segmentBy": "touchScreen",
    "segments": [
        {
            "type": "boolean",
            "value": true,
            "segmentLabel": "Yes"
        },
        {
            "type": "boolean",
            "value": false,
            "segmentLabel": "No"
        },
        {
            "includeNull": true,
            "segmentLabel": "Unknown"
        }
    ]
}
```

In the above example, we are requesting a report with 3 segments. The first segment contains installations on which a touch screen was detected, the second one where no touch screen has been detected, while the last one is where we could not detect whether a touch screen is present due to the client using an old SDK which did not have touch screen detection support.

Special Segmentation Properties

Some properties need to be represented in a special format due to their unique requirements. These special properties are:

- Special Segmentation Format: licenseStatus
- Special Segmentation Format: os
- Special Segmentation Format: geography
- Special Segmentation Format: gpu
- Special Segmentation Format: optOut and backOff

Special Segmentation Format: licenseStatus

The `licenseStatus` value is made up of 4 sub-values: `activated`, `blacklisted`, `expired` and `whitelisted`. These are presented as boolean values. Any number of segments can be defined, and each segment can contain any subset of the 4 sub-values. These values are ANDed together. A `segmentLabel` value is required.

In the following example, 2 segments are specified - the first one showing blacklisted AND not expired and the second one showing whitelisted AND activated:

```
[
    {
        "segmentLabel": "BL and not EXP",
        "blacklisted": true,
        "expired": false
    },
    {
        "segmentLabel": "WL and ACT",
        "whitelisted": true,
        "expired": true
    }
]
```

Special Segmentation Format: os

The `os` value is made up of 3 granularity levels - `platform`, `version`, and `edition`. A particular level needs to be selected, and this is to be included in the property name such as `os.version` or `os.edition`. For a description of the differences between the 3 granularity levels, refer to Special Filter: os.

The following example requests data segmented by all OS versions:

```
{
    "segmentBy": "os.version",
    "segments": [
        {
            "type": "regex",
            "value": ".*",
            "split": true
```

```
        }
    ]
}
```

In the above example, no filtering is being done, and instead, a regular expression to include everything is set as the value. This will result in all OS versions to be returned.

Special Segmentation Format: geography

The geography value is made up of 3 granularity levels - `continent`, `country`, and `usState`. These granularity levels are explained in Special Filter: geography. A particular level needs to be selected, and this is to be included in the property name such as `geography.continent` or `geography.country`.

The following example requests data segmented by all countries:

```
{
    "segmentBy": "geography.country",
    "segments": [
        {
            "type": "regex",
            "value": ".*",
            "split": true
        }
    ]
```

}In the above example, no filtering is being done, and instead, a regular expression to include everything is set as the value. This will result in all countries to be returned.

Special Segmentation Format: gpu

The `gpu` value is made up of 2 granularity levels - `vendor` and `model`. These granularity levels are explained in Special Filter: gpu. A particular level needs to be selected, and this is to be included in the property name, namely `gpu.vendor` or `gpu.model`.

The following example requests data segmented by all GPU vendor:

```
{
    "segmentBy": "gpu.vendor",
    "segments": [
        {
            "type": "regex",
            "value": ".*",
            "split": true
        }
    ]
}
```

In the above example, no filtering is being done, and instead, a regular expression to include everything is set as the value. This will result in all the GPU vendors to be returned.

Special Segmentation Format: optOut and backOff

Both `backOff` and `optOut` values are made up of 2 boolean sub-values: `historical` and `current`. Any number of segments can be defined, and each segment can contain any subset of the 2 sub-values. These values are ANDed together. A `segmentLabel` value is required.

In the following example, 2 segments are specified - the first one showing historical AND not current and the second one showing not historical (i.e. never opted-out):

```
[
    {
        "segmentLabel": "HISTORICAL and not CURRENT",
        "historical": true,
        "current": false
    },
    {
        "segmentLabel": "Never opted-out",
        "historical": false
    }
]
```

<NULL> Values for Segmentation

Null values in segmentation are to be requested in a similar way to null values in filters (<NULL> Values in Global Filters). The same properties that support null in filtering also support null in segmentation.

By default, when segmenting, null values are not included within the segments, since only the values that have been specified in each segment are included. Null values don't match any regular expression, so the only way to request null values to be included is to specify "includeNull" as true in a similar way to filtering. In segmentation, null values are returned as "<NULL>". The API considers all cases where the data has never been set from the SDK, set as an empty string, or set as a string containing "<NULL>" to be the same.

The following example requests all values of `prodBuild` including null:

```
{
    "level1": {
        "property": "prodBuild",
        "segments": [
            {
                "type": "regex",
                "value": ".*",
                "includeNull": true
            }
        ]
    }
}
```

In the case of segmentation properties that use sub-properties (os, geography, and gpu), the includeNull value is to be included in the sub-property and applies to that specific sub-property only. In order to be able to include the includeNull property, instead of providing the value as a string or an array of strings, the value of the sub-property must be a JSON object that cointains a property named "value", and another named "includeNull". Each of these properties is optional, but at least one of them must be present. The same rules that apply for filtering these types of properties for null values also apply to segmentation.

In the following example, we are requesting segmentation by continent and are also requesting the number of clients where we could not detect the geographical location:

```
{
    "level1": {
        "property": "geography",
        "segments": [
            {
```

```
                "type": "regex",
                "continent": {
                                "value": ".*",
                                "includeNull": true
                            }
            }
        ]
    }
}
```

Sorting

The sorting parameter expects a JSON object which is made up of the following:

- **events (string)**—The value with which the event names are to be sorted. Possible values are `alpha` (alphabetical sorting), `eventCounts`, or `uniqueUsersUsedAtLeastOnce`.
- **eventsDirection (string)**—Whether to sort event names in ascending or descending order. Possible values are `asc` or `desc`.
- **segments (string)**—The value with which the segments are to be sorted. Possible values are `alpha` (alphabetical sorting), `active` (sorting by count of active users), `eventCounts`, `uniqueUsersUsedAtLeastOnce`, or `percentUsedAtLeastOnce`.
- **segmentsDirection (string)**—Whether to sort segments in ascending or descending order. Possible values are `asc` or `desc`.

Results Format

The structure of the results depends on whether the `categorizeEvents` property is set to `true` or `false`, and whether segmentation is being used.

The simplest form is when events are not categorized and segmentation is disabled. In this case, the `results` property contains an array of objects each containing details about each event (event category and name), how many times the event occurred in total, how many times each event occurred per user on average, etc.

When setting `categorizeEvents` to `true`, the results are presented into 2 levels. The results are still an array of objects, but each object consists of only 2 properties - `eventCategory` and categoryData. The `categoryData` value is an array of objects, and each object contains the data for one event within that particular category. The only difference is that this sub-object does not contain the `eventCategory` property since that property is already present in the upper level.

Segmentation adds another level in the results. When using segmentation, each event object contains a property named segments. The segments value is an array of objects, with each object containing data about one segment. The segment name can be found in the property value named `segmentLabel`. The rest of the properties are common with the properties inside the event objects.

The following are the properties contained in the data objects:

Table 11-4 • Results Format Properties

Property	Description
`eventCategory (string)`	Present only in event objects when `categorizeEvents` is set to `false`. Contains the event category.
`eventName (string)`	Present in event objects. Contains the event name.
`segmentLabel (string)`	Present in segment objects. Contains the name/label of the segment.
`eventCount (integer)`	The total number of times that the event occurred.
`eventCountPerUserUsedAtLeastOnce (float)`	The average number of times that this event occurred for each client that performed this event at least once.
`eventCountPerUserUsedAtLeastOnceActiveDay (float)`	The average number of times that this event occurred per day for each client that performed this event at least once.
`eventCountPerUserUsedAtLeastOnceActiveWeek (float)`	The average number of times that this event occurred per week for each client that performed this event at least once.
`eventCountPerUserUsedAtLeastOnceActiveMonth (float)`	The average number of times that this event occurred per month for each client that performed this event at least once.
`usersUsedAtLeastOnce (integer)`	The number of clients which performed this event at least once.
`percentUsedAtLeastOnce (float)`	The percentage of clients which performed this event at least once.
`usersNeverUsed (integer)`	The number of clients which never performed this event.
`percentNeverUsed (float)`	The percentage of clients which never performed this event.
`segments (object)`	Present only in event objects when segmentation is being used. Contains an array of objects - one object for each segment.

Example Response with No Event Categorization and No Segmentation

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json

{
    "status": "OK",
    "segmentBy": null,
    "categorizeEvents": false,
    "results": [
                {
                    "eventCategory": "File Menu",
                    "eventName": "Open",
                    "eventCount": 5000,
                    "eventCountPerUserUsedAtLeastOnce": 20.0,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 2.62,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 9.84,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 29.96,
                    "usersUsedAtLeastOnce": 250,
                    "percentUsedAtLeastOnce": 50.0,
                    "usersNeverUsed": 250,
                    "percentNeverUsed": 50.0
                },
                {
                    "eventCategory": "File Menu",
                    "eventName": "Save",
                    "eventCount": 2000,
                    "eventCountPerUserUsedAtLeastOnce": 20.0,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 2.62,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 9.84,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 29.96,
                    "usersUsedAtLeastOnce": 100,
                    "percentUsedAtLeastOnce": 20.0,
                    "usersNeverUsed": 400,
                    "percentNeverUsed": 80.0
                },
                {
                    "eventCategory": "Edit Menu",
                    "eventName": "Enlarge",
                    "eventCountPerUser": 1.0,
                    "eventCountPerUserUsedAtLeastOnce": 0.3,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 2.62,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 9.84,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 29.96,
                    "usersUsedAtLeastOnce": 150,
                    "percentUsedAtLeastOnce": 30.0,
                    "usersNeverUsed": 350,
                    "percentNeverUsed": 70.0
                }
            ]
```

Example Response with Event Categorization and Segmentation by prodVersion

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "status": "OK",
    "segmentBy": "prodVersion",
```

```
"categorizeEvents": true,
"results": [
            {
                "eventCategory": "File Menu",
                "categoryData":
                    [
                        {
                            "eventName": "Open",
                            "eventCount": 5000,
                            "eventCountPerUserUsedAtLeastOnce": 20.0,
                            "eventCountPerUserUsedAtLeastOnceActiveDay": 2.62,
                            "eventCountPerUserUsedAtLeastOnceActiveDay": 9.84,
                            "eventCountPerUserUsedAtLeastOnceActiveDay": 29.96,
                            "usersUsedAtLeastOnce": 250,
                            "percentUsedAtLeastOnce": 50.0,
                            "usersNeverUsed": 250,
                            "percentNeverUsed": 50.0
                            "segments":
                                [
                                    {
                                        "segmentLabel": "V1.1",
                                        "eventCount": 3000,
                                        "eventCountPerUserUsedAtLeastOnce": 20.0,
                                        "eventCountPerUserUsedAtLeastOnceActiveDay": 2.62,
                                        "eventCountPerUserUsedAtLeastOnceActiveDay": 9.84,
                                        "eventCountPerUserUsedAtLeastOnceActiveDay": 29.96,
                                        "usersUsedAtLeastOnce": 150,
                                        "percentUsedAtLeastOnce": 33.33,
                                        "usersNeverUsed": 300,
                                        "percentNeverUsed": 66.66
                                    },
                                    {
                                        "segmentLabel": "V1.5",
                                        "eventCount": 2000,
                                        "eventCountPerUserUsedAtLeastOnce": 10.0,
                                        "eventCountPerUserUsedAtLeastOnceActiveDay": 1.85,
                                        "eventCountPerUserUsedAtLeastOnceActiveDay": 4.27,
                                        "eventCountPerUserUsedAtLeastOnceActiveDay": 14.92,
                                        "usersUsedAtLeastOnce": 100,
                                        "percentUsedAtLeastOnce": 33.33,
                                        "usersNeverUsed": 200,
                                        "percentNeverUsed": 66.66
                                    }
                                ]
                        },
                        {
                            "eventName": "Save",
                            "eventCount": 2000,
                            "eventCountPerUserUsedAtLeastOnce": 20.0,
                            "eventCountPerUserUsedAtLeastOnceActiveDay": 2.62,
                            "eventCountPerUserUsedAtLeastOnceActiveDay": 9.84,
                            "eventCountPerUserUsedAtLeastOnceActiveDay": 29.96,
                            "usersUsedAtLeastOnce": 100,
                            "percentUsedAtLeastOnce": 20.0,
                            "usersNeverUsed": 400,
                            "percentNeverUsed": 80.0,
```

```
                    "segments":
                        [
                            {
                                "segmentLabel": "V1.1",
                                "eventCount": 1300,
                                "eventCountPerUserUsedAtLeastOnce": 13.0,
                                "eventCountPerUserUsedAtLeastOnceActiveDay": 4.98,
                                "eventCountPerUserUsedAtLeastOnceActiveDay": 11.31,
                                "eventCountPerUserUsedAtLeastOnceActiveDay": 24.18,
                                "usersUsedAtLeastOnce": 150,
                                "percentUsedAtLeastOnce": 40.0,
                                "usersNeverUsed": 100,
                                "percentNeverUsed": 60.0
                            },
                            {
                                "segmentLabel": "V1.5",
                                "eventCount": 700,
                                "eventCountPerUserUsedAtLeastOnce": 7.0,
                                "eventCountPerUserUsedAtLeastOnceActiveDay": 3.1,
                                "eventCountPerUserUsedAtLeastOnceActiveDay": 7.27,
                                "eventCountPerUserUsedAtLeastOnceActiveDay": 11.82,
                                "usersUsedAtLeastOnce": 100,
                                "percentUsedAtLeastOnce": 25.0,
                                "usersNeverUsed": 300,
                                "percentNeverUsed": 75.0
                            }
                        ]
                }
            ]
    },
    {
        "eventCategory": "Edit Menu",
        "categoryData":
            [
                {
                    "eventName": "Enlarge",
                    "eventCount": 500,
                    "eventCountPerUserUsedAtLeastOnce": 0.3,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 2.62,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 9.84,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 29.96,
                    "usersUsedAtLeastOnce": 150,
                    "percentUsedAtLeastOnce": 30.0,
                    "usersNeverUsed": 350,
                    "percentNeverUsed": 70.0,
                    "segments":
                        [
                            {
                                "segmentLabel": "V1.1",
                                "eventCount": 200,
                                "eventCountPerUserUsedAtLeastOnce": 3.33,
                                "eventCountPerUserUsedAtLeastOnceActiveDay": 4.32,
                                "eventCountPerUserUsedAtLeastOnceActiveDay": 12.47,
                                "eventCountPerUserUsedAtLeastOnceActiveDay": 25.51,
                                "usersUsedAtLeastOnce": 60,
                                "percentUsedAtLeastOnce": 28.57,
```

```
                    "usersNeverUsed": 150,
                    "percentNeverUsed": 71.43
                },
                {
                    "segmentLabel": "V1.5",
                    "eventCount": 300,
                    "eventCountPerUserUsedAtLeastOnce": 3.33,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 5.33,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 6.42,
                    "eventCountPerUserUsedAtLeastOnceActiveDay": 9.18,
                    "usersUsedAtLeastOnce": 90,
                    "percentUsedAtLeastOnce": 31.03,
                    "usersNeverUsed": 200,
                    "percentNeverUsed": 68.97
                }
            ]
        }
    ]
  }
]
```


Histogram Report

This report returns data that is to be represented in chart format. The result consists of histogram-style data for each different event or event combination that has been requested. The results consist of 4 histograms based on different values: event counts, average event count per day, average event count per week, and average event count per month.

- Request/Response Parameters Summary
- Events Property
- lowerBounds and binUpperBounds Properties
- Results Summary
- Results Histograms

Request/Response Parameters Summary

```
POST /reporting/eventTracking/churn/lifetime/histogram
```

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 11-5 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. Required only for non-cookie authentication. • **sessionId (string)**—The `sessionId` obtained via POST /auth/login. Required only for non-cookie authentication. • **productId (integer)**—The product ID on which this request is being done • **startDate (string)**—The first date of the date range on which to base the report. This is to be formatted as YYYY-MM-DD. • **stopDate (string)**—The last date of the date range on which to base the report. This is to be formatted as YYYY-MM-DD. • **daysUntilDeclaredLost (integer)**—This specifies the number of consecutive days of inactivity that have to pass until a client installation is declared lost. • **dateReportedLost (string)**—When an installation is lost, it can either be shown as lost on the date it last contacted the Usage Intelligence servers (`dateLastSeen`) or else it can be shown as lost when it was declared lost (last date when it contacted Usage Intelligence + the number of days specified in `daysUntilDeclaredLost`) (`dateDeclaredLost`). Therefore, the permitted values are `dateLastSeen` and `dateDeclaredLost`. • **globalFilters (object)**—JSON object containing the filters to be applied to the available data. Details about these filters can be found in the Global Filters section. • **events (array)**—Array of objects specifying which events to include in the result. Supports both single events and event combinations. Details can be found in the Events property section. • **lowerBounds (object)**—Optional parameter to specify the lower bounds of each histogram. Must be used in conjunction with `binUpperBounds`. Details can be found in lowerBounds and binUpperBounds Properties. • **binUpperBounds (object)**—Optional parameter to specify the histogram bin bounds. Must be used in conjunction with `lowerBounds`. Details can be found in lowerBounds and binUpperBounds Properties.

Table 11-5 • Request Properties

Property	Description
Response JSON Object	• status (string)—Contains OK if successful or SYNTAX ERROR or AUTH ERROR. • reason (string)—Present only if status is not OK. Contains error message (reason). • **segmentBy (string)**—The same value that was passed as segmentBy in the request • summary (object) – Contains the number of clients who performed each event at least once and the number of clients which never performed each event. The summary data format is described in Results Histograms. • histograms (object) – Contains the histogram data as requested represented as a JSON object. The histogram data format is described in Results Histograms.

Events Property

The events property consists of an array of objects each representing a single event or a combination of events which are to be grouped together and represented as if they were a single event.

To specify a single event, the events array would contain an object similar to the following example:

```
{
     "category": "File Operations",
     "name":"Open"
}
```

To specify a combination of events, the events array would contain an object that looks like the following:

```
{
     "combiLabel": "Open and Save",
     "combiArray":  [
                       {
                         "categoryType": "string",
                         "nameType": "string",
                         "category": "File Operations",
                         "name": "Open"
                       },
                       {
                         "categoryType": "string",
                         "nameType": "string",
                         "category": "File Operations",
                         "name": "Save",
                       }
                    ]
}
```

In the above example, we are combining the data from the Open and Save events, both under the File Operations category. In this example, “string” is being used both as categoryType and nameType. The other possible value is “regex”. The following example shows a case where all events under the File Operations category are being combined:

```
{
     "combiLabel": "All File Operations",
     "combiArray":  [
                       {
```

```
                        "categoryType": "string",
                        "nameType": "regex",
                        "category": "File Operations",
                        "name": ".*"
                    }
                ]
}
```

In order to allow more advanced reporting, it is also possible to apply different filters for each event using a property named `eventFilters`. This way, it is possible to compare usage of a property or group of properties between different user groups. The format for `eventFilters` is exactly the same as `globalFilters`.

In the following example, we are comparing how the clients that are on version 1.1 have used the “Open” event vs. those clients that are on version 2.0:

```
[
    {
        "category": "File Operations",
        "name":"Open",
        "eventFilters":
            {
                "prodVersion":
                    {
                        "type": "string",
                        "value": "1.1"
                    }
            }
    },
    {
        "category": "File Operations",
        "name":"Open",
        "eventFilters":
            {
                "prodVersion":
                    {
                        "type": "string",
                        "value": "2.0"
                    }
            }
    }
]
```

It is also possible to apply different filters to different events within the same `combiArray`. Note that the use cases for this kind of filtering are rather limited, and in most cases, this method of filtering is not advised unless you have very specific requirements. In the following example, we are requesting the “Open” event for version 1.1 combined with the “Save” event for version 2.0:

```
{
     "combiLabel": "Open and Save in v1.1 and v2.0 respectively",
     "combiArray":  [
                        {
                        "categoryType": "string",
                        "nameType": "string",
                        "category": "File Operations",
                        "name": "Open",
                        "eventFilters":
                            {
```

```
                    "prodVersion":
                        {
                            "type": "string",
                            "value": "1.1"
                        }
                }
            },
            {
              "categoryType": "string",
              "nameType": "string",
              "category": "File Operations",
              "name": "Save",
              "eventFilters":
                  {
                      "prodVersion":
                          {
                              "type": "string",
                              "value": "2.0"
                          }
                  }
              }
          ]
}
```

lowerBounds and binUpperBounds Properties

By default, this report generates the histogram bins (ranges) automatically using a proprietary algorithm to reduce the effect of outliers. The `lowerBounds` and `binUpperBounds` are optional properties to allow the user to manually specify the bins instead of using this algorithm.

Both properties consist of a JSON object which contains 4 properties - `eventCount`, `averageCountPerActiveDay`, `averageCountPerActiveWeek`, and `averageCountPerActiveMonth`.

The `lowerBounds` object contains the lowest boundary in the leftmost bin in the histogram. The expected values inside the `lowerBounds` object are numeric. The `eventCount` value is an integer greater or equal to 1, while the other values are floating point numbers greater or equal to 0. Floating point numbers are expected to contain up to 2 decimal places, otherwise they are rounded.

The following is an example `lowerBounds` object:

```
{
    "eventCount": 1,
    "averageCountPerActiveDay": 0.1,
    "averageCountPerActiveWeek": 0.1,
    "averageCountPerActiveMonth": 0.1
}
```

The `binUpperBounds` property contains the upper boundaries of each bin in the histogram. The expected values inside the `binUpperBounds` object are arrays containing numeric values or a string containing “inf”. Each boundary must be higher than the one before it, and the first boundary must be greater than or equal to its corresponding value in `lowerBounds`. Similar to `lowerBounds`, the `eventCount` values are expected to be integers, while other values are expected to be floating point numbers with up to 2 decimal places.

The following is an example of the `binUpperBounds` object:

```
{
```

```
    "eventCount": [1, 2, 4, 8, 16, 32, "inf"],
    "averageCountPerActiveDay": [0.1, 0.2, 0.4, 0.6, 0.8, 1, 2, 4, 6, 10, 50, 100, "inf"],
    "averageCountPerActiveWeek": [0.1, 0.2, 0.4, 0.6, 0.8, 1, 2, 4, 6, 10, 50, 100, "inf"],
    "averageCountPerActiveMonth": [0.1, 0.2, 0.4, 0.6, 0.8, 1, 2, 4, 6, 10, 50, 100, "inf"]
}
```

Results Summary

The summary section is intended to show how many clients performed each event vs. the number of clients which never performed each event.

```
{
    "usedAtLeastOnce": [{
        "combiLabel": "Save or Delete",
        "value": 5
    }, {
        "category": "File Menu",
        "name": "Open",
        "value": 20
    }],
    "neverUsed": [{
        "combiLabel": "Save or Delete",
        "value": 37
    }, {
        "category": "File Menu",
        "name": "Open",
        "value": 112
    }]
}
```

Results Histograms

The histograms section contains the actual histogram results as requested.

```
{
    "eventCount":
      {
        "1 - 2":
          [
            {
                "combiLabel": "Save or Delete",
                "value": 10
            },
            {
                "category": "File Menu",
                "name": "Open",
                "value": 10
            }
          ],
        "3 - 5":
          [
            {
                "combiLabel": "Save or Delete",
                "value": 10
```

```
        },
        {
            "category": "File Menu",
            "name": "Open",
            "value": 10
        }
      ],
    "6 - 8":
      [
        {
            "combiLabel": "Save or Delete",
            "value": 1
        },
        {
            "category": "File Menu",
            "name": "Open",
            "value": 1
        }
      ],
    "9 - 11":
      [
        {
            "combiLabel": "Save or Delete",
            "value": 0
        },
        {
            "category": "File Menu",
            "name": "Open",
            "value": 0
        }
      ]
  },
"averageCountPerActiveDay":
  {
    "0.0 - 0.1":
      [
        {
            "combiLabel": "Save or Delete",
            "value": 0
        },
        {
            "category": "File Menu",
            "name": "Open",
            "value": 0
        }
      ],
    "0.11 - 0.2":
      [
        {
            "combiLabel": "Save or Delete",
            "value": 4
        },
        {
            "category": "File Menu",
            "name": "Open",
            "value": 4
```

```
        }
      ],
    "0.21 - 3.0":
      [
        {
            "combiLabel": "Save or Delete",
            "value": 20
        },
        {
            "category": "File Menu",
            "name": "Open",
            "value": 20
        }
      ],
    "3.01 - 4.0":
      [
        {
            "combiLabel": "Save or Delete",
            "value": 1
        },
        {
            "category": "File Menu",
            "name": "Open",
            "value": 1
        }
      ],
    "4.01 - 5.0":
      [
        {
            "combiLabel": "Save or Delete",
            "value": 0
        },
        {
            "category": "File Menu",
            "name": "Open",
            "value": 0
        }
      ]
  },
"averageCountPerActiveWeek":
  {
    "0.5 - 1.0":
      [
        {
            "combiLabel": "Save or Delete",
            "value": 18
        },
        {
            "category": "File Menu",
            "name": "Open",
            "value": 18
        }
      ],
    "1.01 - 2.0":
      [
        {
```

```
                "combiLabel": "Save or Delete",
                "value": 2
            },
            {
                "category": "File Menu",
                "name": "Open",
                "value": 2
            }
        ],
      "2.01 - 3.0":
        [
          {
                "combiLabel": "Save or Delete",
                "value": 3
            },
            {
                "category": "File Menu",
                "name": "Open",
                "value": 3
            }
        ],
      "3.00 - 4.0":
        [
          {
                "combiLabel": "Save or Delete",
                "value": 2
            },
            {
                "category": "File Menu",
                "name": "Open",
                "value": 2
            }
        ],
      "4.01 - 5.0":
        [
          {
                "combiLabel": "Save or Delete",
                "value": 0
            },
            {
                "category": "File Menu",
                "name": "Open",
                "value": 0
            }
        ]
    },
  "averageCountPerActiveMonth":
    {
      "0.7 - 1.0":
        [
          {
                "combiLabel": "Save or Delete",
                "value": 16
            },
            {
                "category": "File Menu",
```

```
                    "name": "Open",
                    "value": 16
                }
              ],
            "1.01 - 3.0":
              [
                {
                    "combiLabel": "Save or Delete",
                    "value": 3
                },
                {
                    "category": "File Menu",
                    "name": "Open",
                    "value": 3
                }
              ],
            "3.01 - 5.0":
              [
                {
                    "combiLabel": "Save or Delete",
                    "value": 4
                },
                {
                    "category": "File Menu",
                    "name": "Open",
                    "value": 4
                }
              ],
            "6.01 - 7.0":
              [
                {
                    "combiLabel": "Save or Delete",
                    "value": 1
                },
                {
                    "category": "File Menu",
                    "name": "Open",
                    "value": 1
                }
              ]
        }
}
```

12

License Key Registry Management

This section explains how to retrieve and search license keys from the key registry, and how to update and insert new keys in the key registry.

- Retrieving and Searching License Keys from the Key Registry
- Updating and Inserting New Keys in the Key Registry

Retrieving and Searching License Keys from the Key Registry

This request is used in order to get a list of license keys known by the Usage Intelligence servers along with their details. This is also used to search the key registry for a particular key or set of keys either by the key value itself or by another property such as by the expiry date or the assigned license type.

- Request/Response Parameters Summary
- Example Request
- Example Response

Request/Response Parameters Summary

`POST /licenseKeys/listKeys`

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

- Request JSON Object Properties
- Request Headers. Response Headers, and Status Codes Properties
- Response JSON Object Properties

Request JSON Object Properties

The following is a summary of the properties inside the request JSON object.

Table 12-1 • Request Properties

Property	Description	
user (string)	The username of your Usage Intelligence user account. Required only for non-cookie authentication.	
sessionId (string)	The sessionId obtained via POST /auth/login. Required only for non-cookie authentication.	
productId (integer)	The product ID on which this request is being done	
filters (object)	Optional JSON object to apply filters to retrieve subset of the keys on the server. Can contain any number of the following members:	
	licenseKey (object)	Filter by license key • **type (string)**—The data type of the value. Can be string, stringArray, or regex • **value (string/array)**—Contains either a string with the full license key, a list of strings with full license keys, or a string containing a regular expression.
	addedDate (object)	Filter based on the date added to the key registry • **type (string)**—The data type of the filter. Can be date or dateRange. • **value (string)**—To be used only if type is date. Should contain a single date formatted as YYYY-MM-DD • **min (string)**—To be used if type is dateRange. Should contain the minimum date formatted as YYYY-MM-DD. Can be used by itself or in conjuction with max. • **max (string)**—To be used if type is dateRange. Should contain the maximum date formatted as YYYY-MM-DD. Can be used by itself or in conjuction with min.
	expiryDate (object)	Filter based on the date the key is set to expire • **type (string)**—The data type of the filter. Can be date or dateRange • **value (string)**—To be used only if type is date. Should contain a single date formatted as YYYY-MM-DD. • **min (string)**—To be used if type is dateRange. Should contain the minimum date formatted as YYYY-MM-DD. Can be used by itself or in conjuction with max. • **max (string)**—To be used if type is dateRange. Should contain the maximum date formatted as YYYY-MM-DD. Can be used by itself or in conjunction with min.

Table 12-1 • Request Properties

Property	Description	
filters (object) (continued)	**autoCollected (boolean)**	Whether the keys to show were automatically collected by the SDK or not (uploaded to the server by the vendor)
	overQuotaCount (object)	Filter based on whether the key is oversubscribed and by how much. A negative number means that the key is running under-quota meaning that the user may still use it on more machines. The number 0 means that the quota has all been used but it hasn't been exceeded. Positive numbers mean that the key is over-subscribed. • **type (string)**—The data type of the filter. Can be number or numberRange. • **value (integer)**—To be used only if type is number. Should contain a single number. • **min (integer)**—To be used if type is numberRange. Should contain the minimum overQuota number to include. Can be used by itself or in conjuction with max. If max is present, the min value should be smaller than max. • **max (integer)**—To be used if type is numberRange. Should contain the maximum overQuota number to include. Can be used by itself or in conjuction with min. If min is present, the max value should be greater than min.
	licenseType (object)	Filter based on the license type assigned to the key • **type (string)**—The data type of the filter value. Can be string or stringArray • **value (string/array)**—Contains either a string with the license type, or a list of strings with license types to include
	licenseStatus (array)	Filter based on the license status flags assigned to the key. This filter contains an array of objects. Each object defines a set of flags and their respecive value. The flags in each object are "ANDed" together. If the array contains multiple objects, these objects are "ORed" together. In the example above, the keys have to be either "not activated" and "not expired" OR "activated" and "not blacklisted". Each object in the array can contain any number of the following members: • **activated (boolean)**—Whether the key is set to activated • **blacklisted (boolean)**—Whether the key is set to blacklisted • **expired (boolean)**—Whether the key is set to expired • **whitelisted (boolean)**—Whether the key is set to whitelisted

Table 12-1 • Request Properties

Property	Description
sorting (object)	Optional JSON object to apply sorting on the list of keys. Should contain the following members: • **field (string)**—The field by which to sort. Can be addedDate or overQuotaCount. Default is addedDate. • **order (string)**—The order by which to sort. Can be ascending or descending. Default is ascending.
paging (object)	Optional JSON object to specify the maximum number of keys to retrieve and how many to skip. Should contain the following members: • **startAt (integer)**—The number of keys to skip from the beginning. Default is 0. • **limit (integer)**—The maximum number of keys to include in the response. Default is 100. Maximum allowed is

Request Headers. Response Headers, and Status Codes Properties

The following is a summary of the properties.

Table 12-2 • Request Properties

Property	Description
Request Headers	• **Content-Type**—Can be set to `application/json` or `text/javascript` • **Accept**—Should be set to `text/javascript`
Response Headers	• **Content-Type**—Will contain `text/javascript`
Status Codes	• **200 OK**—OK (no error) • **400 Bad Request**—Malformed request • **403 Forbidden**—Wrong username, sessionId, session expired, or not authorized

Response JSON Object Properties

The following is a summary of the properties inside the response JSON object.

Table 12-3 • Request Properties

Property	Description
status (string)	Contains OK if successful or SYNTAX ERROR or AUTH ERROR
reason (string)	Present only if status is not OK. Contains error message (reason)
returnedKeysCount (integer)	Present only if status is OK. Contains the number of keys returned in this response

Table 12-3 • Request Properties

Property	Description	
matchingKeysCount (integer)	Present only if status is OK. Contains the total number of keys matching the supplied filters	
results (array)	Present only if status is OK. Contains a JSON object for each license key returned. Each object contains the following:	
	licenseKey (string)	The license key as supplied to the Usage Intelligence SDK
	licenseType (string)	The license type such as purchased, freeware, etc.
	licenseStatus (object)	A JSON object containing the 4 license status flags: • **activated (boolean)**—Whether the key is set to activated • **blacklisted (boolean)**—Whether the key is set to blacklisted • **expired (boolean)**—Whether the key is set to expired • **whitelisted (boolean)**—Whether the key is set to whitelisted
	expiryDate (string/null)	The date the key is set to expire formatted as YYYY-MM-DD. If an expiry date is not set, this value contains null.
	installQuota (integer)	The maximum number of installations allowed to use this key
	installCount (integer)	The number of installations using this key
	overQuotaCount (integer)	This field is computed by subtracting the installQuota from the installCount. If a key is not being over-used, this number will be a negative number or 0. If it is running over-quota, it will show a positive number.
	addedDate (string)	The date and time when this key was added to the key registry formatted as YYYY-MM-DDThh:mm:ss
	autoCollected (boolean)	Whether the key was automatically collected by the SDK or not (uploaded to the server by the vendor)

Table 12-3 • Request Properties

Property	Description	
results (array) (continued)	**lastEditedDate (string/ null)**	The date and time when this key was last edited formatted as YYYY-MM-DDThh:mm:ss. If this key has been auto-collected and never edited, this field will contain a null value.
	lastEditedBy (object/ null)	A JSON object containing the details about the user who last edited this key. If this key has been auto-collected and never edited, this field will contain a null value. If the user who last edited this key was removed from Usage Intelligence after requesting account deletion, this will contain an empty JSON object. • **name (string)**—The name of the user who last edited this key • **surname (string)**—The surname of the user who last edited this key • **email (string)**—The authentication email address of the user who last edited this key
	notes (string)	A string value used to store reference notes about this key.

Example Request

```
POST /licenseKeys/listKeys  HTTP/1.1
Host: api.revulytics.com
Content-Type: application/json
Accept: application/json

{
    "user": "testuser@test.com",
    "sessionId": "VSB8E2BzSC2eZSJm4QmTpA",
    "productId": 12345678901,
    "filters": {
        "overQuotaCount": {
            "type": "numberRange",
            "min": 1
        },
        "addedDate": {
            "type": "dateRange",
            "min": "2017-06-21"
        },
        "licenseStatus": [
            {
                "activated": false,
                "expired": false
            },
            {
                "activated": true,
                "blacklisted": false
            }
```

```
        ]
    },
    "sorting": {
        "field": "overQuotaCount",
        "order": "descending"
    },
    "paging": {
        "startAt": 20,
        "limit": 10
    }
}
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "status": "OK",
  "returnedKeysCount": 2,
  "matchingKeysCount": 22,
  "results": [
                {
                    "licenseKey": "14e7821b-694d-4d2d-90c2-adb10c07df0c",
                    "licenseType": "purchased",
                    "licenseStatus": {
                        "activated": false,
                        "blacklisted": true,
                        "expired": false,
                        "whitelisted": false
                    },
                    "expiryDate": "2018-05-17",
                    "installQuota": 2,
                    "installCount": 27,
                    "overQuotaCount": 25,
                    "addedDate": "2017-07-24T08:18:28",
                    "autoCollected": false,
                    "lastEditedDate": "2017-07-24T08:10:19",
                    "lastEditedBy": {
                        "name": "Revulytics",
                        "surname": "Demonstration",
                        "email": "demo@revulytics.com"
                    }
                },
                {
                    "licenseKey": "70cda86c-46c3-4399-b358-9e762ed1fcf5",
                    "licenseType": "unknown",
                    "licenseStatus": {
                        "activated": false,
                        "blacklisted": true,
                        "expired": false,
                        "whitelisted": false
                    },
                    "expiryDate": null,
                    "installQuota": 1,
                    "installCount": 501,
```

```
                    "overQuotaCount": 500,
                    "addedDate": "2017-07-24T08:18:30",
                    "autoCollected": true,
                    "lastEditedDate": "2017-07-24T08:10:19",
                    "lastEditedBy": {
                        "name": "Revulytics",
                        "surname": "Demonstration",
                        "email": "demo@revulytics.com"
                    }
                }
            ]
}
```


Updating and Inserting New Keys in the Key Registry

This request is used in order to update existing keys or to insert new keys into the key registry. Keys can be updated one by one or in batch. A single batch can contain both new and existing keys to be inserted or updated accordingly.

- Request/Response Parameters Summary
- Example Request
- Example Response

Request/Response Parameters Summary

`POST /licenseKeys/updateKeys`

Request JSON Object

The following is a summary of the properties inside the response JSON object.

Table 12-4 • Request Properties

Property	Description
user (string)	The username of your Usage Intelligence user account.
sessionId (string)	The sessionId obtained via POST /auth/login.
productId (integer)	The product ID on which this request is being done

Table 12-4 • Request Properties

Property	Description	
keyUpdates (array)	Array containing a number of JSON objects each containing a license key and the fields to be updated. The license key is the only compulsory value. All the rest are optional as you may send a subset of the fields. If a license key is already present in the key registry, the values that are not sent in this object are left untouched in the key registry. If the key is not present and an insert is taking place, default values are applied to the missing fields. The object members are as follows:	
	licenseKey (string)	The license key to be updated or inserted to the key registry. This value is compulsory.
	licenseType (string)	The license type such as purchased, freeware, etc.. Default is unknown.
	licenseStatus (object)	A JSON object containing the 4 license status flags or a subset: • **activated (boolean)**—Whether the key is to be set as activated. Default is false. • **blacklisted (boolean)**—Whether the key is to be set as blacklisted. Default is false. • **expired (boolean)**—Whether the key is to be set as expired. Default is false. • **whitelisted (boolean)**—Whether the key is to be set as whitelisted. Default is false.
	expiryDate (string/null)	The date the key is set to expire formatted as YYYY-MM-DD. A null value may also be sent to set this field as *null. Default is null.
	installQuota (integer)	The maximum number of installations allowed to use this key. Default is *1.
	notes (string)	A string value used to store reference notes about this key. Default is a zero-length (empty) string.

Example Request

```
POST /licenseKeys/updateKeys  HTTP/1.1
Host: api.revulytics.com
Content-Type: application/json
Accept: application/json

{
    "user": "testuser@test.com",
    "sessionId": "VSB8E2BzSC2eZSJm4QmTpA",
    "productId": 12345678901,
    "keyUpdates": [
        {
            "licenseKey": "70cda86c-46c3-4399-b358-9e762ed1fcf5",
```

```
            "licenseType": "purchased"
        },
        {
            "licenseKey": "14e7821b-694d-4d2d-90c2-adb10c07df0c",
            "licenseStatus": {
                "whitelisted": true,
                "activated": true,
                "expired": false
            },
            "expiryDate": "2017-10-31"
        }
    ]
}
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "status": "OK",
  "insertedKeys": 0,
  "rejectedKeys": 0,
  "updatedKeys": 2
}
```

13

Custom Event Tracking

Custom Event Tracking works in a similar way to Exception Tracking. In both cases, data can either be previewed by retrieving the latest data in JSON format or else, zipped CSV files can be downloaded for offline processing.

- Latest Data Preview
- Downloadable File Listing
- Download Zipped CSV File

Latest Data Preview

This request returns the last few custom events that have been collected including the product and system metadata of the client on which each event was collected in JSON format.

- Request/Response Parameters Summary
- Results Format

Request/Response Parameters Summary

```
POST /customEventTracking/getLastLines
```

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 13-1 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. Required only for non-cookie authentication. • **sessionId (string)**—The sessionId obtained via POST /auth/login. Required only for non-cookie authentication. • **productId (integer)**—The product ID on which this request is being done. • **lineCount (string)**—The number of lines/events to retrieve.
Response JSON Object	• **status (string)**—Contains OK if successful or SYNTAX ERROR or AUTH ERROR. • **reason (string)**—Present only if status is not OK. Contains error message (reason). • **fieldNames (object)**—Maps the key names in the result object with a user friendly name • **results (array)**—Contains the results as requested represented as a JSON array. The result format is described below.

Results Format

The results element is an array containing the actual results. Each event is presented as a JSON object inside the results array. This is the same data contained in the downloadable zipped CSV files.

In the following example, the last 2 lines in the CSV are being requested (the lineCount property is set to 2)

Example Request

```
POST /customEventTracking/getLastLines  HTTP/1.1
Host: api.revulytics.com
Content-Type: application/json
Accept: application/json

{
    "user": "testuser@test.com",
    "sessionId": "VSB8E2BzSC2eZSJm4QmTpA",
    "productId": 2376158762,
    "lineCount": 2
}
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
   "status":"OK",
```

```
"fieldNames":{
   "timestamp":"Timestamp",
   "clientId":"Client ID",
   "prodEdition":"Edition",
   "prodVersion":"Version",
   "prodBuild":"Build",
   "prodLanguage":"Language",
   "licenseType":"License Type",
   "licenseStatus.whitelisted":"Key Whitelisted",
   "licenseStatus.blacklisted":"Key Blacklisted",
   "licenseStatus.activated":"Key Activated",
   "licenseStatus.expired":"Key Expired",
   "os.version":"OS Type",
   "osWordLength":"OS Architecture",
   "osLanguage":"OS Language",
   "cpuType":"CPU Type",
   "cpuCores":"CPU Cores",
   "ram":"RAM",
   "displayCount":"Number of Monitors",
   "resolution":"Screen Resolution",
   "dotNetVersion": ".NET Versions",
   "gpu.model": "GPU",
   "formFactor":"Computer Type",
   "geography.country":"Country",
   "geography.usState":"US State",
   "event.category":"Event Category",
   "event.name":"Event Type",
   "event.data":"Event Data"
},
"results":[
   {
      "timestamp":"2017-11-22T12:41:19",
      "clientId":"AE99797E4B480921",
      "prodEdition":"Standard",
      "prodVersion":"3.1",
      "prodBuild":"951",
      "prodLanguage":"German",
      "licenseType":"purchased",
      "licenseStatus.whitelisted":"N",
      "licenseStatus.blacklisted":"N",
      "licenseStatus.activated":"N",
      "licenseStatus.expired":"N",
      "os.version":"MS Win XP",
      "osWordLength":"64-bit",
      "osLanguage":"German",
      "cpuType":"Intel Core i7",
      "cpuCores":"4",
      "ram":"8192",
      "displayCount":"1",
      "resolution":"1920x1080",
      "dotNetVersion": "2.0;3.5 SP1;3.0 SP2",
      "gpu.model": "Intel HD Graphics 620",
      "formFactor":"Desktop",
      "geography.country":"DE",
      "geography.usState":"",
      "event.category":"File Operations",
```

```
            "event.name":"Save",
            "event.data":"final21.doc"
         },
         {
            "timestamp":"2017-11-22T14:51:54",
            "clientId":"2958A7C5911ABF77",
            "prodEdition":"Premium",
            "prodVersion":"2",
            "prodBuild":"719",
            "prodLanguage":"English",
            "licenseType":"purchased",
            "licenseStatus.whitelisted":"N",
            "licenseStatus.blacklisted":"N",
            "licenseStatus.activated":"N",
            "licenseStatus.expired":"N",
            "os.version":"MS Win Vista",
            "osWordLength":"64-bit",
            "osLanguage":"English",
            "cpuType":"Intel Pentium",
            "cpuCores":"2",
            "ram":"1024",
            "displayCount":"1",
            "resolution":"1920x1080",
            "dotNetVersion": "2.0;3.5 SP1;3.0 SP2",
            "gpu.model": "Intel HD Graphics 620",
            "formFactor":"Desktop",
            "geography.country":"US",
            "geography.usState":"IL",
            "event.category":"File Operations",
            "event.name":"Open",
            "event.data":"testfl.doc"
         }
      ]
   }
```


Downloadable File Listing

This request returns the list of zipped CSV files that are on the server. This list is to be used to see what files are available for download and then be able to request files for downloading.

- Request/Response Parameters Summary
- Example Request/Response

Request/Response Parameters Summary

```
POST /customEventTracking/listFiles
```

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 13-2 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. • **sessionId (string)**—The sessionId obtained via POST /auth/login. • **productId (integer)**—The product ID on which this request is being done.
Response JSON Object	• status (string)—Contains OK if successful or SYNTAX ERROR or AUTH ERROR. • reason (string)—Present only if status is not OK. Contains error message (reason). • **openDailyFiles (array)**—Array of JSON objects containing information about daily files that are not finalized yet. This normally contains only 1 element which refers to today's file. This file is still being used to collect data, and therefore, if it is downloaded now, the contents would be different than if it is downloaded later, even though the file name is the same. The objects in this array do not contain a `compressedSizeKB` element since this is constantly changing. Also, the objects contain a boolean property named `isAvailable`. If this file is currently being processed by the system, it may be unavailable for download, and therefore this value would be false. • **dailyFiles (array)**—Array of JSON objects containing information about daily files that are finalized. These files can be requested for download. • **monthlyFiles (array)**—Array of JSON objects containing information about monthly files. These files can be requested for download.

Example Request/Response

This section includes and example request and an example response.

Example Request

```
POST /customEventTracking/listFiles  HTTP/1.1
Host: api.revulytics.com
Content-Type: application/json
Accept: application/json

{
    "user": "testuser@test.com",
    "sessionId": "VSB8E2BzSC2eZSJm4QmTpA",
    "productId": 2376158762
}
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
   "status":"OK",
   "openDailyFiles":[
      {
         "fileName":"2376158762_cust_2017-11-24.zip",
         "uncompressedSizeKB":1.317,
         "fileDate":"2017-11-24",
         "isAvailable":true
      }
   ],
   "dailyFiles":[
      {
         "fileName":"2376158762_cust_2017-11-22.zip",
         "uncompressedSizeKB":3.055,
         "compressedSizeKB":1.018,
         "fileDate":"2017-11-22"
      },
      {
         "fileName":"2376158762_cust_2017-11-23.zip",
         "uncompressedSizeKB":2.129,
         "compressedSizeKB":0.881,
         "fileDate":"2017-11-23"
      }
   ],
   "monthlyFiles":[
      {
         "fileName":"2376158762_cust_2017-11-01_to_2017-11-23.zip",
         "uncompressedSizeKB":4.897,
         "compressedSizeKB":1.273,
         "fileStartDate":"2017-11-01",
         "fileStopDate":"2017-11-23"
      }
   ]
}
```


Download Zipped CSV File

In order to download the data files, the file needs to be requested to the API which returns a secure URL from which the file can be downloaded.

```
POST /customEventTracking/getDownloadUrl
```

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 13-3 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. • **sessionId (string)**—The sessionId obtained via POST /auth/login. • **productId (integer)**—The product ID on which this request is being done. • **fileName (string)**—The filename of the file being requested to download
Response JSON Object	• status (string)—Contains OK if successful or SYNTAX ERROR or AUTH ERROR. • reason (string)—Present only if status is not OK. Contains error message (reason). • **downloadUrl (string)**—The one-time URL used to download the zip file

Example Request/Response

Example Request

```
POST /customEventTracking/getDownloadUrl  HTTP/1.1
Host: api.revulytics.com
Content-Type: application/json
Accept: application/json

{
    "user": "testuser@test.com",
    "sessionId": "VSB8E2BzSC2eZSJm4QmTpA",
    "productId": 2376158762,
    "fileName": "2376158762_cust_2017-11-23.zip"
}
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
   "status":"OK",
   "downloadUrl":"https://sh1.revulytics.com/downloadExc/1ij4buyJtsTrx"
}
```

14

Exception Tracking

Exception Tracking works in a similar way to Custom Event Tracking. In both cases, data can either be previewed by retrieving the latest data in JSON format or else, zipped CSV files can be downloaded for offline processing.

- Latest Data Preview
- Downloadable File Listing
- Download Zipped CSV File

Latest Data Preview

This request returns the last few exceptions that have been collected including the product and system metadata of the client on which each exception was collected in JSON format.

- Request/Response Parameters Summary
- Results Format

Request/Response Parameters Summary

```
POST /exceptionTracking/getLastLines
```

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 14-1 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. Required only for non-cookie authentication. • **sessionId (string)**—The sessionId obtained via POST /auth/login. Required only for non-cookie authentication. • **productId (integer)**—The product ID on which this request is being done. • **lineCount (string)**—The number of lines/events to retrieve.
Response JSON Object	• status (string)—Contains OK if successful or SYNTAX ERROR or AUTH ERROR. • reason (string)—Present only if status is not OK. Contains error message (reason). • **fieldNames (object)**—Maps the key names in the result object with a user friendly name • **results (array)**—Contains the results as requested represented as a JSON array. The result format is described below.

Results Format

The results element is an array containing the actual results. Each exception is presented as a JSON object inside the results array. This is the same data contained in the downloadable zipped CSV files.

In the following example, the last 2 lines in the CSV are being requested (the lineCount property is set to 2)

Example Request

```
POST /exceptionTracking/getLastLines  HTTP/1.1
Host: api.revulytics.com
Content-Type: application/json
Accept: application/json

{
    "user": "testuser@test.com",
    "sessionId": "VSB8E2BzSC2eZSJm4QmTpA",
    "productId": 2376158762,
    "lineCount": 2
}
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
   "status":"OK",
```

```
"fieldNames":{
   "timestamp":"Timestamp",
   "clientId":"Client ID",
   "prodEdition":"Edition",
   "prodVersion":"Version",
   "prodBuild":"Build",
   "prodLanguage":"Language",
   "licenseType":"License Type",
   "licenseStatus.whitelisted":"Key Whitelisted",
   "licenseStatus.blacklisted":"Key Blacklisted",
   "licenseStatus.activated":"Key Activated",
   "licenseStatus.expired":"Key Expired",
   "os.version":"OS Type",
   "osWordLength":"OS Architecture",
   "osLanguage":"OS Language",
   "cpuType":"CPU Type",
   "cpuCores":"CPU Cores",
   "ram":"RAM",
   "displayCount":"Number of Monitors",
   "resolution":"Screen Resolution",
   "dotNetVersion": ".NET Versions",
   "gpu.model": "GPU",
   "formFactor":"Computer Type",
   "geography.country":"Country",
   "geography.usState":"US State",
   "exception.class":"Exception Class",
   "exception.method":"Exception Method",
   "exception.message":"Exception Message",
   "exception.stackTrace":"Exception Stack Trace"
},
"results":[
   {
      "timestamp":"2017-11-22T01:38:07",
      "clientId":"A966094C61A6C181",
      "prodEdition":"Standard",
      "prodVersion":"2",
      "prodBuild":"719",
      "prodLanguage":"English",
      "licenseType":"purchased",
      "licenseStatus.whitelisted":"N",
      "licenseStatus.blacklisted":"N",
      "licenseStatus.activated":"N",
      "licenseStatus.expired":"N",
      "os.version":"MS Win XP",
      "osWordLength":"64-bit",
      "osLanguage":"English",
      "cpuType":"Intel Core i7",
      "cpuCores":"4",
      "ram":"2048",
      "displayCount":"1",
      "resolution":"1280x1024",
      "dotNetVersion": "2.0;3.5 SP1;3.0 SP2",
      "gpu.model": "Intel HD Graphics 620",
      "formFactor":"Desktop",
      "geography.country":"FR",
      "geography.usState":"",
```

```
            "exception.class":"File",
            "exception.method":"openFile",
            "exception.message":"No such file or directory",
            "exception.stackTrace":"Traceback (most recent call last):\r\n  File \"main.py\", line 1, in
<module>\r\nIOError: [Errno 2] No such file or directory: 'full-template.doc'"
         },
         {
            "timestamp":"2017-11-22T01:38:07",
            "clientId":"A966094C61A6C181",
            "prodEdition":"Standard",
            "prodVersion":"2",
            "prodBuild":"719",
            "prodLanguage":"English",
            "licenseType":"purchased",
            "licenseStatus.whitelisted":"N",
            "licenseStatus.blacklisted":"N",
            "licenseStatus.activated":"N",
            "licenseStatus.expired":"N",
            "os.version":"MS Win XP",
            "osWordLength":"64-bit",
            "osLanguage":"English",
            "cpuType":"Intel Core i7",
            "cpuCores":"4",
            "ram":"2048",
            "displayCount":"1",
            "resolution":"1280x1024",
            "dotNetVersion": "2.0;3.5 SP1;3.0 SP2",
            "gpu.model": "Intel HD Graphics 620",
            "formFactor":"Desktop",
            "geography.country":"FR",
            "geography.usState":"",
            "exception.class":"File",
            "exception.method":"openFile",
            "exception.message":"No such file or directory",
            "exception.stackTrace":"Traceback (most recent call last):\r\n  File \"main.py\", line 1, in
<module>\r\nIOError: [Errno 2] No such file or directory: 'full-template.doc'"
         }
      ]
}
```


Downloadable File Listing

This request returns the list of zipped CSV files that are on the server. This list is to be used to see what files are available for download and then be able to request files for downloading.

- Request/Response Parameters Summary
- Example Request/Response

Request/Response Parameters Summary

```
POST /exceptionTracking/listFiles
```

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 14-2 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. • **sessionId (string)**—The sessionId obtained via POST /auth/login. • **productId (integer)**—The product ID on which this request is being done.
Response JSON Object	• status (string)—Contains OK if successful or SYNTAX ERROR or AUTH ERROR. • reason (string)—Present only if status is not OK. Contains error message (reason). • **openDailyFiles (array)**—Array of JSON objects containing information about daily files that are not finalized yet. This normally contains only 1 element which refers to today's file. This file is still being used to collect data, and therefore, if it is downloaded now, the contents would be different than if it is downloaded later, even though the file name is the same. The objects in this array do not contain a `compressedSizeKB` element since this is constantly changing. Also, the objects contain a boolean property named `isAvailable`. If this file is currently being processed by the system, it may be unavailable for download, and therefore this value would be false. • **dailyFiles (array)**—Array of JSON objects containing information about daily files that are finalized. These files can be requested for download. • **monthlyFiles (array)**—Array of JSON objects containing information about monthly files. These files can be requested for download.

Example Request/Response

This section includes an example request and an example response.

Example Request

```
POST /exceptionTracking/listFiles  HTTP/1.1
Host: api.revulytics.com
Content-Type: application/json
Accept: application/json

{
    "user": "testuser@test.com",
    "sessionId": "VSB8E2BzSC2eZSJm4QmTpA",
    "productId": 2376158762
}
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
   "status":"OK",
   "openDailyFiles":[
      {
         "fileName":"2376158762_exc_2017-11-24.zip",
         "uncompressedSizeKB":2.105,
         "fileDate":"2017-11-24",
         "isAvailable":true
      }
   ],
   "dailyFiles":[
      {
         "fileName":"2376158762_exc_2017-11-22.zip",
         "uncompressedSizeKB":3.673,
         "compressedSizeKB":1.072,
         "fileDate":"2017-11-22"
      },
      {
         "fileName":"2376158762_exc_2017-11-23.zip",
         "uncompressedSizeKB":3.587,
         "compressedSizeKB":1.081,
         "fileDate":"2017-11-23"
      }
   ],
   "monthlyFiles":[
      {
         "fileName":"2376158762_exc_2017-11-01_to_2017-11-23.zip",
         "uncompressedSizeKB":6.937,
         "compressedSizeKB":1.344,
         "fileStartDate":"2017-11-01",
         "fileStopDate":"2017-11-23"
      }
   ]
}
```


Download Zipped CSV File

In order to download the data files, the file needs to be requested to the API which returns a secure URL from which the file can be downloaded.

```
POST /exceptionTracking/getDownloadUrl
```

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 14-3 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. • **sessionId (string)**—The sessionId obtained via POST /auth/login. • **productId (integer)**—The product ID on which this request is being done. • **fileName (string)**—The filename of the file being requested to download
Response JSON Object	• status (string)—Contains OK if successful or SYNTAX ERROR or AUTH ERROR. • reason (string)—Present only if status is not OK. Contains error message (reason). • **downloadUrl (string)**—The one-time URL used to download the zip file

Example Request/Response

This section includes an example request and an example response.

Example Request

```
POST /exceptionTracking/getDownloadUrl  HTTP/1.1
Host: api.revulytics.com
Content-Type: application/json
Accept: application/json

{
    "user": "testuser@test.com",
    "sessionId": "VSB8E2BzSC2eZSJm4QmTpA",
    "productId": 2376158762,
    "fileName": "2376158762_exc_2017-11-23.zip"
}
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
   "status":"OK",
   "downloadUrl":"https://sh1.revulytics.com/downloadExc/1ij4buyJtsTrx"
}
```

15

Client Profile Report

The aim of this report is to retrieve a subset or all of the data about a client or a set of clients. This can also be used as a means to export all data about all clients by paging through all the data and storing the returned data for offline processing.

Request/Response Parameters Summary

```
POST /reporting/clientPropertyList
```

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 15-1 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. Required only for non-cookie authentication. • **sessionId (string)**—The sessionId obtained via POST /auth/login. Required only for non-cookie authentication. • **productId (integer)**—The product ID on which this request is being done. • **startAtClientId (string)**—Optional property to specify the client ID to start at for paging purposes. • **properties (array)**—Array of strings containing the list of properties to be included in the current data. • **retDailyData (object)**—Optional parameter used if the daily data is being requested. The format for this object is described below. • **globalFilters (object)**—Optional JSON object containing the filters to be applied to the available data. Details about these filters can be found in Global Filters.
Response JSON Object	• status (string)—Contains OK if successful or SYNTAX ERROR or AUTH ERROR. • reason (string)—Present only if status is not OK. Contains error message (reason). • **results (array)**—Contains the results as requested represented as a JSON object. The result format is described below. • **reachedEnd (boolean)**—Boolean value showing whether the end has been reached or not. This value is false if there are remaining clients which can be retrieved by requesting more pages. • **nextClientId (string)**—Returned only if `reachedEnd` is false. This value can be passed to the `startAtClientId` property to request the next page.

retDailyData Property

The `retDailyData` property is to be used when daily data is being requested as part of the report. If this is not present, the only data that is returned is the last known values for each client being included. This property is expected to contain an object made up of the following properties:

- **startDate (string)**—The first day to include in the daily data. This is to be formatted as YYYY-MM-DD.
- **stopDate (string)**—The last day to include in the daily data. This is to be formatted as YYYY-MM-DD.
- **properties (array)**—Array of strings containing the list of properties to be included in the daily data.

properties Property

This property can be present both in the root request object and also in the `retDailyData` object. In both cases, this property is optional. If this property is not present in the root object, the only current property that is returned is the client ID. Similarly, if this property is not present in the `retDailyData` object, the daily data would contain only the dates on which each client was active without any further details about what the value of client data properties was on each day.

Current Data Properties

The following properties can be included in the current (root) properties array:

```
machineId
optOut.historical
optOut.current
backOff.historical
backOff.current
prodEdition
prodVersion
prodLanguage
prodBuild
licenseType
licenseStatus.activated
licenseStatus.blacklisted
licenseStatus.expired
licenseStatus.whitelisted
os.platform
os.version
os.edition
geography.continent
geography.country
geography.usState
cpuType
ram
resolutionWidth
resolutionHeight
osWordLength
displayCount
cpuCores
licenseKey
formFactor
dotNetVersion
lifetimeSessionCount
lifetimeRuntimeMinutes
gpu.vendor
gpu.model
javaVersion
javaVmVersion
javaVmName
javaVendor
javaRuntime
```

```
javaGraphics
vm
reachOutDeliveries.auto
reachOutDeliveries.manual
lifetimeEventUsage
```

Note • *licenseKey requires a special user permission to filter by license key.*

Daily Data Properties

The following properties can be included in the daily (inside `retDailyData`) properties array:

```
prodEdition
prodVersion
prodLanguage
prodBuild
licenseType
licenseStatus.activated
licenseStatus.blacklisted
licenseStatus.expired
licenseStatus.whitelisted
os.platform
os.version
os.edition
geography.continent
geography.country
geography.usState
```

Global Filters

Most of the available filter properties are string-based. This means that when applying a filter, the requested field can be represented as a `string`, `stringArray` or `regex`. There are also some filters which are numeric. These filters should be represented as `number` or `numberRange`.

- String-Based Filters
- Numeric Filters
- Date Range Filters
- Boolean Filters
- Special Filters
- <NULL> Values in Global Filters

String-Based Filters

The following properties are stored as strings:

```
machineId *
clientId *
```

```
prodVersion
prodEdition
prodBuild
prodLanguage
licenseType
formFactor *
osLanguage
osWordLength *
cpuType *
dotNetVersion *
javaVersion *
javaVendor *
javaRuntime *
javaGraphics *
javaVmVersion *
javaVmName *
vm *
C01 .. C20 (Custom properties)
licenseKey *
```

Note • *`licenseKey` requires a special user permission to be used as a filter.*

Note • *Properties marked with an asterisk (*) are based on the current (latest known) values.*

The **type** field in the above filters needs to be `string`, `stringArray` or `regex`. A **value** field is always required. The contents of this field should be according to the specified type.

- If `string` is specified, then the value field must contain a single string that needs to be matched precisely with the stored data.
- If `stringArray` is specified, then the value field must contain an array of strings where one of which needs to match precisely with the stored data.
- If specifying a `regex`, the value field should contain a string which is treated as a regular expression and the stored data will be matched against it using regular expression rules.

Example Filter Using a String Value

In this example, the product build value needs to be exactly “3014.int-12214”:

```
{
      "prodBuild":
          {
              "type": "string",
              "value": "3014.int-12214"
          }
}
```

Example Filter Using a String Array

In this example, the product build value needs to be either “3014.int-12214”, “3017.enx-57718”, or “4180.vrx-81059”. Note that since the type is declared as stringArray, the value field needs to contain an array. Consider all elements in the array to have an OR logical expression between them.:

```
{
    "prodBuild":
        {
            "type": "stringArray",
            "value": ["3014.int-12214", "3017.enx-57718", "4180.vrx-81059"]
        }
}
```

Example Filter Using a Regular Expression

In this example, the product build value needs to start with “30” and end with “18” whilst having 10 characters in between:

```
{
     "prodBuild":
         {
             "type": "regex",
             "value": "^30.{10}18$"
         }
}
```

Numeric Filters

The following properties are stored as numeric values:

```
cpuCores *
displayCount *
ram *
resolutionWidth *
resolutionHeight *
lifetimeRuntimeMinutes *
lifetimeSessionCount *
screenPpi *
javaVmRam *
```

Note • *Properties marked with an asterisk (*) are based on the current (latest known) values.*

The type field in the above filters needs to be number or numberRange.

- If number is specified, then a value field must also be present. The value field should contain a number, which may contain a decimal point if required.
- If numberRange is specified, then the value field should NOT be used. Instead, the properties min and max are to be used. These refer to the minimum and maximum number to be included in the report. If only one limit needs to be set, the other property is to be left out. Therefore, if you want to include installations with up to 2 display devices, you would not specify a min value, but instead specify only a max and set it as 2.

Example Filter Using a Number Value

In this example, the number of display devices needs to be exactly 3:

```
{
      "displayCount":
          {
              "type": "number",
```

```
            "value": 3
        }
}
```

Example Filter Using a Number Range Value

In this example, the RAM needs to be between 1025MB and 4096MB (both included):

```
{
      "ram":
          {
              "type": "numberRange",
              "min": 1025,
              "max": 4096
          }
}
```

Date Range Filters

The following properties are stored as dates:

`dateInstalled`
`dateLastSeen`

The type field in the above filters needs to be `date` or `dateRange`.

- If `date` is specified, then a value field must also be present. The value field should contain a date.
- If `dateRange` is specified, then the value field should NOT be used. Instead, the properties min and max are to be used. These refer to the minimum and maximum dates to be included in the report. If only one limit needs to be set, the other property is to be left out.

In the following example, users installed after January 1st 2018 are to be shown:

```
{
    "dateInstalled":
        {
            "type": "dateRange",
            "min": "2018-01-01"
        }
}
```

Note • *All dates must be in ISO 8601 format.*

Boolean Filters

The following property is stored as boolean:

`touchScreen`

The type field in the above filters needs to be boolean. The value must be true or false. In the following filter, clients with a touch screen are being requested.

```
{
```

```
    "touchScreen":
        {
            "value": true
        }
}
```

Special Filters

Some filters need to be represented in a special format due to their unique requirements. These special filters are:

- Special Filter: licenseStatus
- Special Filter: os
- Special Filter: geography
- Special Filter: gpu
- Special Filters: optOut and backOff
- Special Filter: lifetimeEventUsage
- Special Filter: reachOutDeliveries

Special Filter: licenseStatus

The `licenseStatus` filter is made up of 4 sub-values: `activated`, `blacklisted`, `expired` and `whitelisted`. These are presented as boolean values.

Unlike other filters, this filter is presented as an array of JSON objects. Each object can contain a subset (or all) of these 4 boolean values.

Consider the following example. In this example, for a client to be included, the license has to either be activated AND whitelisted, or else it can be not whitelisted AND expired. In other words, ((activated AND whitelisted) OR ((NOT)whitelisted AND expired)).

```
{
    "licenseStatus":
        [
            {
                "activated": true,
                "whitelisted": true
            },
            {
                "whitelisted": false,
                "expired": true
            }
        ]
}
```

Special Filter: os

The os filter is made up of 3 granularity levels. These are `platform`, `version`, and `edition`. These are meant to split the OS name into levels of detail as required by the user. Consider the following:

- **platform**: Microsoft Windows
- **version**: Microsoft Windows 7
- **edition**: Microsoft Windows 7 Professional

If a filter is set on the version “Microsoft Windows 7”, the result would include all editions of Windows 7. One or more of these granularity levels may be specified. If more than 1 granularity level is specified, the values are ORed together.

In the following example, all editions of “Microsoft Windows 7” are included, and also “Microsoft Windows Vista Home Premium”:

```
{
    "type": "string",
    "version": "Microsoft Windows 7",
    "edition": "Microsoft Windows Vista Home Premium"
}
```

In the following example, the type is `stringArray`. Note that an array needs to be passed if the type is set as such, even if it is to contain only 1 element. In this case, the version can be either “Microsoft Windows 7” or “Microsoft Windows 8” (which are ORed together). Also, clients running on “Microsoft Windows XP Professional” are to be included.

```
{
    "type": "stringArray",
    "version": ["Microsoft Windows 7", "Microsoft Windows 8"],
    "edition": ["Microsoft Windows XP Professional"]
}
```

Special Filter: geography

The geography filter is made up of 3 granularity levels. These are continent, country, and usState. The usState value applies only to United States. Continents and countries are presented in 2-letter codes. Countries follow ISO standard 3166-1 alpha-2. US states are presented in ISO 3166-2:US format.

In the following example, the clients have to be either:

- In the continents *Asia* or *Oceania*
- In the country *Germany*
- In the US states *New York*, *New Jersey*, or *Kansas*

```
{
    "type": "stringArray",
    "continent": ["AS", "OC"],
    "country": ["DE"],
    "usState": ["US-NY", "US-NJ", "US-KS"]
}
```

Important • *In this filter, the type can be `string` or `stringArray`. Regular expressions are not supported in geography filters.*

Special Filter: gpu

The gpu filter is made up of 2 granularity levels. These are `vendor` and `model`. Both are represented as string values.

In the following example, the clients have to have a GPU:

- From the vendors NVIDIA or Intel
- With the model AMD Radeon HD 4600

```
{
    "type": "stringArray",
    "vendor": ["NVIDIA", "Intel"],
    "model": ["AMD Radeon HD 4600"]
}
```

Special Filters: optOut and backOff

The opt-out mechanism was introduced in SDK version 5.1.0. With this feature, vendors can have their application report to the Usage Intelligence servers that a user does not want to be tracked. Using this property, vendors can filter out installations that were opted-out.

Similarly, backoff filtering was introduced with version 5.0.0. Backoff is when a product account runs over-quota and the server starts rejecting data. Although filtering for backed-off installations was introduced with version 5, it was also backported to previous versions. However, when a new installation with an SDK prior to version 5 tries to register with the server and is rejected, it is not marked as being once backed-off when it is eventually accepted by the server. With version 5 onwards, the server flags an installation as being historically backed-off in such cases.

Both `backOff` and `optOut` filters are made up of 2 boolean sub-values: `historical` and `current`.

- The `historical` value refers to installations that were once backed-off or opted-out. These may include installations that are still currently backed-off or opted-out.
- The `current` value refers to the status during the last time that the client called the server. Therefore, if an installation was opted-out yesterday but got opted-in today, it will be marked as historically opted-out but not currently opted-out.

In the following example, for a client to be included, the `optOut` status has to either be `historical` AND not `current`, or else it can be not `historical` (i.e. users have to be currently opted-in but used to be opted-out at some point or never opted out).

```
{
    "optOut":
        [
            {
                "historical": true,
                "current": false
            },
            {
                "historical": false
            }
        ]
}
```

Special Filter: lifetimeEventUsage

Using lifetime event usage filters, clients can be filtered based on whether a particular event or set of events occurred or not within the client's lifetime. Alternatively, one can set a filter based on the number of times an event has occurred.

In the following example, clients that are included must have done the “File Operations - Open” event at least 5 times to be counted.

```
{
     "category": "File Operations",
     "name": "Open",
     "min": 5
}
```

In the following example, clients must have done either “File Operations - Open” or “File Operations - Save” for a combined total of between 10 to 50 times.

```
{
    "combiArray": [
                    {
                      "categoryType": "string",
                      "nameType": "string",
                      "category": "File Operations",
                      "name": "Open"
                    },
                    {
                      "categoryType": "string",
                      "nameType": "string",
                      "category": "File Operations",
                      "name": "Save",
                    }
                ],
    "min": 10,
    "max": 50
}
```

In the following example, clients must have done any event within the “File Operations” category for a combined total of not more than 100 times. This is done using a regular expression in the name.

```
{
    "combiArray":  [
                     {
                       "categoryType": "string",
                       "nameType": "regex",
                       "category": "File Operations",
                       "name": ".*"
                     }
                  ],
    "max": 100
}
```

Special Filter: reachOutDeliveries

Using ReachOut delivery filters, clients can be filtered based on whether a particular ReachOut message or a combination of ReachOut messages were delivered or not within the client’s lifetime.

The filter consists of a JSON array that includes one or more objects. Each object is a combination of delivered and undelivered campaigns, and the different combinations are ORed together. Therefore, it is possible to show users that either received ReachOut message 1 but not 2, or else received 3 but not 4 as in the following example:

In the following example, we are looking for clients who either received campaign 1 but not 2, OR received campaign 2 but not 3.

```
[
     {"auto": {"delivered": ["1"], "undelivered":["2"]}},
     {"auto": {"delivered": ["2"], "undelivered":["3"]}}
]
```

The above example contains only “auto” ReachOut campaigns. Manual campaigns can be specified using “manual” instead of “auto” as in the above example. Each object can contain a mix of “auto” and “manual” campaigns.

<NULL> Values in Global Filters

Most of the available properties can include null values. There are different reasons why a value would be null. When these are properties that are set by the application, the possible reasons why a value would be null are cases where the value has not been set by the application (such as `prodBuild` never being set), and cases where values are set to an empty string (“”) or to a string containing “<NULL>”.

One other reason is that although these values have been set, the SDK has not yet had time to sync with the servers to provide this new information. In cases where the properties are set automatically such as hardware or OS related information, the values would be null if the SDK failed to retrieve that value from the OS or if the server failed to identify the value retrieved by the SDK.

Other reasons include cases where Java version is requested from an application that does not use the Java SDK, US state is requested for users who are not running within the US, etc.

The following are the properties that support null values:

```
prodVersion
prodEdition
prodBuild
prodLanguage
machineId
formFactor
vm
cpuType
cpuCores
ram
resolutionWidth
resolutionHeight
javaVersion
javaVmVersion
javaVmName
javaVendor
javaRuntime
javaGraphics
osLanguage
licenseKey
C01 .. C20 (Custom properties)
os
geography
gpu
```

Null values can be requested either on their own or as part of a filter containing other values.

The following example would return only cases where the `prodVersion` is null:

```
{
      "prodVersion":
          {
              "includeNull": true
          }
}
```

The following example would return cases where the prodVersion is either 1.1, 1.2 or null:

```
{
      "prodVersion":
          {
              "type": "stringArray",
              "value": ["1.1", "1.2"],
              "includeNull": true
          }
}
```

By default, when specifying a filter, null values would not be included. Therefore, in the following example, only clients with prodVersion set to 1.1 or 1.2 will be included, while null values are excluded:

```
{
      "prodVersion":
          {
              "type": "stringArray",
              "value": ["1.1", "1.2"]
          }
}
```

However, if no filter is specified, then nulls are included by default. Therefore, if you want to include any value of prodVersion as long as it is not null, a prodVersion filter needs to be included as follows:

```
{
      "prodVersion":
          {
              "type": "regex",
              "value": ".*",
              "includeNull": false
          }
}
```

In the case of filters that use sub-properties (os, geography, and gpu), the includeNull filter is to be included in the sub-property and applies to that specific sub-property only. In order to be able to include the includeNull property, instead of providing the value as a string or an array of strings, the value of the sub-property must be a JSON object that contains a property named “value”, and another named “includeNull”. Each of these properties is optional, but at least one of them must be present.

In the case of geography, this has a very particular meaning. Requesting for null “country” value does not return all cases where the country could not be retrieved, but only cases where the continent could be retrieved but the country could not. Similarly, requesting null “usState” returns cases where the continent and country could be retrieved but the US state could not. This does not include clients that are not situated in the US. If you are interested in finding clients where we could not detect any geographical data, the includeNull filter needs to be applied in the continent sub-property.

In the following example, we are requesting cases where we know that the client is within the US but the state could not be identified:

```
{
      "geography":
```

```
            {
                "type": "string",
                "country": "US",
                "usState":
                    {
                        "includeNull": true
                    }
            }
}
```

In the following example, we are requesting cases where the GPU is either “NVIDIA”, “AMD” or null (unidentified):

```
{
      "gpu":
            {
                "type": "stringArray",
                "vendor":
                    {
                        "value": ["NVIDIA", "AMD"],
                        "includeNull": true
                    }
            }
}
```

Results Format

The results consist of an array of JSON objects. Each object contains data about 1 single client. Each of these objects contain a `clientId` property as minimum. All other properties are optional and are dependent on the properties that were requested in the `properties` field, and whether `reqDailyData` has been specified. The current data (i.e. the last known values for each requested property for each client) is presented as properties in this object. The values are either strings, numbers or boolean values - depending on what each property contains.

If requested, the daily data is inside a property in the above mentioned object named `dailyData`. The value of this property is an array of objects - one object for each day on which the installation was active. As a minimum, each of these objects contain a `date` property formatted as YYYY-MM-DD. The rest of the properties depend on what properties were requested in the properties value of `reqDailyData`.

The `reachOutDeliveries.auto` and `reachOutDeliveries.manual` properties contain an array of objects. Each object refers to 1 `reachOut` campaign delivery. These objects contain 2 properties: `reachoutID` and name. The `reachoutId` is a numeric value given to each `reachOut` campaign as an identifier. The name is the campaign name as specified by the creator of the campaign. In case the `reachOut` campaign had been deleted, the value of the name property will be null.

The `lifetimeEventUsage` property contains data about all events performed by the client throughout the client lifetime. This is an array of objects. Each object contains 3 properties - `eventCategory`, `eventName`, and `eventCount`.

Example Request/Response

In the following example, we are requesting the following properties: `geography.country`, `ram`, `lifetimeEventUsage`, and `reachOutDeliveries.auto`. We are requesting daily data for 1 - 10 January 2018, and asking for 1 property in the daily data: `prodVersion`. We are also filtering for 1 single client ID.

Example Request

```
POST /reporting/clientPropertyList  HTTP/1.1
Host: api.revulytics.com
Content-Type: application/json
Accept: application/json

{
    "user": "testuser@test.com",
    "sessionId": "VSB8E2BzSC2eZSJm4QmTpA",
    "productId": 12345678901,
    "globalFilters":
          {
              "clientId":
                  {
                      "type": "string",
                      "value": "0AC28257D3E9F271"
                  }
          },
    "properties":
          [
              "geography.country",
              "ram",
              "lifetimeEventUsage",
              "reachOutDeliveries.auto"
          ],
    "retDailyData":
          {
              "startDate": "2018-01-01",
              "stopDate": "2018-01-10",
              "properties":
                  [
                      "prodVersion"
                  ]
          }
}
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "status": "OK",
    "results":
          [
              {
                  "clientId": "0AC28257D3E9F271",
                  "geography.country": "US",
                  "ram": 8192,
                  "lifetimeEventUsage":
                      [
                          {
                              "eventCategory": "File Operations",
                              "eventName": "Open",
                              "eventCount": 42
                          },
```

```
                            {
                                "eventCategory": "File Operations",
                                "eventName": "Delete",
                                "eventCount": 5
                            }
                        ],
                    "reachOutDeliveries.auto":
                        [
                            {
                                "reachoutID": 7,
                                "name": "Welcome message"
                            },
                            {
                                "reachoutID": 15,
                                "name": "Upgrade to premium"
                            }
                        ],
                    "dailyData":
                        [
                            {"date": "2018-01-03", "prodVersion": "1.1"},
                            {"date": "2018-01-04", "prodVersion": "1.1"},
                            {"date": "2018-01-08", "prodVersion": "1.1"},
                            {"date": "2018-01-10", "prodVersion": "1.4"}
                        ]
                }
            ],
        "reachedEnd": true
}
```

16

Raw Data Export

Raw Data Export functionality needs to be enabled on your product account for this functionality to work. Downloading of raw data export files works in a similar way to Custom Event Tracking. In both cases, the list of files can be retrieved, and then a temporary URL may be requested for downloading.

- Downloadable File Listing
- Download Files

Downloadable File Listing

This request returns the list of zipped files that are on the server. This list is to be used to see what files are available for download and then be able to request files for downloading.

- Request/Response Parameters Summary
- Example Request/Response

Request/Response Parameters Summary

`POST /rawEvents/download/listFiles`

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 16-1 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. • **sessionId (string)**—The sessionId obtained via POST /auth/login. • **productId (integer)**—The product ID on which this request is being done/

Table 16-1 • Request Properties

Property	Description
Response JSON Object	• status (string)—Contains OK if successful or SYNTAX ERROR or AUTH ERROR. • reason (string)—Present only if status is not OK. Contains error message (reason). • **fileList (array)**—Array of JSON objects containing information about available files. These files can be requested for download.

Example Request/Response

This section includes a sample request and sample response.

Example Request

```
POST /rawEvents/download/listFiles  HTTP/1.1
Host: api.revulytics.com
Content-Type: application/json
Accept: application/json

{
    "user": "testuser@test.com",
    "sessionId": "VSB8E2BzSC2eZSJm4QmTpA",
    "productId": 2376158762
}
Example response:
HTTP/1.1 200 OK
Content-Type: application/json

{
   "status": "OK",
   "fileList": [
      {
          "fileName": "2376158762_2019-05-08.zip",
          "fileDate": "2019-05-08",
          "compressedSizeKB": 275.28
      }
      {
          "fileName": "2376158762_2019-05-07.zip",
          "fileDate": "2019-05-07",
          "compressedSizeKB": 327.14
      }
   ]
}
```

Download Files

In order to download the data files, the file needs to be requested to the API which returns a secure URL from which the file can be downloaded.

```
POST /rawEvents/download/getDownloadUrl
```

The request and response are both JSON objects. The following is a summary of the properties inside the request and response objects.

Table 16-2 • Request Properties

Property	Description
Request JSON Object	• **user (string)**—The username of your Usage Intelligence user account. • **sessionId (string)**—The sessionId obtained via POST /auth/login. • **productId (integer)**—The product ID on which this request is being done. • **fileName (string)**—The filename of the file being requested to download
Response JSON Object	• **status (string)**—Contains OK if successful or SYNTAX ERROR or AUTH ERROR. • **reason (string)**—Present only if status is not OK. Contains error message (reason). • **downloadUrl (string)**—The one-time URL used to download the zip file

Example Request/Response

This section includes a sample request and sample response.

Example Request

```
POST /rawEvents/download/getDownloadUrl  HTTP/1.1
Host: api.revulytics.com
Content-Type: application/json
Accept: application/json

{
    "user": "testuser@test.com",
    "sessionId": "VSB8E2BzSC2eZSJm4QmTpA",
    "productId": 2376158762,
    "fileName": "2376158762_2019-05-08.zip"
}
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
   "status":"OK",
   "downloadUrl":"https://analytics-evt.revulytics.com/download/CzyLGrtHXpDixrPlMItFCyZTCagLEy"
}
```