Quest™

Quest® Unified Communications Analytics 8.7

Resource Kit

Legend

! | **CAUTION: A CAUTION icon indicates potential damage to hardware or loss of data if instructions are not followed.**

i | **IMPORTANT NOTE**, **NOTE**, **TIP**, **MOBILE**, or **VIDEO:** An information icon indicates supporting information.

UC Analytics Resource Kit
Updated - July 2020
Software Version - 8.7

Contents

Using the resource kit to extract data

UC Analytics provides a resource kit to give you easier access to the data you have collected. The resource kit includes the following tools:

- A bulk data exporter that allows you to retrieve large amounts of data from UC Analytics. It allows you to export your collected data to a .csv file. The bulk data exporter is automatically installed with the UC Analytics Data Engine.

  Using the bulk data exporter
- An Open Data Protocol (OData) endpoint. You can access the OData endpoint using third party OData tools such as Microsoft Power BI or PowerView.

  Using Microsoft Power BI to explore the OData endpoint
- An OData client library. You can use the OData library to create ad hoc queries using PowerShell or C#.

  Using the OData client library

Using the bulk data exporter

Using a command line interface, you can use the bulk exporter to extract specific data from your UC Analytics collected data. You run the bulk exporter on the directory that hosts the Data Engine.

To run the bulk exporter

1. Navigate to the Data Engine directory located in the UC Analytics installation directory on the server on which the Data Engine is installed.
2. Enter the executable name (UC.Analytics.Insights.DataEngine.BulkDataExport.exe) followed by the parameters that meet your inquiry.

Table 1. Parameters that can be used with the bulk exporter.

Available parameters	Definition
/FilePath	File path and file name of the output file
/ColumnDelimiter	Default character is **,** (comma). You can specify a different delimiter character (optional)
/Entity	Target entity name such as EmailMessage
/Fields	Comma separated field names such as SendDate or Sender.EmailAddress.Person.Name
/ObfuscatedFields	Comma separated field names (optional)
/Filter	Filter expression (optional)
/StartDate	Start date (optional)
/EndDate	End date (optional)
/PageSize	Buffer size (The default value is 1000)

Table 1. Parameters that can be used with the bulk exporter.

Available parameters	Definition
/MaximumNumberOfResults	Number of objects (optional)
/OrderAscending	Sets the order of the results. True or false (default value is false)
/OrderByField	Field name (optional)
/ShowDataModel	Provides a complete up-to-date list of all the entities, fields, and their data types. Cannot be combined with other parameters

Listing all available entities and fields

To see a list of all the available entities, fields and their data types, use the /ShowDataModel switch on the command line and send the output to a file. For example, you could enter the following command:

```
D:\Program Files\Quest\UC Analytics\Data Engine>
UC.Analytics.Insights.DataEngine.BulkDataExport.exe /ShowDataModel > dataModel.txt
```

TIP: When you enter parameters, remember that field names are case-sensitive and must exactly match the names as shown by the /ShowDataModel switch.

Common entities in the UC Analytics data model

The following is a list of common entities that comprise the UC Analytics data model:

- EmailMessage
- ActiveDirectoryObjectSnapshot
- LyncConference
- LyncPeerToPeerSession
- LyncQoeSession
- CiscoConference
- CiscoPeerToPeerSession

Additionally, you can retrieve the full metadata from OData (http://<*server name*>:1336/$metadata).

The resource kit supports obfuscated data so that you can do data analysis without revealing private data.Data obfuscation is a form of data masking where data is purposely scrambled to prevent unauthorized access to sensitive materials.

Examples of bulk exporter commands

Your query can be simple or complex, depending on the depth of data that you want to extract.

In a simple example, you can extract the subject, send date, and sender name for all the email message in your database:

```
UC.Analytics.Insights.DataEngine.BulkDataExport.exe /Entity EmailMessage /FilePath
output.csv /Fields "Subject, SendDate, Sender.EmailAddress.Person.Name"
```

In a more complex example, you could filter the query to include only the subject lines from email messages that were sent by Stuart Murgal as of January 1, 2015.

```
UC.Analytics.Insights.DataEngine.BulkDataExport.exe /Entity EmailMessage /FilePath
output.csv /Filter "Sender.EmailAddress.Person.Name == \"Stuart Mughal\" "/Fields
"Subject" /StartDate "2015-01-01"
```

Examples using multiple filters

In another situation, you might want to filter for multiple users in a single query. In this example, you could export the message subject lines for messages sent on a specific date by several specified senders and also see the recipients (To, CC, and BCC) for each message. In this example, the || (double pipe) means OR.

```
UC.Analytics.Insights.DataEngine.BulkDataExport.exe /Entity EmailMessage /FilePath
C:\output.csv /Fields "Subject, SendDate,
Sender.EmailAddress.Person.Name,RecipientsInTo.EmailAddress.Person.Name,RecipientsI
nCc.EmailAddress.Person.Name,RecipientsInBcc.EmailAddress.Person.Name","
Recipients.EmailAddress.Person.Name" /Filter "Sender.EmailAddress.Person.Name ==
\"Salvatore Denham\" || Sender.EmailAddress.Person.Name == \"Tony Brownlee\" ||
Sender.EmailAddress.Person.Name == \"Maria Santos\" ||
Sender.EmailAddress.Person.Name == \"Eric Johnson\" ||
Sender.EmailAddress.Person.Name == \"Frederick Johansson\" ||
Sender.EmailAddress.Person.Name == \"Sandra Butterfield\" ||
Sender.EmailAddress.Person.Name == \"Vladimir Ivanov\" ||
Sender.EmailAddress.Person.Name == \"Caroline Northwood\" ||
Sender.EmailAddress.Person.Name == \"Andrew Chen\"" /StartDate "2018-10-09" /EndDate
"2018-10-10"
```

In the following example, the query is filtering message recipients (To, CC, and BCC) in a specific department (R&D-Development) who received messages from senders in a specific company (Sitraka) and office. The query defines a specific date and includes the message subject line. In this example, the && (double ampersand) means AND.

```
UC.Analytics.Insights.DataEngine.BulkDataExport.exe /Entity EmailMessage /FilePath
C:\output.csv /Fields "Subject, SendDate,
Sender.EmailAddress.Person.Name,RecipientsInTo.EmailAddress.Person.Name,RecipientsI
nCc.EmailAddress.Person.Name,RecipientsInBcc.EmailAddress.Person.Name","
Recipients.EmailAddress.Person.Name" /Filter
"Recipients.EmailAddress.Person.Department == \"R&D-Development\" &&
Sender.EmailAddress.Person.Company == \"Sitraka\" &&
Sender.EmailAddress.Person.Office==\"AMER-US-Aliso Viejo\"" /StartDate "2018-10-09"
/EndDate "2018-10-11"
```

Suppose that you want to refine your query to view the recipients (To, CC, and BCC) of all messages sent on a specific day from two specific senders in a company. In this example, the query is retrieving messages sent by Anthony Jones or Maria Santos at Sitraka to recipients in your R&D-Development department. This example uses both OR (||) and AND (&&).

```
UC.Analytics.Insights.DataEngine.BulkDataExport.exe /Entity EmailMessage /FilePath
C:\output.csv /Fields "Subject, SendDate,
Sender.EmailAddress.Person.Name,RecipientsInTo.EmailAddress.Person.Name,RecipientsI
nCc.EmailAddress.Person.Name,RecipientsInBcc.EmailAddress.Person.Name","
Recipients.EmailAddress.Person.Name" /Filter
"(Recipients.EmailAddress.Person.Department == \"R&D-Development\" ||
Sender.EmailAddress.Person.Company == \"Sitraka\")&&
(Sender.EmailAddress.Person.Name ==\"Anthony Jones\"||
Sender.EmailAddress.Person.Name==\"Maria Santos\")" /StartDate "2018-10-09" /EndDate
"2018-10-11"
```

Why would the SendDate field be empty?

Why would some fields such as SendDate be empty?

Answer

The Send Date is affected by the types of data collections you have scheduled, the sources from which you are collecting, and by the message origins. If you only collect from the Exchange Tracking Logs data source, but do not collect from mailboxes using the Exchange Mailbox Contents data source, the Send Date may be approximate or missing for some messages. For example, the SendDate may not be set if you are not collecting all the tracking logs from all your Exchange mailbox and hub servers.

If you collect from both the Exchange Mailbox Contents and the Exchange Tracking Logs data sources, the Send Date is always set by the Mailbox Contents data collection. To always collect the exact Send Date, you must collect using both the Tracking Logs and the Mailbox Contents data sources.

When Send Date is approximate

If you collect only from the Exchange Tracking Logs data source, the Send Date for SMTP or inbound messages is approximate. For messages submitted through SMTP or originating from outside your Exchange organization, the Delivery Time is not calculated until the Exchange Calculation job has run.

For multi-hop message delivery, a message might appear in the tracking logs on several Exchange servers. Since the Send Date will be set by an event (StoreDriver Submit or SMTP Receive) which can appear in any one of the tracking logs, the Send Date is approximate.

Why would fields that link to Sender not be set?

Sometimes Sender and fields that follow links such as Sender.DeliveryTime are not set. Why?

Answer

The SenderDeliveryTime field is set only if senders send the message to themselves. The RecipientsDeliveryTime field is usually set.

The SenderFirstResponseTime is the duration between the receipt of the message, and the first reply to the message. So the field is mainly applicable to Recipients (rather than Sender) and is set only for messages that are collected from mailboxes using the Mailbox Contents data source (not Exchange Tracking Logs).

The SenderSendTimeofDay and SentAfterHours fields are also set only for messages collected using the Mailbox Contents data source.

If you want the sender name rather than the key, you can use Sender.EmailAddress.DisplayName field instead.

Using Microsoft Power BI to explore the OData endpoint

You can use Microsoft Power BI Desktop to access the UC Analytics OData endpoint to extract and view your collected data. For performances reasons, it is recommended that you always edit and filter the query before loading the data. For example, you might filter by limiting the query to a short period of time or limiting the query to selected people.

The workflow for using Power BI can be divided into the following procedures:

1. Connecting to OData.
2. Selecting and downloading the data to your local computer.
3. Creating a visualization of the data.

The procedure to connect to OData is provided followed by a sample scenario to show the procedures for selecting and downloading data and for creating a visualization.

Prerequisites

You must install Microsoft Power BI Desktop on your computer. You can download Power BI Desktop from the following Microsoft web site: https://powerbi.microsoft.com

You also must have unrestricted access to all the data (configured in the Security settings in the UC Analytics Admin Setting) that you want to extract. For information about granting access to the UC Analytics data, see the *UC Analytics Deployment Guide*.

To connect to OData

1. Open Power BI Desktop.
2. On the home ribbon, click **Get Data**.

   Data types are organized in the following categories:

   - All
   - File
   - Database
   - Azure
   - Other
3. Select the **All** category which includes all data connection types from all categories.
4. In the All list, scroll down to locate the OData Feed.
5. Select **OData Feed** and click **Connect**.
6. Under URL, enter the URL for the server on which the UC Analytics Data Engine is installed and specify port 1336.

   For example, for server MyAnalytics, you would enter: http://MyAnalytics:1336

7 Click **OK**.

Power BI Desktop makes the connection to OData Feed and, in the Navigator, presents a list of the available tables.

! **TIP:** Do not select the Load button at the bottom of the Navigator pane before you have edited your query. If you simply select a table and click **Load**, all the data in the table is loaded into local RAM on your computer. For example, if you selected **EmailMessages** and clicked **Load**, data for every email message stored in UC Analytics would be downloaded to your computer. The download could take a very long time.

8 To edit the query before loading data, select the **Edit** button.

Sample scenarios

The following sample scenarios explain how to download selected data and to create visualizations using Power BI Desktop. By following the steps in these scenarios, you can become familiar with some of the Power BI Desktop functions.

For more information about using Power BI Desktop, see the knowledge base at the Microsoft Power BI support site: https://support.powerbi.com/knowledgebase.

Using Power BI Desktop to recreate a graph from the Email - Activity insight

The following example shows you how to recreate a graph from the Email - Activity insight that shows the number of email messages over time. To recreate the graph, you need two fields from the EmailMessages table: Key and Timestamp.

To select email message data over time

1 After you have connected to OData and displayed the tables in the Navigator, select **EmailMessages** and click **Edit**.

2 In the Query Editor ribbon, click **Choose Columns**.

3 Clear **(Select All Columns)**.

4 Select **Key** and **Timestamp**, and click **OK**.

5 Right-click **Timestamp** and select **Change Type**.

6 Select **Date**.

Email messages are stored using the exact date and time of each message. You change the Timestamp column from Date/Time to Date so that messages can be aggregated by day to produce a reasonable line graph.

7 In the Query Editor ribbon, click **Close & Apply**.

Your changes are saved. When you first load data in Power BI Desktop, you will see Report View with a blank canvas.

The Visualizations and Fields panes should be displayed on the right.

Optional: Using the Advanced Editor

If you want to see the code that Query Editor is creating with each step, or want to create your own code, you can use the Advanced Editor.

To launch the advanced editor, select **View** from the ribbon and select **Advanced Editor**. A window appears, showing the existing Query code.

For example, you could select the Key and Timestamp fields from the EmailMessages table using the Advanced Editor by entering the following sample query.

To enter the selection code in the advanced editor

1. After you have connected to OData and displayed the tables in the Navigator, right-click **EmailMessages** and select **Advanced Editor**.
2. Paste the following query into the Advanced Editor:

```
let
   Source = OData.Feed("http://MyAnalytics:1336",
   EmailMessages_table = Source{[Name="EmailMessages",Signature="table"]}[Data],
   #"Removed Other Columns" = Table.SelectColumns(EmailMessages_table,{"Key", "Timestamp"}),
   #"Changed Type" = Table.TransformColumnTypes(#"Removed Other Columns",{{"Timestamp", type date}})
in
   #"Changed Type"
```

3. Click **Done**.

Restricting a query to a specified date range

To improve query performance, you can insert code to specify a start date and an end date for the data shards to be included in the query. You can insert the following code immediately after first line as follows:

```
let
Source = OData.Feed("http://MyAnalytics:1336",
[ ],
   [Query=[
      IsFromExternalToolRequest="true",
      StartDate="2019-03-01T00:00:00",
      EndDate="2019-03-07T00:00:00"]
   ] ),
EmailMessages_table = Source{[Name="EmailMessages",Signature="table"]}[Data],
   #"Removed Other Columns" = Table.SelectColumns(EmailMessages_table,{"Key", "Timestamp"}),
   #"Changed Type" = Table.TransformColumnTypes(#"Removed Other Columns",{{"Timestamp", type date}})
in
   #"Changed Type"
```

For the StartDate and EndDate you enter the dates that specify the time period for which you want to include records.

After you click **Done**, the query does not reflect the selected date range. In the Query Editor ribbon, you must click **Refresh Preview** to preview the data with specified shards. You can click **Close & Apply** to view the data for the date range from the database.

To create a graph that shows email messages over time

After you have downloaded the email message data by date, you can create a visualization of the data.

1. In the Visualizations pane, select the line graph icon.

   Under the Visualizations pane, a secondary pane shows the line graph parameters such as Axis, Legend, and Values.

2. Drag the **Key** and **Timestamp** fields, listed in the Fields pane, to the Axis and Values labels respectively.

   The resulting line graph should closely resemble the graph in the Email - Activity insight.

Using Power BI Desktop to create a graph for specific user email activity

The following example shows you how to recreate a graph that shows the number of email messages over time for a specific user. You need to use two different tables: EmailMessages and EmailMessageParticipants.

To select email message data for a specific user over time

1. After you have connected to OData and displayed the tables in the Navigator, select **EmailMessages** and **EmailMessageParticipants** and click **Edit**.
2. When the Query Editor has the EmailMessages table in preview, in the Query Editor ribbon click **Choose Columns**.
3. Clear **(Select All Columns)**.
4. Select **Key** and **Timestamp** and **Sender,** and click **OK**.
5. Right-click **Timestamp** and select **Change Type**.
6. Select **Date**.

   Email messages are stored using the exact date and time of each message. You change the Timestamp column from Date/Time to Date so that messages can be aggregated by day to produce a reasonable line graph.

7. Click the down arrow at the top right of the column header to expand the **Sender** column.
8. Clear **(Select All Columns).**
9. Select **Key**.
10. When the Query Editor has the EmailMessageParticipants table in preview, in the Query Editor ribbon click **Choose Columns.**
11. Clear **(Select All Columns).**
12. Select **Key** and **EmailAddress** and click **OK**.
13. Click the down arrow at the top right of the column header to expand the **EmailAddress** column.
14. Select **DisplayName**.
15. Right-click on the **DisplayName** column and rename the column to **Participant Name**.
16. In the Query Editor ribbon, click **Close & Apply**.

    Your changes are saved. When you first load data in Power BI Desktop, you will see Report View with a blank canvas. The Visualizations and Fields panes should be displayed on the right.

17. In the Power BI ribbon, click **Manage Relationships** and select **Autodetect…**

    This step detects the relationships between the data.

To create a graph that shows email messages for a specific user

After you have downloaded the email message data by date and the email participant data, you can create a visualization of the data.

1. In the Visualizations pane, select the line graph icon.

   Under the Visualizations pane, a secondary pane shows the line graph parameters such as Axis, Legend, and Values.

2. Drag **Key** and **Timestamp** fields, listed in the Fields pane, from the EmailMessages entity into the Values and Axis fields respectively.
3. Drag **Participant Name** field, listed in the Fields pane, from the EmailMessageParticipants entity into the Legend field for the line graph.

   You should see the email activity trends for all the participants.

4. To analyze a specific participant, expand Participant Name under the Filters, and select the name of the participant that you want to see.

   The resulting line graph should show the email activity over time for a specific user.

Using Power BI Desktop to create a bar chart for Exchange public folder item counts

The following example shows you how to create a vertical bar chart for public folders that shows the number of items in each public folder on a particular day.

To select public folder item count for a specific date

1. After you have connected to OData and displayed the tables in the Navigator, select **ExchangePublicFoldersSnapshot** and click **Edit**.
2. In the Query Editor ribbon, click **Choose Columns**.
3. Clear **(Select All Columns)**.
4. Select **Key, Fullpath, ItemCount, Name,** and **SnapshotDate** and click **OK**.
5. Right-click **SnapshotDate** and select **Change Type**.
6. Select **Date**.

   Messages are stored in public folders using the exact date and time of each message. You change the SnapshotDate column from Date/Time to Date so that messages can be aggregated by day to produce a total item count for the specified date.

7. At the top of the SnapshotDate column, click the down arrow to see a list of snapshot dates.
8. Select the snapshot date that you want.
9. In the Query Editor ribbon, click **Close & Apply**.

   Your changes are saved. When you first load data in Power BI Desktop, you will see Report View with a blank canvas. The Visualizations and Fields panes should be displayed on the right.

To create a bar chart that shows public folder item count for a specific date

After you have downloaded the public folder items and snapshot date, you can create a visualization of the data.

1. In the Visualizations pane, select the vertical bar chart icon.

   Under the Visualizations pane, a secondary pane shows the bar chart parameters such as Axis and Values.

2. Drag the **Name** and **ItemCount** fields, listed in the Fields pane, to the Axis and Values labels respectively.

3 Under Filters, click the down arrow to expand **Name(All)**.

4 Locate **IPM_SUBTREE** and clear the check box.

5 Under Visualizations, click the paintbrush icon to display customization options.

6 Set Data Labels to **On**.

The resulting bar chart will show all the public folders and the number of items in each for the selected day.

Using Power BI Desktop to create a trend for item counts in an Exchange public folder

The following example shows you how to create a trend that shows the number of items in a specific public folder over time.

To select public folder item counts over time

1 After you have connected to OData and displayed the tables in the Navigator, select **ExchangePublicFoldersSnapshot** and click **Edit**.

2 In the Query Editor ribbon, click **Choose Columns**.

3 Clear **(Select All Columns)**.

4 Select **Key, Fullpath, ItemCount, Name,** and **SnapshotDate** and click **OK**.

5 Right-click **SnapshotDate** and select **Change Type**.

6 Select **Date**.

Messages are stored in public folders using the exact date and time of each message. You change the SnapshotDate column from Date/Time to Date so that messages can be aggregated by day to produce a total item count for the each date.

7 In the Query Editor ribbon, click **Close & Apply**.

Your changes are saved. When you first load data in Power BI Desktop, you will see Report View with a blank canvas. The Visualizations and Fields panes should be displayed on the right.

To create a trend that shows the item counts in a public folder over time

After you have downloaded the public folder item count data, you can create a visualization of the data.

1 In the Visualizations pane, select the line graph icon.

Under the Visualizations pane, a secondary pane shows the line graph parameters such as Axis, Legend, and ItemCount.

2 Drag the **SnapshotDate**, **Name**, and **ItemCount** fields, listed in the Fields pane, to the Axis, Legend, and ItemCount labels respectively.

3 Under Filters, click the down arrow to expand **Name(All)**.

4 Clear **(All)** and check the public folder name that you want to trend.

5 Under Visualizations, click the paintbrush icon to display customization options.

6 Set Data Labels to **On**.

The resulting trend graph will show the public folder you selected and the number of items in the folder for each day in the snapshot dates.

About query performance

If you query for a large amount of data from the tables, the query can take a significant amount of time to complete. It is recommended that, when you edit queries, you only select the fields that you need.

Table 2. Examples of load times for different query sizes.

Number of Rows	Number of Columns	Query Size (MB)	Query Load Time (seconds)
30,000	3	4.5	13
30,000	23	18	47

About columns with multiple values

By default, in OData each row contains a single entity such as one message, one conference, and so on. So for columns that can contain multiple values (such as participant), if you select Count for the Count of Key, you will see duplicate values.

For example, if you selected participants and a message has several participants, the same message appears in multiple rows, one entry for each participant. In this case, change the Count of Key from Count to Count (Distinct) to get a count of distinct keys.

Limitations

Only scalar values can be expanded. This means that if you attempt to expand a nested collection or a dynamic field, you get a an error or null values.

Workaround

To get around the limitation of being unable to expand a nested collection, you could query and download the primary entity and the entity you want to expand. Then you can allow Power BI to autodetect the relationship between the entities using foreign keys.

An example of this workaround is shown in the scenario Using Power BI Desktop to create a graph for specific user email activity on page 12.

In this scenario Participants.EmailAddress.Name could not be expanded since EmailAddress is a collection of a collection (Participants). So how do we get EmailAddress.DisplayName? We must also query for the EmailMessageParticipants entity which has the property EmailAddress. Here we can expand and get EmailAddress.DisplayName because DisplayName is a scalar property of EmailAddress.

3

Using the OData client library

Using the OData client library you can create ad hoc queries using PowerShell or C#. The OData client library complements the bulk data exporter but is slower and is meant to support a smaller amount of data. Unlike the bulk data exporter, you can create queries to get data from calculated fields (such as direction or chargeback amounts) and you can also retrieve aggregated data.

While the bulk data exporter must be run locally, the OData library is secure and can be run remotely.

The OData client library is installed in the Resource Kit folder on the server that hosts the Data Engine. By default, the path is as follows:

C:\Program Files\Quest\UC Analytics\Resource Kit

In the Resource Kit folder are the .dll files that you must copy to the server from which you want to run your queries.

About the sample files

In the Resource Kit folder there is a folder named Samples that contains examples of both PowerShell and C# queries. For example, the AggregateQuerySamples files for C# and for PowerShell (.cs and .ps1) provide samples that show you how to create queries for aggregated data such as:

- the number of Inbound messages sent to a specific department
- the number of messages received by a specific email address grouped by the offices that sent the most messages
- the average message delivery times grouped by country
- the number of messages sent to a distribution group, grouped by month, for a defined number of months

To see individual data records, the ObjectQuerySamples files provide a sample query to show a list of the individual inbound messages sent to a specific department.

To get a dump of all the entities and fields that comprise the UC Analytics data model, you can use the DataModelSamples files.

IMPORTANT: In version 8.4.1, UC Analytics was rebranded from Dell to Quest and the namespace definitions were changed. In older versions, file names and namespaces began with **Dell.UC.** but were renamed to **UC.Analytics.** The sample files were updated in release 8.4.1.

If you have scripts based on sample files that were created before version 8.4.1, ensure that you change instances of Dell.UC. to UC.Analytics. to update them.

Prerequisites

You access the OData library using the .NET interface and must be proficient in either PowerShell or C# to write your own queries.

Overview: Using the OData library with PowerShell queries

The OData client library allows you to query the UC Analytics database for information that has been gathered from your target environments. This section describes the main syntax used in the query layer. Though the examples are provided in PowerShell, the information is applicable to C#. Since the syntax is similar, you can extend the examples for C#.

To establish the queries, you must understand PowerShell and be able to read and understand the main PowerShell syntax constructs.

The current OData implementation is a single-view query that answers single-focused questions. The query layer is not a substitute for the more complex types of queries you can do in UC Analytics.

Initializing the client library

For a query to be executed, you must create a connection to the OData client library. All examples include this information but the initialization step is highlighted here:

```
# -----------------------------------------------------------------------------------------------------------------------
# Initialize client library
$clientLibraryPath = "..\UC.Analytics.Insights.ResourceKit.ODataClientLibrary.dll"
$additionalLibraryPath = "..\Simple.OData.Client.Net40.dll"


try
{
    Add-Type -Path $clientLibraryPath
    Add-Type -Path $additionalLibraryPath


    $serverUrl = "http://" + $analyticsServerName
    $clientLibrary = New-Object
UC.Analytics.Insights.ResourceKit.ODataClientLibrary.AnalyticsQueryClient($serverUrl)
}
catch
{
    Write-Host "Encounter error : $Error[0].Message"
    Exit
}
```

About the initialization step

For the initialization step, the following rules apply:

- You enter the computer name but do not add the Analytics folder (as in the website)
- Even if you have configured https for the website, the URL remains http.

The libraries required to run the PowerShell commands are located at:

```
%UCA_INSTALL_DIRECTORY%\Resource Kit
```

The initialization code assumes that the script files in the Samples subdirectory are open. If you move your script file, you must update the initialization code to explicitly point to the new location of the dependent library files.

Introducing the query structure

Once initialization is complete, the next step is to define the query. UC Analytics supports two types of queries:

- aggregate queries
- object queries.

About aggregate queries

Aggregate queries are queries that take a collection item and aggregate a metric over a given time period. For example, a query might show the number of messages sent by mailboxes over the last month.

- The query output is in the form of a key/value pair. The key is the field defined in the grouping and the value is the given metric value.
- Examples of aggregate queries are provided in the Samples directory in the file named AggregateQuerySamples.ps1.
- Aggregate queries have the following syntax:

```
$result = $clientLibrary.`
                For("{OBJECT_ITEM}").`
                          {TIME_CLAUSE}, `
                          {WHERE_CLAUSE}, `
                          {METRIC_VALUE}, `
                          {GROUPING_CLAUSE}, `
                          {TRENDING_CLAUSE}, `
      QueryAsync().Result
```

About object queries

Object queries are queries that query the objects in database. For example, a query might show all mailboxes with the Display Name, Primary Email Address, and Custom Attribute 8.

- The output of the queries are the selected attributes.
- Examples of object queries are provided in the Samples directory in the file named ObjectQuerySamples.ps1.
- Object queries have the following syntax:

```
   $result = $clientLibrary.`
      For("{OBJECT_ITEM}").`
                {TIME_CLAUSE}, `
                {WHERE_CLAUSE}, `
                {SELECT_CLAUSE}, `
```

```
                    {ORDER_CLAUSE}, `
            QueryAllAsync().Result
```

Each main clause is described later in this section.

Common query items

The query syntax is generic but order does matter. If you are familiar with C# LinQ, you might expect that you can change the order. However, for all OData library queries, you must enter the clauses in the order in which they are shown in the examples.

Also, the command structure shows a single command over multiple lines, which can be done in PowerShell. But to implement this option, you must include the backtick character `. Without the backtick character, your command will not work.

UC Analytics has a number of object items. To see a list of the items, run a PowerShell command called DataModelSamples.ps1 in the Samples folder. When you run the DataModelSamples.ps1 command, it lists all the available object types. Since the object types can change over time, it is important to run the command for your version of UC Analytics.

The following list shows the main object types in UC Analytics:

Table 3. Example of DataModelSamples.ps1 output showing object types

EmailMessage
EmailFileAttachment
EmailMailboxDelegateSnapshot
EmailMailboxDelegateSnapshot
EmailMailboxPermissionSnapshot
EmailMailboxSnapshot
ActiveDirectoryObjectSnapshot
CiscoConference
CiscoConferenceSession
CiscoCucmGroupSnapshot
CiscoCucmGroupSnapshot
CiscoCucmServerSnapshot
CiscoCucmServerSnapshot
CiscoDevicePoolSnapshot
CiscoDevicePoolSnapshot
CiscoDeviceSnapshot
CiscoDeviceSnapshot
CiscoDirectoryNumberSnapshot
CiscoGatewaySnapshot
CiscoGatewaySnapshot
CiscoPeerToPeerSession
CiscoUserConfigurationSnapshot
CrossPlatformActivity
DnsDomain
EmailPersonalArchiveMailboxSnapshot
EmailPersonalArchiveMailboxSnapshot
ExchangeActiveSyncEvent

Table 3. Example of DataModelSamples.ps1 output showing object types

ExchangeAppointment
ExchangeAppointmentAttachment
ExchangeAppointmentParticipant
ExchangeDatabaseAvailabilityGroupSnapshot
ExchangeDatabaseCopySnapshot
ExchangeDatabaseSnapshot
ExchangeDlpMatch
ExchangeLegacyPublicFolderReplicaSnapshot
ExchangeLegacyPublicFolderReplicaSnapshot
ExchangeLegacyPublicFolderSnapshot
ExchangeOrganizationSnapshot
ExchangePublicFolderSnapshot
ExchangePublicFolderSnapshot
ExchangeServerAddressSnapshot
ExchangeServerSnapshot
LyncArchivingPolicySnapshot
LyncConference
LyncConferenceMediaSession
LyncConferencingPolicySnapshot
LyncExternalAccessPolicySnapshot
LyncPeerToPeerSession
LyncPoolSnapshot
LyncQoeMediaSession
LyncQoeSession
LyncQoeSubnetSnapshot
LyncServerSnapshot
LyncUserConfigurationSnapshot
MobileDeviceSnapshot
Office365UserSubscriptionLicenseSnapshot
Office365UserSubscriptionServiceSnapshot
Office365UserSubscriptionSnapshot

Many sections require an attribute name and there is a naming convention for these attributes. The names are identified in the output of DataModelSamples.ps1 command. You must specify the full attribute name if it transcends the associated objects.

Finding a full attribute name

The following walkthrough explains the process of finding the full attribute name for an object. Suppose that you want to filter all email messages by the sender's department.

First, you run the DataModelSamples.ps1 to get a list of all the objects and their attributes.

To find the sender department for email messages

1 Since the query is on EmailMessage, find that object.

2 Look for the Sender field. It says the Sender field is a link to the EmailMessageParticipant item.

3 Find the EmailMessageParticipant item.
    - The EmailAddress item indicates the address of the object.
    - In the EmailAddress attribute, it says this field is a link to the MessagingAddressSnapshot

4 Find the MessagingAddressSnapshot item.

  This provides a long list of attributes.

5 Since you want to query on an ActiveDirectory attribute, locate the Person attribute.

  In the Person attribute, it says this field is a link to ActiveDirectoryObjectSnapshot.

6 Find the ActiveDirectoryObjectSnapshot item.

  Now you have the list of Active Directory attributes for the Sender.

7 Find the Department attribute.

The final step is to follow the attributes to establish the full attribute name. You use a period to separate the items. The required items are bolded to make it easy to see.

In the example, the field would be: **EmailAddress.Person.Department**. You do not need to enter the first item as the query assumes that you will be starting from the Object Item.

About the Object item

The first required item for a query is the object item against which you are querying. The object items are found in the output of the DataModelSamples PowerShell sample. You provide the object type in quotes, such as the following: For("EmailMessage"). This part of the query is required for both object and aggregate query types.

About the Time clause

The next step in the query is to establish the time slice to which you want to limit your query. You add the time clause as an explicit item so that you do not need to include the time clause in the main Where clause. The list of supported time clauses is as follows:

Table 4. Time clauses used in queries.

Time clause	Definition	Syntax
WithinLast	Within the last X periods	Specified as WithinLast(PeriodCount, PeriodIdentifier)
Before	Before a particular date	Specified as Before(DateValue) When a date is specified, the query uses the end of the specified date so the query will include the date. (The time portion of the date is ignored.)
After	After a particular date	Specified as After(DateValue) When a date is specified, the query uses the start of the specified date so the query will include the date. (The time portion of the date is ignored.)
Between	Between two dates	Specified as Between(StartStartDate, EndDateValue) For each date, the date follows the same rules as Before and After. (The time portion of the date is ignored.)

For classes that require a date value, all dates must be UTC date/times. The best way to establish the date is to use the following two-step process:

```
$startDate = Get-Date -Date "2014-09-01 00:00:00Z"

$startDate = $startDate.ToUniversalTime()
```

Here is an example:

After($startDate)

For clauses that reference a time period, UC Analytics supports a number of pre-defined periods. They are as follows:

- Minute
- Hour
- Day
- Week
- Month
- Quarter
- Year

To use a time clause that references a period, you must associate the given period as a local variable that is referenced in the actual clause. The periods are part of the RelativeDate attribute and are as follows:

- $minutes = [UC.Analytics.Insights.ResourceKit.ODataClientLibrary.RelativeDate]::Minute
- $hours = [UC.Analytics.Insights.ResourceKit.ODataClientLibrary.RelativeDate]::Hour
- $days = [UC.Analytics.Insights.ResourceKit.ODataClientLibrary.RelativeDate]::Day
- $weeks = [UC.Analytics.Insights.ResourceKit.ODataClientLibrary.RelativeDate]::Week
- $months = [UC.Analytics.Insights.ResourceKit.ODataClientLibrary.RelativeDate]::Month
- $quarters = [UC.Analytics.Insights.ResourceKit.ODataClientLibrary.RelativeDate]::Quarter
- $years = [UC.Analytics.Insights.ResourceKit.ODataClientLibrary.RelativeDate]::Year

For example, if you want to look at messages in the last five weeks, you would add the $weeks item and establish the following time clause:

WithinLast(5, $weeks)

About the Where clause

The Where clause in the query definition is used to specify conditions that restrict the selected items. As with most query syntax, the Where clause follows a simple structure:

{AttributeName} {Operator} {Value}

To establish a filter, you must establish a reference to the filter type:

$filter = [UC.Analytics.Insights.ResourceKit.ODataClientLibrary.Filter]

Specifying the Filter operator

Specifying the Attribute name and value

Specifying the Filter operator

UC Analytics supports the following comparison filter operators that must be specified as local variables:

Table 5. Operators for comparison filters

Operator	Definition
Equal	$equal = [UC.Analytics.Insights.ResourceKit.ODataClientLibrary.FilterOperator]::Equal
NotEqual	$notequal = [UC.Analytics.Insights.ResourceKit.ODataClientLibrary.FilterOperator]::NotEqual
GreaterThan	$greaterthan = [UC.Analytics.Insights.ResourceKit.ODataClientLibrary.FilterOperator]:: GreaterThan

Table 5. Operators for comparison filters

Operator	Definition
GreaterThanOrEqual	$greaterthanorequalto = [UC.Analytics.Insights.ResourceKit.ODataClientLibrary.FilterOperator]:: GreaterThanOrEqual
LessThan	$lessthan = [UC.Analytics.Insights.ResourceKit.ODataClientLibrary.FilterOperator]:: LessThan
LessThanOrEqua	$lessthanorequalto = [UC.Analytics.Insights.ResourceKit.ODataClientLibrary.FilterOperator]:: LessThanOrEqual

Specifying the Attribute name and value

Next, you must specify the attribute name. The naming convention for the attributes is explained in Finding a full attribute name on page 20 and applies in this clause. Enter the full name of the attribute on which you want to filter.

For the value, there are different values that you can enter. The value that you need to specify depends on what is listed as the attribute type. Possible values are listed here:

- String: Enter the text in double quotes.
- Int32, Int64, Double: Enter the numeric value without any quotes.
  - Some of the numeric values are enumerated types that provide the mapping.
  - The mapping items are provided in the output of DataModelSamples.ps1. For example, Email Message Sensitivity has its own item EmailMessage/Sensitivity, with one of four items. To query based on Private, you would use 4.
- DateTime: Specify the UTC date time using the format described earlier in the document.
- Boolean: Specify one of the PowerShell Boolean values: $TRUE $FALSE

Based on this simple Where clause, you would enter:

```
Where($filter::Clause("EmailAddress.Person.Department ", $equal, "R&D"))
```

About more complex Where clauses

Though a simple Where will work, you can use more complicated Where clauses. See the following examples for more complex scenarios:

Filtering on multi-valued attributes

Specifying the Not operator

Specifying the AND / OR operator

Filtering on multi-valued attributes

Some fields are collections of records. One example is Recipients. UC Analytics provides the ability to establish a query based on the following criteria for these multi-valued fields:

- Any: At least one of the items in the collection satisfies the criteria.
- All: All the items in the collection satisfy the criteria.

The query for the underlying recipient follows the previous example. The additional field is added above it to provide the total filter.

For example, to query on any recipient with something specified in the recipient name:

Where($filter::Clause("Recipients", $any, $filter::Clause("EmailAddress.Person.Name", $equal, "Joe Smith")))

To use either of these two fields, you must define the associated local variables:

- $any = [UC.Analytics.Insights.ResourceKit.ODataClientLibrary.FilterOperator]::Any
- $all = [UC. Analytics.Insights.ResourceKit.ODataClientLibrary.FilterOperator]::All

Specifying the Not operator

The Not operator is a common Boolean operator. UC Analytics provides support for this construct using the same style as the multi-valued attribute. The only difference is that you define a filter expression that you reference to the specified filter clause.

You must specify the Not operator:

$not = [UC.Analytics.Insights.ResourceKit.ODataClientLibrary.FilterOperator]::Not

For example, you could query on any recipient that does not have a value in the recipient name:

Where($filter::Expression($not, $filter::Clause("EmailAddress.Person.Name", $equal, "Joe Smith")))

Specifying the AND / OR operator

Much like the Not operator, the AND / OR operators are common. These operators are used in a similar manner as the Not operator, but with two filter clauses.

You must specify the additional operators:

- $and = [UC.Analytics.Insights.ResourceKit.ODataClientLibrary.FilterOperator]::And
- $or = [UC.Analytics.Insights.ResourceKit.ODataClientLibrary.FilterOperator]::Or

For example, to query on any recipient that does not have a value in the recipient name:

Where($filter::Expression($and, $filter::Clause("EmailAddress.Person.City", $equal, "New York"), $filter::Clause("EmailAddress.Person.Department", $equal, "R&D")))

You can achieve the AND operator by providing two consecutive Where conditions. The following statement is the same as the previous Where condition for AND:

Where($filter::Clause("EmailAddress.Person.City", $equal, "New York")), `

Where($filter::Clause("EmailAddress.Person.Department", $equal, "R&D")), `

You can use this clause in both query types.

About Metric values

When you run queries against the UC Analytics database, you can provide a specific metric. You can also provide an operation on the metric. The following metric values are supported:

- Count()
- CountDistinct()
- Average(“AttributeName”)
- Min(“AttributeName”)
- Max(“AttributeName”)
- Sum(“AttributeName”)

All metrics are specified in the format: Value(“Fieldname”). The Fieldname uses the same format as the Attribute name in the Where clause. Since fields that require Attribute names are computational, this applies only to numeric fields.

You can specify multiple metric values in a single query. However, the values apply to the grouping that is established for the entire query. The Metric clause applies only to Aggregate queries.

About the Group clause

A common scenario in querying a database is being able to group the results. This part of the query allows you to specify one or more grouping levels.

Each grouping level is a definition of the attribute on which you want to group. The attribute syntax is described in Specifying the Attribute name and value on page 23.

Here is an example of a two level group-by:

```
GroupBy("Recipients.EmailAddress.Person.ObjectType").`
GroupBy("Recipients.EmailAddress.Person.Name").`
```

The Group clause only applies to Aggregate queries.

About the Trending clause

Trending in this component is a time-based grouping that is sorted. The trending clause uses the following model:

```
TrendBy("AttributeName", TrendingTimeFrame)
```

The AttributeName syntax is described in Specifying the Attribute name and value on page 23. As this clause is time-based grouping, the attribute specified needs to be a time-based attribute.

The TrendingTimeFrame is a defined variable that must be referenced in your script. The following variables are supported in UC Analytics:

- $hourly = [UC.Analytics.Insights.ResourceKit.ODataClientLibrary.TrendMode]::Hourly
- $weekly = [UC.Analytics.Insights.ResourceKit.ODataClientLibrary.TrendMode]::Weekly
- $monthly = [UC.Analytics.Insights.ResourceKit.ODataClientLibrary.TrendMode]::monthly
- $quarterly = [UC.Analytics.Insights.ResourceKit.ODataClientLibrary.TrendMode]::Quarterly
- $yearly = [UC.Analytics.Insights.ResourceKit.ODataClientLibrary.TrendMode]::Annually

You can specify only a single trending clause. The Trending clause applies only to Aggregate queries.

Using the Select clause to return additional information

In many cases, the real information is the number that the query is returning. But sometimes you want additional information as well. This Select clause allows you to specify the additional fields that you want. You provide the list of AttributesNames (using the syntax previously described) in a comma-separated list.

Here is a simple example:

```
Select("EmailAddress.Person.City", "EmailAddress.Person.Department"), `
```

The Select clause can be specified only once in a query and is only applicable to Object queries. If you specify the Select clause in an Aggregate query, it will result in an error.

Using the Order clause to define sorting

Once you have all the main query criteria specified, the last step is a sorting definition. This definition returns the list sorted according to the specified AttributeName.

The syntax is simple, so an example will be enough:

```
OrderBy("EmailAddress.Person.City"), `
```

UC Analytics supports only a single attribute name and a single attribute clause per query. As such, only a single sort order is supported. The Order clause is only applicable to Object queries. If you specify the Order clause in an aggregate query, the query will result in an error.

Working with the Query results

Once you have defined and executed your query, the next step is to process the call results. The first step is to check the main result value from the query call. The result value should be an object and not the Null value. If you encounter a Null value, there is a problem. For information about what to check in this case, see the section titled Notes about using theOData client library on page 28.

The next step depends on the type of query you are running.

If you are running an Object query

The result of an object query is a collection of the object types that you specified in the query definition. The first step is to check for the number of returned objects. You can do this as follows:

```
Write-Host "Object Count =" $messages.Count
```

This code goes with the example in the section that follows.

To get the records, you iterate through the items in the collection using the PowerShell ForEach item. The attributes on the iterated object are the same as those specified in the Select clause. As shown in the ObjectQuerySamples.ps1 sample file, this is done in the following way:

```
$messages = $clientLibrary.`
For("EmailMessages").`
        Between($i_startDate, $i_endDate).`
        Where($filter::Clause(`
            "Recipients", $any, $filter::Clause(`
                "EmailAddress.Person.Department", $equal, $i_department))).`
         Select("Sender.EmailAddress.DisplayName", "SendDate", "Subject").`
         QueryAllAsync().Result
Write-Host
 ForEach ($message in $messages)
  {
        Write-Host $message.SendDate
        Write-Host $message.Sender.EmailAddress.DisplayName
        Write-Host $message.Subject
        Write-Host
  }
```

If you are running an Aggregate query

When you run an aggregate query, you are provided with a collection of results. These results represent the total number of objects that were used in the aggregation.

```
Write-Host Total Messages: $result.TotalObjects
```

After you check the object count, check the Metric count. In the Metric value section, it shows that you can configure your query to have several metrics. For each metric specified in the Metric value, there is a Metric value in the Metric results collection. To get each metric, look at each item and look at the Group results:

```
$metric = $result.Metrics[0]

Write-Host Total Messages: metric.GroupResults.Key $metric.GroupResults.Value
```

Here is an example about how to access the main values:

```
    $result = $clientLibrary.`
            For("EmailMessages").`
            WithinLast($i_numberOfWeeks, $weeks).`
            Where($filter::Clause(`
                "Recipients", $any, $filter::Clause(`
                        "EmailAddress.Person.Department", $equal, $i_department))).`
            Count().`
            Average("Size").`
            QueryAsync().Result
   if ($result.Metrics.Count -gt 0)
  {
            $metric = $result.Metrics[0]
            Write-Host $metric.MetricId $metric.GroupResults.Value
            $metric = $result.Metrics[1]
            Write-Host $metric. MetricId $metric.GroupResults.Value
  }
```

This results in two lines being written for each metric:

```
0:Count 877

1:Average(Size) 30662.976
```

These lines are the total values of the aggregation. If your query contained a GroupBy clause, you can see the value for each item in the grouping by iterating through the Groups method of the GroupResults (shown previously).

For each Group, you can output the Key and Value pair. If you have included a Trending clause, you can look at the Groups collection of that object and output the Key and Value pair for the trended time frame. An example is shown in the AggregateQuerySamples.ps1 sample file, in the TrendAverageMessageDeliveryTimeByCountry method. See the following example:

```
   $result = $clientLibrary.
            For("EmailMessageParticipants").`
            After($i_startDate).`
   Average("DeliveryTime").`
            GroupBy("EmailAddress.Person.CountryOrRegion").`
            TrendBy("Message.Timestamp", $daily).`
```

```
              QueryAsync().Result
  Write-Host Total Messages: $result.TotalObjects
  if ($result.Metrics.Count -gt 0)
 {
              $metric = $result.Metrics[0]
    Write-Host $metric.MetricId : $metric.Value
             ForEach ($countryGroup in $metric.GroupResults.Groups)
             {
                           Write-Host $countryGroup.Key : $countryGroup.Value
                          ForEach ($dayGroup in $countryGroup.Groups)
                          {
                                         Write-Host $dayGroup.Key : $dayGroup.Value
                          }
                          Write-Host
             }
     }
```

Notes about using theOData client library

The OData client library has a few specific characteristics.

Minimal error reporting

Generally, no errors are returned from the backend. For example, no errors are returned for a bad server name, if the database is not functional, or for anything that is not a syntax problem.

The only times when errors are returned are:

- Syntax errors for a specific part.
- When a clause is used in the wrong query type.

When you check the main results of the query, if nothing is returned, there a problem. If your query returns nothing (a NULL value), check the following:

- The name of the UC Analytics server – an incorrectly named server will return NULL.
- Whether UC Analytics is up and running – if the main UC Analytics website is not running correctly, the PowerShell layer will likely not work.
- An incorrect attribute name. Review all your attribute names. If you specify an incorrect attribute name, it can result in a NULL value being returned. For example, if you specified “EmailAddress.Person.CountryRegion” instead of “EmailAddress.Person.CountryOrRegion”, the query returns nothing.

Merging query contents

To combine the contents of an object query and an aggregate query, or two aggregate queries, you must write the applicable PowerShell code. The data returned from these query calls can be programmatically merged through standard PowerShell techniques.