ALLAN THRAEN | 3 years ago | PDF |

Tips and Tricks Optimizely (Episerver) Personalization CMS

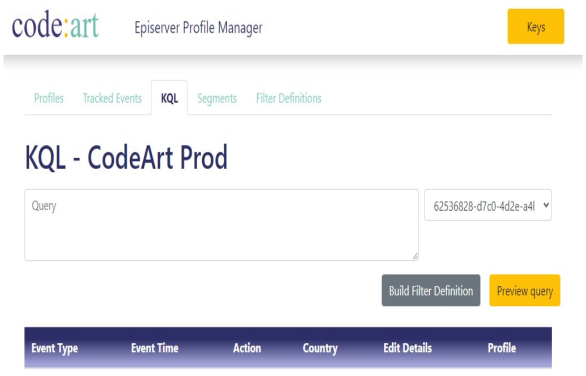
PROFILE STORE KQL CHEAT SHEET

With KQL support in the Rest API for Episerver profile store we have been given a powerful tool to query against the tracked profiles. In this post I share a collection of cool and useful queries you can use.

Getting started

First of all - as much as you might like Swagger - I personally find it practical to use a more dedicated interface to test the queries in. You can of course build your own client directly against the rest API - but if you just want to test nice and easy, feel free to use our free **Profile Manager tool**.

In this tool, you will want to move to the KQL tab and select the correct Scope, to start querying. By default, the scopes are named after the Site Guid - and I'm still trying to figure out why it's not just using the site name or domain or if it's even a good idea with scopes in the first place. But I digress.



Obviously before you do this, you'll need to configure the Keys with your endpoint and credentials to a profile store you want to query. You can learn more about the profile manager here.

Starting to query

KQL is Kusto Query Language (not to be confused with Keyword Query Language - another MSFT product) and you can read the documentation here. For now, all you can query are the Events tracked in the profile store - not the actual profiles (there you still have to use the oData-like syntax). But don't fear - profiles are merely on top of the Events which does hold a lot of really useful and valid data, assuming you have been careful to track relevant data points on your site.

KQL is basically a piped syntax, where you start with a dataset and pipe it through various commands and functions. The simplest query you can make is just "*Events*" which will return all events tracked - from there you can add pipes, commands and functions.

Note, the queries below are some of my favorite ones - but they might not work for you. It all depends on what data you are tracking.

Filtering

Let's look at some common filtering patterns. Remember you can filter using "where" at any level - but for performance reasons it's a good idea to filter as much out as possible (like for example date range filters or EventType filters) as early as possible - that means that the data being worked on in the rest of the pipes is much less.

Events between two dates (in this case december 2020):

Events|where EventTime>datetime(2020-12-01) and EventTime<datetime(2021-01-01)

Events where the Uri contains a certain string:

Events|where PageUri contains '404'

Events that match on a payload item (in this case I'm tracking a NewCommentEvent and looking for the comment text):

Events|where EventType == 'NewCommentEvent' and Payload.comment contains 'great'

Pageview events, except those to a specific url:

Events|where EventType == 'epiPageView' and not(PageUri contains '/contact/')

Visits from either Sweden or Finland:

```
Events|where CountryCode in ('SE','FI')
```

As you see above, you can use 'in' to filter against a list. But that list can also be a dynamic subquery. To make it a bit easier you can even define it before using it (or alternatively write it inline). Here I have a query that returns events coming from any of the 50 rarest countries to visit your website:

```
let rarecountries=Events|summarize cnt=count() by CountryCode|order by cnt asc|project
CountryCode|take 50;
Events|where CountryCode in (rarecountries)
```

There are a tons of extension functions in Kusto Query Language - and they can come in quite handy. One of them is the parse_url which can be used to extend your dataset with an object containing the elements of a parsed url (Host, Path, Query parameters and so on). For example you can use it to filter on only retrieving the events which contains a search query in the query parameters. Notice that we first have to extend the events with this data, before we can filter on it. This of course also means that we can summarize and project on it afterwards as well.

```
Events|extend UrlDetails=parse_url(PageUri)|where UrlDetails["Query Parameters"].q!=""
```

You can find other very useful scalar functions here.

Summarizing

Usually, when using the rest api, you'd expect to get events back from a KQL call. However, you can still use 'summarize' to group data and return aggregated data - and it will work. If you are using an SDK that expects the data to be events, you might have to use projection to put summarized values into the fields normally used by Events. If you are using Profile Manager, it will try to adjust to the data - at least if you are summarizing it. However if you are extending or projecting data then you can benefit from opening the Developer Console, as the raw Json objects are written to Console as they are received.

Events distribution by types:

```
Events|where EventTime > datetime(2020-01-01)|summarize count() by EventType
```

Inferred Country distribution:

```
Events|where EventType=='MitAlmBrandEntry'|summarize cnt=count() by CountryCode|order by cnt
desc
```

Histogram by month:

```
Events|project monthstring=strcat(getyear(EventTime),"-",getmonth(EventTime)),
m=startofmonth(EventTime)|summarize cnt=count() by monthstring,m|order by m|project
monthstring,cnt
```

Most common form submissions (assuming you are tracking form submissions):

```
Events|where EventType == 'FormSubmission'|summarize cnt=count() by Value, PageUri|sort by cnt
desc
```

Most popular pages seen by visitors that has also seen a given page (in this case /contact/):

```
Events|where EventType == 'epiPageView' and not(PageUri contains '/contact/')|join (Events|where
PageUri contains '/contact/' and EventType == 'epiPageView'|project DeviceId) on DeviceId|summarize
Count = count() by Value = tostring(PageUri)| top 1000 by Count desc
```

Hits on a given page (in this case /404) in a given time period, aggregated per day.

```
Events|where EventTime>datetime(2020-08-01) and PageUri contains '404'|summarize count() by
bin(EventTime,1d)
```

Most common query parameters used in requests to the site:

```
Events|extend UrlDetails=parse_url(PageUri)|extend keys=bag_keys(UrlDetails["Query
Parameters"])|where array_length(keys)>0|project k=strcat_array(keys,',')|summarize cnt=count() by
k|sort by cnt desc
```

Most common referrer hosts pointing to your site (note the use of 'tostring' since some functions doesn't work with dynamics):

```
Events|where Payload.epi.referrer!=""|extend
rdetails=parse_url(tostring(Payload.epi.referrer))|summarize cnt=count() by
tostring(rdetails["Host"])|order by cnt desc
```

Joins

An extremely useful tool is joins - with joins you can merge two queries and find where they intersect (left join) - but you can essentially do any of the classic join types.

Here is an example of a query that uses joins to find pageviews by a visitor who also submitted a certain form (in this case identified by a partial url):

```
Events|where EventType == 'epiPageView' and EventTime>datetime(2021-01-01)|join (Events|where
PageUri contains 'partial-url-to-page-with-form' and EventType=='FormSubmission'|project DeviceId)
```

```
on DeviceId
```

A common use case for joins is as an alternative to using the aggregated profiles (especially since you can't query those with KQL). For example finding visitors using the same device matching on multiple different actions - like above - those that submitted a form or logged in, or wrote a complaint - what else did they do that led up to that.

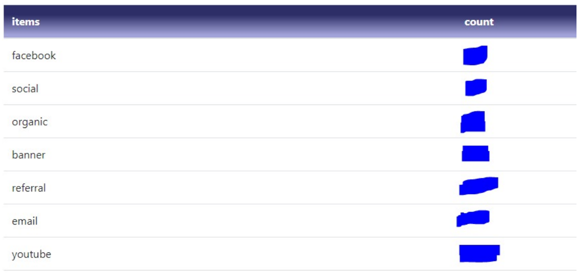
Here is an example, where we look at all events from visitors who arrived at your website in the last day from a campaign and we've extended all the events with "campaign","source" and "medium" taken from the landing page's query parameters utm_medium,utm_campaign and so on. To make it display nicely in ProfileManager, I use the little trick of assigning the medium to the pre-existing "Value" property:

```
let utmtable=Events|where EventTime > ago(1d) and EventType=='epiPageView'|extend
u=parse_url(PageUri)|where u["Query Parameters"].utm_medium!=""|project
DeviceId,medium=u["Query Parameters"].utm_medium, source=u["Query
Parameters"].utm_source,campaign=u["Query Parameters"].utm_campaign;
Events|where EventTime > ago(1d)|join utmtable on DeviceId|extend Value=medium
```

And naturally we can extend on this above to summarize it, to see which medium brought traffic that had the most events today:

```
let utmtable=Events|where EventTime > ago(1d) and EventType=='epiPageView'|extend
u=parse_url(PageUri)|where u["Query Parameters"].utm_medium!=""|project
DeviceId,medium=u["Query Parameters"].utm_medium, source=u["Query
Parameters"].utm_source,campaign=u["Query Parameters"].utm_campaign;
Events|where EventTime > ago(1d)|join utmtable on DeviceId|summarize cnt=count() by
tostring(medium)|order by cnt desc
```

Which would yield something like this:



items	count
facebook	[redacted]
social	[redacted]
organic	[redacted]
banner	[redacted]
referral	[redacted]
email	[redacted]
youtube	[redacted]

Useful Filter Definitions

Filter Definitions are an incredibly powerful tool. Filter Definitions are essentially templates built by the developer, that marketing manager can then use to construct segments in Profile Store - and eventually use those segments to personalize content.
You can learn more about segmenting in Dmytro Duk's great post here.
In profile store, you can easily create Filter Definitions from a KQL query, by using the "Build Filter Definition" button on the KQL tab, after you have adjusted an appropiate query.

Here are a few common use filter definitions I like...

Visitors that has searched for a given query since a given timespan (note, this assumes you are tracking query parameters - alternatively you can use parse_url as shown above):

```
{
"Id": "fd_HasSearchedFor",
"Name": "fd_HasSearchedFor",
"Description": "Visitors that has searched for a given query",
"Query": "Events|where EventTime >=(now()-{{since}}) and Payload.epi.queryParameters.q contains
{{query}}",
"Parameters": {
"since": "timespan",
"query": "string"
},
"Category": "Behavior",
"Type": "Events"
}
```

Visitors that has arrived from a given referrer (this assumes you are tracking referrers):

```
{
"Id": "fd_HasArrivedFrom",
"Name": "fd_HasArrivedFrom",
"Description": "Has arrived from a given partial url",
"Query": "Events|where EventTime>=(now()-{{since}}) and Payload.epi.referrer contains {{url}}",
"Parameters": {
"since": "timespan",
"url": "string"
},
"Category": "Behavior",
"Type": "Events"
}
```

Visitors that has seen a given Uri on your site:

```
{
"Id": "fd_HasVisitedUri",
"Name": "fd_HasVisitedUri",
"Description": "Has visited a given Uri",
"Query": "Events|where EventTime>=(now()-{{since}}) and PageUri contains {{uri}}",
"Parameters": {
"since": "timespan",
"uri": "string"
},
"Category": "Behavior",
"Type": "Events"
}
```

Visitors arriving with a specific utm campaign (again, this is based on tracking of query parameters, but could be done with parse_url):

```
{
"Id": "fd_ArrivalWithUTMCampaign",
"Name": "fd_ArrivalWithUTMCampaign",
"Description": "Profiles that has arrived within a certain amount of days, from a specific UTM
Campaign",
"Query": "Events|where EventType == 'epiPageView' and
Payload.epi.queryParameters.utm_medium!=\"\" and Payload.epi.queryParameters.utm_campaign==
{{campaign}} and EventTime>=(now()-{{within}})",
"Parameters": {
"campaign": "string",
"within": "timespan"
},
"Category": "Campaigns",
"Type": "Events"
}
```

Visitors that left a comment on a blog post (tracked in NewCommentEvent) with a certain keyword:

```
{
"Id": "fd_CommentWithKeyword",
"Name": "CommentWithKeyword",
"Description": "Left a comment with a specific keyword on an article.",
"Query": "Events|where EventType == 'NewCommentEvent' and Payload.comment contains '{{word}}'",
"Parameters": {
"word": "string"
},
"Category": "Engagement",
"Type": "Events"
}
```

Do you have any useful KQL

Please drop a line in the comments if you have some awesome KQL snippet you'd like to share.
I will also try to keep this document updated with more.

Tips and Tricks | Optimizely (Episerver) | Personalization | CMS

RECENT POSTS

CodeArt ApS
Teknikerbyen 5, 2830 Virum, Denmark
Email: info@codeart.dk
Phone: +45 26 13 66 96
CVR: 39680688