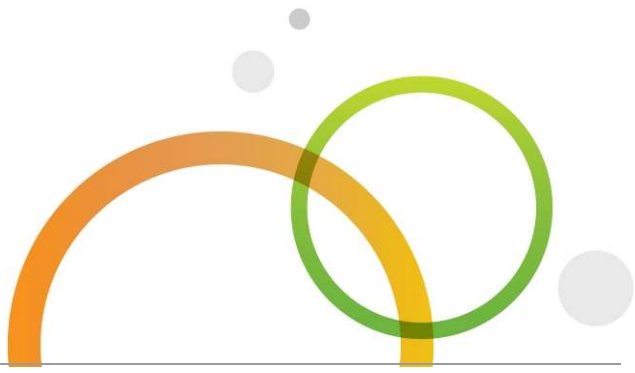
Qlik Sense Extension for On-Demand App Generation

User Guide

May, 2016



qlik.com

Table of Contents

Introduction

The Qlik Sense extension for On-Demand App Generation provides a comprehensive example of how to use the Qlik Analytic Platform APIs to create Qlik Sense applications for visualizing very large and/or frequently changing data sets. At its essence, the extension provides a way to navigate between a source application and a newly generated target application using a button placed on one of the sheets of the source application. As part of the navigation, the target app's data content is dynamically loaded from its underlying data source, using data filtering criteria controlled by selections made in the source application.

The On Demand App Generation approach expands the potential use cases for Business Discovery, enabling business users to conduct associative analysis on larger data sources. It provides a shopping list experience to allow users to first select data they are interested in discovering insights on which interactively generates an analysis app with full Qlik in-memory capabilities.

The main value of the On Demand App Generation extension stems from its ability to parameterize the target application's data load operation using values taken directly from the active selection state of the source app to finely control the subset of data loaded into the generated app.
By reading this document, you will learn:

- Setup of the solution and details of the sample apps.
- The specific set of features of the On Demand App Generation extension.
- Technical details of the internals of the extension to better understand how it uses QAP APIs and ways you might build your own extension that provides similar functionality.

Features

At its core, the On Demand App Generation extension provides a way to navigate from a source app to a target app through means of a button. However, given that a Qlik Sense app can have multiple sheets each with different views of a complex data space, why would you need a button to navigate between different Qlik Sense apps? The answer to this question is twofold.

First, the volume of detail level data can be very large making it difficult to load all of the data, at all possible levels of dimension granularity, into a single app. While it is possible to partition the data space into distinct subsets with a different app for each subset, if there are many different usage profiles, it can be difficult to accurately predict which subsets of data are needed across the full user community. And even if it is possible to predict the necessary data subsets, the sheer number of them can be so large that it might be costly to proactively maintain all of them.

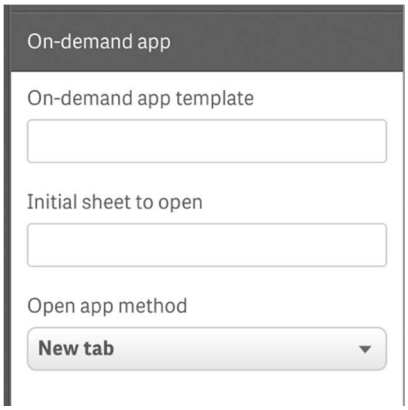
The second reason to link two applications together using the On Demand App Generation extension is when the detailed level data is frequently changing, and users need to see the most recent data to make good decisions. Of course, it is common for many or all of these characteristics to co-exist in the same high data volume, high velocity environment.

Regardless of the exact mix of data conditions, every use of the On Demand App Generation extension shares a common usage pattern. The user of a “Data Selection App”:

- uses the Qlik associative experience to explore a data space in a Selection App, typically at a moderate level of granularity (e.g. sales by quarter, region and product category rather than by hour, store and sku).
- uses the On-Demand App Generation feature to navigate to a separate, on-demand Analysis App that presents a set of data related to the selections made in the data selection app.

Use selections in data selection app to control content of on-demand app

The first property to be specified when using the On-Demand App Generation extension is the name of an existing Qlik Sense app to use as a template for generating apps. The **On-demand app template** property identifies an app that you must create in advance of using the extension in the Data Selection app.



The on-demand app template is a normal Qlik Sense app with one additional important difference: its load script contains one or more data binding expressions of the following form:

```
$(od_FIELDNAME)
```

The "od_" prefix is a pattern that the On-Demand App Generation extension uses to bind the selection state of the data selection app into the load script of the on-demand app. Before that binding process begins, the On-Demand App Generation extension makes a fresh copy of the on-demand app template so that the template can be used for other future app generation cycles. All data binding operations are applied to the new copy.

The part of the data binding expression that follows the "od_" prefix (in this case *FIELDNAME*) must be a name that matches a field in the data selection app. When the app generation process is triggered, the extension uses the current selection state of the data selection app to obtain the desired values to bind for each field. The extension performs the binding operation by replacing each occurrence of a $(od_FIELDNAME) expression in the script of the newly created on-demand app with the list of values selected for the corresponding field in the selection state of the data selection app.

For example, in order to query a set of product sales records for a specific product SKU, you could write a SQL SELECT statement in the on-demand app's script using the following WHERE clause:

```
SELECT …
FROM SALES
WHERE SKU = $(od_PRODSKU);
```

For the on-demand app template to work properly, that data selection app must have PRODSKU as one of its fields otherwise no data binding will be possible. Assuming that the user selects a single PRODSKU value of '1234' in the data selection app, then the submitted form of the SELECT statement will be:

```
SELECT …
FROM SALES
WHERE SKU = '1234';
```

But what if the user of the data selection app selects *multiple* values of PRODSKU? In this case the extension replaces $(od_PRODSKU) with a comma-separated list of the PRODSKU values, each value wrapped in single quotes (this choice of quoting and separation character are the defaults but can be changed using advanced properties described below). In order for the on-demand app's SELECT statement to be valid SQL syntax, its form needs to be changed to use an IN clause as follows:

```
SELECT …
FROM SALES
WHERE SKU IN ( $(od_PRODSKU) );
```

Now if the user chooses PRODSKU values '1234' and '1235', the generated SQL statement will still be correct, as it would be even if the user chooses a single PRODSKU value:

```
SELECT …
FROM SALES
WHERE SKU  IN ('1234','1235');
```

Navigating to a specific sheet in the generated app

In addition to specifying the name of the on-demand app template, you can also identify a specific sheet in the generated app to navigate to when the generated app is first opened. Since the On Demand App Generation extension copies all sheets of the on-demand app template when generating each new app, there can be a choice of several sheets to navigate to, the best choice of which may depend on the context in which the extension is used. If no value is provided for the **Initial sheet to open** property, the app overview page is presented when the generated app is first opened.

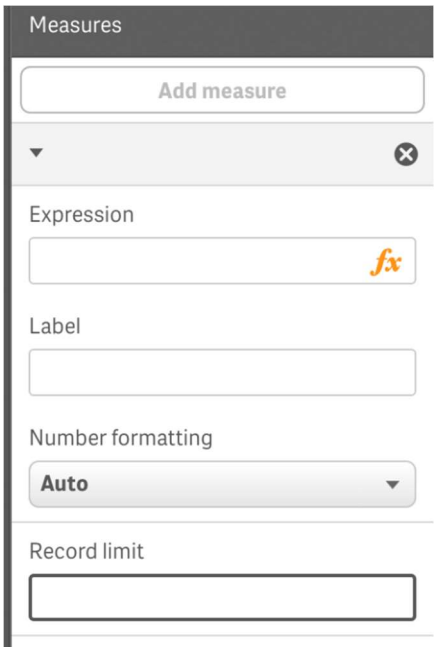
Controlling how the on-demand app opens

Depending on the browser device, you may have the option of controlling how the generated app is initially opened, using the **Open app method** property. For example, on a desktop device, you can choose between **New tab** or **New window**.

- If **New tab** is chosen, the extension will open the generated app in a new browser tab when the user clicks the open app button following app generation.
- If **New window** is chosen, the extension opens the generated app in a separate browser window.

Controlling the record volume of on-demand apps

In the case that the data source has a large number of detail records, you may wish to limit the use of the extension such that the number of actual records loaded into the on-demand app are kept within reasonable limits. The On Demand App Generation extension contains a pair of properties that work together to control the data volumes of the generated app.



- **Expression** is a measure expression that, when evaluated in the context of the data selection app, computes the total number of detail records qualified by the current selection state.
- **Record limit** holds a positive integer value representing the maximum number of detail records that can be qualified by the current selection state. To prevent users from generating apps with excessively large numbers of records, the extension's app generation button is only enabled when the qualified record count value computed by the measure expression is less than or equal to the **Record limit** value.

In very large data volume environments, it is common practice to have the data selection app loaded with data that has only a modest level of dimension granularity. In this scenario, the On-demand measure expression property is typically based on a computed aggregate result from a GROUP BY query used to load the data selection app. Using the SALES app example, a data selection app whose data is based on sales data aggregated to the quarter, region and product category level could use a SQL SELECT statement that looks as follows:

```
SELECT  SUM(S.UNIT_COST) AS TOTAL_UNIT_COST,
        SUM(S.QUANTITY) AS TOTAL_QUANTITY,
        SUM(S.UNIT_PRICE * S.QUANTITY) AS TOTAL_SALE,
        SUM( (S.UNIT_PRICE - S.UNIT_COST) * QUANITY) AS TOTAL_PROFIT
        SUM(1) AS TOTAL_LINE_ITEMS,
        S.REGION,
        S.YEARQUARTER,
        S.PRODCAT
```

```
FROM SALE_DETAIL
GROUP BY S.REGION, S.YEARQUARTER, S.PRODCAT
```

Since the data selection app uses a GROUP BY query to aggregate the SALE_DETAIL records, it is necessary to use aggregation functions – in this case SUM – on the measure fields of UNIT_COST, QUANTITY and the computed values for TOTAL_SALE and TOTAL_PROFIT amounts. If the data to be loaded by the on-demand app is at the level of individual SALE_DETAIL records and we want to ensure that all generated on-demand apps have record volumes no larger than 100,000 SALE_DETAIL records, then we need to specify a measure expression that computes the total number of SALE_DETAIL records qualified by the selection state of the data selection app.

To accomplish that, we first need a base value for the total line item count for every distinct combination of region, quarter and product category. This is precisely what "SUM(1) AS TOTAL_LINE_ITEMS" accomplishes in the GROUP BY load query above. Given that the user of the data selection app can select multiple product categories, multiple regions and/or multiple quarters, the measure expression needs to sum all of the TOTAL_LINE_ITEMS values for each distinct combination of region-quarter-productcat activated by the current selection state in order to reflect the total number of underlying SALE_DETAIL records that are qualified for on-demand load. Therefore, the measure property would be defined as "SUM(TOTAL_LINE_ITEMS)" and the **Record limit** property set to 100000.

Whenever the user's selection state is such that the SUM(TOTAL_LINE_ITEMS) measure value exceeds 100000, the button for generating the on-demand app will be disabled. The button is only enabled when the selection state results in a value of SUM(TOTAL_LINE_ITEMS) that is less than or equal to 100000.

Bind 'Optional' values in addition to 'Selected' values

Sometimes it is necessary to create a filter condition in the query of the On-demand template app's script using fields that are not directly selectable in the data selection app. For example, it may be the case that the SALE_DETAIL records use a code for region like REGION_CODE but the data selection app shows a friendlier REGION_NAME field for controlling region selections. The data selection app can have a model that uses a separate table to associate REGION_CODE with REGION_NAME so that users can select values of REGION_NAME as a way to control which regions are part of the selection state.

Unfortunately, there is a problem with this approach. By selecting values of REGION_NAME, the user will cause those values of REGION_NAME to enter the 'selected' state whereas the associated REGION_CODE values will only be in the 'optional' state (i.e. white rather than green). Furthermore, if the design of the data selection app's sheets excludes REGION_CODE from its set of filter panes, then there is no way to have the bind expression $(od_REGION_CODE) in the script of the On-demand app expand to the list of *selected* regions since REGION_CODE values will never actually be selected (i.e. made green). At best, they will only ever enter the optional (i.e. white) selection state when their associated REGION_NAME values are selected.

To handle this situation, the On Demand App Generation extension provides an optional syntax to more precisely control which selection state values are used in each data binding. Specifically, the "od_" prefix that precedes the field name portion in every on-demand bind expression may include any combination of the letters 's', 'o' and/or 'x' to denote whether the values to be used in the binding

are those taken from the '**s**elected' (i.e. green), '**o**ptional' (i.e. white) or e**x**cluded (or grey) state. The following table summarizes the valid combinations, using the REGION_CODE example:

Pattern	Expansion
$(ods_REGION_CODE)	Selected (i.e. green) values of REGION_CODE
$(odo_REGION_CODE)	Optional (i.e. white) values of REGION_CODE
$(odx_REGION_CODE)	Excluded (i.e. grey) values of REGION_CODE
$(odso_REGION_CODE)	Selected or Optional values of REGION_CODE
$(odsx_REGION_CODE)	Selected or Excluded values of REGION_CODE
$(odox_REGION_CODE)	Optional or Excluded values of REGION_CODE
$(od_REGION_CODE)	Same as $(ods_REGION_CODE) (i.e. only selected values)

In the case of our hypothetical on-demand sales app, we need to use the following data binding expression to be sure that *either* the selected or optional values of REGION_CODE are included in the REGION_CODE binding:

```
$(odso_REGION_CODE)
```

Bind numeric values

When the data to be bound to the on-demand app consists of numbers and not strings, it is helpful to be able to disable the quote wrapping behavior on those fields that contain numbers. For example, if the SALE_DETAIL records include a numeric DAY_OF_WEEK column and we want to allow the user of the data selection app to choose arbitrary combinations of DAY_OF_WEEK values, we would augment the aggregation query used for loading the data selection app to include DAY_OF_WEEK in both the SELECT list as well as the GROUP BY list. However, unless we have a way to suppress wrapping quotes around the DAY_OF_WEEK values that the user chooses, a load query that uses DAY_OF_WEEK as follows:

```
SELECT …
FROM SALES
WHERE DAY_OF_WEEK IN ( $(od_DAY_OF_WEEK) );
```

will expand to the following:

```
SELECT …
FROM SALES
WHERE DAY_OF_WEEK IN ( '1', '2' );
```

This could produce a runtime query error from the database engine if that database does not support automatic type coercion from string to numeric.

To handle this situation, the On Demand App Generation extension supports an optional "_n" suffix that can be added at the end of the FIELDNAME portion of the bind expression to force the field binding to use the numeric values from the selection app rather than the string values. To use the numeric values, the query for the on-demand app would use the following WHERE clause instead:

```
WHERE DAY_OF_WEEK IN ( $(od_DAY_OF_WEEK_n) );
```

Requiring a certain number of selections be made

In some situations, it may be necessary to require that the on-demand query contain a specific number (or range) of values of a specific field. For example, if the on-demand app's query contains a BETWEEN clause to obtain all sales between a start and end date, the bind expression for the YEARQUARTER field can have a suffix syntax of '[2]' that will require exactly 2 values be selected for YEARQUARTER, as in:

```
$(od_YEARQUARTER)[2]
```

If the selection state of the data selection app is such that there are not exactly 2 values of YEARQUARTER selected, the data load button will be disabled and a message will be displayed indicating that exactly 2 values of 'YEARQUARTER' must be selected (additional script logic can be written to split the two date values at their separator character to formulate a SQL BETWEEN clause using the two selected date values).
Note that by using selection quantity constraints, you are creating a prerequisite linkage between the data selection app and the on-demand app template. This behaves differently from bind expressions that lack the quantity constraints. For example, when your on-demand template app's script contains a bind expression without a quantity constraint, as in:

```
$(od_MYFIELD)
```

there is no requirement for there to be a field named MYFIELD in the selection app nor for there to be any selected values of that field even if it does exist. If the selection app does not contain a field named MYFIELD or, if it does and the user simply neglects to make any selections, the load data button can still become enabled assuming enough other selections are made to fulfil the record limit value condition.

In contrast, by changing that same bind expression to:

```
$(od_MYFIELD)[1+]
```

you introduce two requirements:

- The data selection app contains a field named 'MYFIELD'
- The user selects at least one value of MYFIELD in order to enable the data load button.

This form of bind expression needs to be used more carefully since it will limit which data selections apps can be used. Also, because the data binding process applies to bind expressions inside comments as well as normal script code, care must be taken to ensure that comments in your script code do not contain bind expressions of this form unless you are certain you want to impose that selection quantity requirement on all data selection apps.

There are other selection quantity constraint combinations possible. The following table summarizes the different combinations of selection quantity constraints:

Constraint pattern	Selection requirement
$(od_YEARQUARTER[2])	Exactly 2 values of YEARQUARTER must be selected.
$(od_YEARQUARTER[2-4])	Between 2 and 4 values of YEARQUARTER must be selected.
$(od_YEARQUARTER[2+])	At least 2 values of YEARQUARTER must be selected.
$(od_YEARQUARTER[2-])	At most 2 values of YEARQUARTER can be selected.

Sharing on-demand apps

The On Demand App Generation Extension utilizes publicly available QAP APIs. As such, it is constrained by the same set of security rules that apply to any normal Qlik Sense user and resource. The following sections describe security rule conditions that must be in effect in order to use the On Demand App Generation extension.

Changes needed to the "Stream" rule

In a production setting, the creator of the data selection and template apps would typically publish these apps so that other users can make use of the ODAG feature from the same selection app. In that case, users other than the creator of the template app will need read access to the template app in order to use it as a template to generate new apps. This presents a problem because, in its default installation state, the Qlik Sense server has a security rule, named "Stream", that specifically restricts access to the script of an app that the user doesn't own. In the case that the "Stream" security rule prevents access to the template app's script, the On Demand App Generation extension reports the following message:

```
No bind expressions of the form $(od_FIELDNAME) in On-demand app's
script match fields in this app.
```

In order to bypass this restriction for template apps, it is necessary to modify the "Stream" security rule to specifically allow access to the scripts of template apps. One way to achieve this is to add a custom app property named "ODAGTemplate", set that property to 'true' on all template apps and revise the "Stream" security rule to include a test for this property setting. Here are the steps necessary to accomplish this:

1) Create a custom "ODAGTemplate" property
    a) In QMC, go to Start->Custom Properties
    b) Click on "Create new"
    c) Set "Name" to "ODAGTemplate"
    d) Check the "App" resource type
    e) Add Values: "true" and "false"
2) Assign ODAGTemplate = "true" on all template apps
    a) In QMC, go to Start->Apps
    b) Select your template app from the list
    c) Click on "Edit"
    d) Check "Custom Properties"
    e) Under "Custom Properties", set ODAGTemplate = "true"
3) Modify "Stream" Security Rule test for ODAGTemplate="true"
    a) In QMC, go to Start->Security Rules
    b) Select the "Stream" security rule
    c) Click on "Edit"
    d) Under the advanced section, change the security from:

```
(resource.resourcetype = "App" and
resource.stream.HasPrivilege("read")) or
((resource.resourcetype = "App.Object" and resource.published
="true" and resource.objectType != "app_appscript") and
```

```
resource.objectType != "loadmodel") and
resource.app.stream.HasPrivilege("read"))
```

to the following (the red text is added):

```
(resource.resourcetype = "App" and
resource.stream.HasPrivilege("read")) or
((resource.resourcetype = "App.Object" and resource.published
="true" and (resource.objectType != "app_appscript" or
resource.App.@ODAGTemplate = "true") and resource.objectType !=
"loadmodel") and resource.app.stream.HasPrivilege("read"))
```

Selection App User must have Create Permission on App and AppObject

If the user of the data selection app does not have app creation permission, the On Demand App Generation extension will report the following error:

```
Failed to create new on-demand app.
```

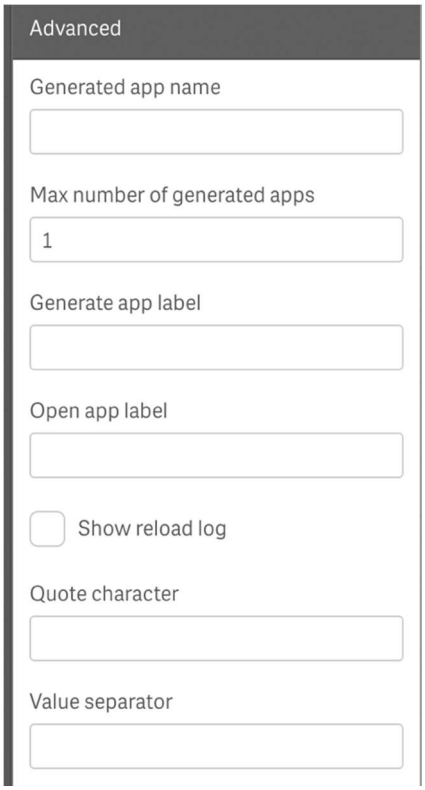
This can be circumvented by creating a new security rule that allows create app permission to any users, groups or roles that need to use the On Demand App Generation extension.

Selection App User must have "Read" access to Template App Data Connection

If the user of the data selection app does not have read access to any of the data connections that the template app uses to query data, the On Demand App Generation extension will fail in the data load phase with an error message reporting that the data connection object could not be found. In this case, it is necessary to use the QMC to augment the security rules for the Data Connections to allow read access to the Data Connections used by the template app scripts. Only users who have access to the streams in which the template apps have been published require read access to those template apps' data connections.

Advanced properties

Most uses of the On Demand App Generation extension will be possible using the features described in the prior sections. However, there are additional features that either make the extension more manageable in high use scenarios or where non-SQL data sources are used to load data into the on-demand app. The following sections describe properties that are organized in the "Advanced" section on the extension's property panel. These properties can remain unset and the extension will function with reasonable defaults (the default for each advanced property is described below). These advanced properties are arranged according to the layout shown below.



Control the generated app name

The **Generated app name** property is used to set the name of the generated app. If left unspecified, the name of the generated app will be the same as the template app's name but with an appended suffix of the form:

```
_username
```

where *username* is the identity of the current Qlik Sense user. When a value for **Generated app name** is specified, then this exact name is used for the generated app instead (no suffix is appended).

Control the maximum number of generated apps

When left at the default setting of 1, the **Max number of generated apps** property will, following a successful generation of a new on-demand app, check to see if there is more than 1 app with the same name as the newly generated app. If so, the oldest app with that name will be removed. This feature makes it possible to purge prior generated apps so that they do not accumulate in an uncontrolled manner. Note that if an error is encountered during the app generation process (e.g. a data connection fails) or if the user explicitly cancels the app generation process, then the generated app clean-up step is suppressed since the user may wish to return to a prior generated version. Also, the clean-up process only deletes at most one prior generated app in any given app generation cycle, regardless of the value for this property. The value only determines how many prior generations can accumulate prior to the automatic clean-up step to start to take effect.

Set the labels for buttons

The default label for the button to start the app generation process is constructed by adding the prefix “Get “ to the name of the on-demand app template. For example, if the on-demand app template for the sales example is named “Sales Details”, the default label for the app generation button will be

“Get Sales Details”. By setting the value of the **Generate app label** property, you can override this behavior and set the label to a string of your choice.

Once the app generation phase is complete, the status messages and progress animation are replaced with a new button for opening the generated app. The default label for this button is constructed by adding the prefix “Open “ to the name of the generated app. For example, if the On-demand App template for the sales example is named “Sales Details”, and the user has not entered a value for “Generated app name” property, the default label for the open app button will be “Open Sales Details_username”, where username is the name of the current user. By setting the value of the **Open app label** property, you can override this behavior and set the button label to a string of your choice.

Show the reload log

During the app generation process, the On Demand App Generation extension presents a short, continuously-updating message indicating the current phase of the process and an animated image indicating progress. You also have the option of viewing the status messages produced by the load script of the generated app while its data load process is underway. This option is disabled by default but can be enabled by selecting the **Show reload log** option in the advanced section of the property panel. It is also possible to toggle the data reload log panel in midstream by clicking on the small, radio checkbox located next to **Cancel**.

Control packing of bound field values

As described in Use selections in data selection app to control content of on-demand app, the default behavior for formatting the values of bound fields in the script of the on-demand app is to wrap the harvested values with single quotes and separate their values with commas. While this behavior works well for formulating SQL IN clauses, it may not be the best choice for other data sources. For example, if the On-demand app needs to query data using a REST API, it may be the case that the values of the bound fields need to be encoded into a URL string. In that case, you may wish to completely suppress wrapping the individual values and use a different value separator character. In order to suppress quote wrapping, you can enter the value of ‘none’ into the **Quote character** property. With the exception of the ‘none’ settings for the **Quote character** property, the extension will only use the first character of the settings for **Quote character** and **Value separator** and ignore any characters after the first character (e.g. using !! as the value of the **Value separator** property will result in only a single ! being used as the separator).

Technical details

This section provides technical documentation on the internals of the On Demand App Generation Extension for QAP developers who may wish to create their own similar extension using some of the same concepts demonstrated by the On Demand App Generation extension.

QAP APIs used

The On Demand App Generation extension makes use of the following Qlik Analytic Platform APIs:

- Extension API:
  - to implement the “paint” behavior for drawing the buttons and status messages
  - to establish the extension’s property panel configuration using the initialProperties object
  - to make use of the built-in support for AngularJS by returning the HTML template in the “template” property
- qlik.app capability API:
  - to identify the names of all the fields of the source app
  - to determine the selection state of the fields that are bound to expressions in the script of the target app
  - to count the number of selected (or optional and/or excluded) values of bound fields that have selection quantity constraints
- Engine API:
  - to find and open the on-demand template app
  - to access the script of the on-demand app template and identify the bind expressions
  - to create a copy of the template app (i.e. ‘the generated app’)
  - to set the script of the generated app so that all all bind expressions from the template app’s script have been updated to contain the desired selection state harvested from the data selection app
  - to save the generated app
  - to reload the generated app

Runtime view states

The visual layout of the On Demand App Generation extension varies depending on which of the following states the extension code determines is relevant at any given point in time:

No-Condition

One or more of the property values of the extension are improperly configured (e.g. measure expression, record limit, target app template name missing or invalid, no bindings in script of the target app template, binding expression error). In this state, one or more messages are shown to indicate the cause of the error and the generate app button is not shown.

Condition-Not-Met

The current, computed value of the Record Limit measure expression exceeds the Record Limit. In this state, the generate app button is shown but in an inactive state. A text message below the inactive button informs the user that too many records are qualified by the current selection state and that more (or a different) selection(s) must be made to reduce the total number of qualified records.

Qty-constraint-violation

The number of values in the selection state of one (or more) of the bound fields falls outside of the required range specified by a quantity constraint expression in the script of the target app. In this state, the generate app button is shown but in an inactive state. A text message below the button informs the user which fields need to have their selections modified to match their respective quantity constraints.

Ready

The number of records qualified, as computed by the record count measure expression, is less than or equal to the record limit value and there are no field binding quantity constraint violations. In this state, the generate app button is enabled and a message displays the number of detail records that will be loaded into the generated app if the button is clicked.

Processing

In this state, the generate app button is replaced with a “Cancel” button that can be used to cancel the app generation process. Below the “Cancel” button is a dynamically updating message showing the current phase of the app generation process. Below the status message is an animation graphic depicting the in-progress nature of the extension’s current runtime state. Next to the “Cancel” button is a radio box that can be used to replace the animation graphic with a scrolling text box containing the progress message generated by the app reload process.

Done

This is the view state that the extension transitions to when the app generation and data load process completes successfully. In this state, there is a single active button to open the generated app. Once the user clicks on that button, the state of the extension transitions back to the ‘Ready’ state. The ‘Done’ state is abandoned if the user changes the current selection state.

Error

This is the view state when the extension encounters any errors while generating the on-demand app or performing its data load operation. In this state, a text message containing a description of the error is shown in addition to an OK” button to allow the user to acknowledge the error and return to the ‘ready’ state.

Desktop

This state is set when the extension detects that it is running in Desktop mode. Since the extension does not function in desktop mode, this state presents a message informing the user and also suppresses the presentation of the ‘get data’ button.

Localizable strings

All of the strings used in the On Demand App Generation extension are localizable according to the localized language in the user’s web browser. As shipped, there is a single translation dictionary for the en-US locale but this translation dictionary can be expanded by adding support for new locales in the module xltns/propxltns.js. Whenever the extension fails to find a locale-specific dictionary matching the locale of the browser, the en-US locale dictionary will be used as a fall back.

Sample apps

This section describes how to use and adapt the on-demand app generation sample apps.
The sample apps provided use a publicly available dataset from the US department of transportation which contains flight data origins/destinations/carriers with passenger and distance metrics.
The link to download the data is:
http://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=289&DB_Short_Name=Origin%20and%20Destination%20Survey
Each dataset can be downloaded by quarter and the following entities were used in the samples:

Field name on page	Lookup tables
Origin Code	Code (ORIGIN)/Description (ORIGIN_AIRPORT)
ORIGIN_STATE_ABR	
Destination Code	Code (DEST)/Description (DEST_AIRPORT)
DEST_STATE_ABR	
Year	
Quarter	
Ticket Carrier Code (TICKET_CARRIER in script)	Code (TICKET_CARRIER)/Description (AIRLINE)
Fare Class Code	Description (FARE_CLASS_NAME)
Passengers	
Distance	

The following skillsets and software are required to deploy and modify this solution:

- Qlik scripting skillset.
- Qlik Sense Server with QMC access to import apps and extensions.

Script details

The Flight Data Shopper app contains the dimensions which will be incorporated into the subsequent WHERE clause for the detail app and may also contain a data summary view of the underlying source.
The Flight Data Detail app contains the script logic which interprets the dimensional selections and dynamically formulates the WHERE clause according to the selections made. The following sections describe the step-by-step instructions for incorporating the On Demand data bindings and WHERE clause construction logic into the Flight Detail app's load script.

Stage 1 – SET statements

Change #1 in the detail app contains a series of SET statements specify which dimensions would be involved in generating the resultant where clause:

Clearing any previous selections	`SET ORIGIN = ;`

Specifying column for on demand and set selection state: • ods – selected values • odo – associated values • odx -excluded values • odso – selected/associated	`SET ORIGIN = $(odso_ORIGIN);`
Specifying a column name for inclusion in where clause	`SET ORIGIN_COLNAME = 'ORIGIN';`
Optional parameters can also be specified to force a min/max number of selections which need to be made. The allowed patterns enclosed in the optional trailing square brackets are as follows: • <integer> : Meaning exactly that number of values needs to be selected • <integer>+ : Meaning at least that number of values must be selected • <integer>-<integer> (e.g. 2-4) : Meaning a number of values in that range of the two integers supplied.	`SET FARE_CLASS = $( odso_Fare Class Name)[2-4];`

Stage 2 – Connection string

Change #2 is the place to add the connection string and any dimensional load statements.

Stage 3 – WHERE clause

Change #3 allows an additional WHERE clause to be specified in addition to the dynamic WHERE clause generated by the on demand process:

```
SET WHERE_PART = '';
```

Stage 4 – Constructing the value list

Change #4 contains a FOR NEXT loop to construct the value list of the field selections and calls the ExtendedWhere subroutine to construct the WHERE clause:

Specifying the field list assigned to a variable (fldname)	`FOR EACH fldname IN 'ORIGIN', 'DEST'.....`

Specifying a new variable (vallist) to check length of the fldname list and invoking the ExtendWhere sub routine if >0	`LET vallist = $($fldnname);`
	
	`WHEN (IsNull(vallist)) LET vallist = '';`
	`IF len(vallist) > 0 THEN`
	`CALL ExtendWhere('$(fldname)','vallist');`

Stage 5 – Constructing the WHERE clause

The value list is passed into the ExtendWhere subroutine which constructs the WHERE clause to be applied to the Fact table SQL query:

Specifying input parameters for the sub routine: • Name = field name from the for loop • ValVarName = values from field	`SUB ExtendWhere(Name, ValVarName)`

Creates a variable (T) to construct the on demand field name with _COLNAME suffix and assigning ValVarName variable to the Values variable (this allows for the database column names to be different from their association Qlik fields).	`LET T = Name & '_COLNAME';`
	`LET ColName = $(T);`
	`LET Values = $(ValVarName);`

Checking if any (Values) are available and construct the where clause variable (WHERE_PART)	`IF len(Values) > 0 THEN`
	`  IF len(WHERE_PART) > 0 THEN`
	`  LET WHERE_PART = '$(WHERE_PART) AND $(ColName) IN ( $(Values) )';`
	`  ELSE`
	`  LET WHERE_PART = ' WHERE $(ColName) IN ( $(Values) )';`
	`  ENDIF`
	`ENDIF`

Stage 6 – Modifying SQL Select script with the generated WHERE clause

The final part of the process is to add/modify the SQL Select script to generate the appropriate SELECT statement to be sent to the source database including the dynamically generated WHERE clause from the $(WHERE_PART) variable. Additional SQL/LOAD statements can also be added with different dynamically generated WHERE clauses by replicating sections from the previous stages.

150 N. Radnor Chester Road
Suite E120
Radnor, PA 19087
Phone: +1 (888) 828-9768
Fax: +1 (610) 975-5987

qlik.com