Oracle® Big Data Discovery Cloud Service

Extensions Guide

E65372-04

September 2016

ORACLE

Oracle Big Data Discovery Cloud Service Extensions Guide,

E65372-04

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Contents

Preface

Oracle Big Data Discovery is a set of end-to-end visual analytic capabilities that leverage the power of Apache Spark to turn raw data into business insight in minutes, without the need to learn specialist big data tools or rely only on highly skilled resources. The visual user interface empowers business analysts to find, explore, transform, blend and analyze big data, and then easily share results.

About this guide

This guide describes how to use the Custom Visualization Component to develop unique visualizations for your particular business needs.

Audience

This guide is intended for developers who want to create custom components for Studio.

Conventions

The following conventions are used in this document.

Typographic conventions

The following table describes the typographic conventions used in this document.

Typeface	Meaning
User Interface Elements	This formatting is used for graphical user interface elements such as pages, dialog boxes, buttons, and fields.
`Code Sample`	This formatting is used for sample code segments within a paragraph.
Variable	This formatting is used for variable values. For variables within a code sample, the formatting is *`Variable`*.
`File Path`	This formatting is used for file names and paths.

Symbol conventions

The following table describes symbol conventions used in this document.

Symbol	Description	Example	Meaning
>	The right angle bracket, or greater-than sign, indicates menu item selections in a graphic user interface.	File > New > Project	From the File menu, choose New, then from the New submenu, choose Project.

Path variable conventions

This table describes the path variable conventions used in this document.

Path variable	Meaning
`$ORACLE_HOME`	Indicates the absolute path to your Oracle Middleware home directory, where BDD and WebLogic Server are installed.
`$BDD_HOME`	Indicates the absolute path to your Oracle Big Data Discovery home directory, `$ORACLE_HOME/BDD-<version>`.
`$DOMAIN_HOME`	Indicates the absolute path to your WebLogic domain home directory. For example, if your domain is named `bdd-<version>_domain`, then `$DOMAIN_HOME` is `$ORACLE_HOME/user_projects/domains/bdd-<version>_domain`.
`$DGRAPH_HOME`	Indicates the absolute path to your Dgraph home directory, `$BDD_HOME/dgraph`.

Contacting Oracle Customer Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. This includes important information regarding Oracle software, implementation questions, product and solution help, as well as overall news and updates from Oracle.

You can contact Oracle Customer Support through Oracle's Support portal, My Oracle Support at https://support.oracle.com.

1

Overview of the Custom Visualization Component

This section defines a Custom Visualization Component and describes how to develop a Custom Visualization Component for use in Studio.

About the Custom Visualization Component

A Custom Visualization Component is an extension to Studio that lets you create customized visualizations in cases where the default components in Studio do not meet your specific data visualization needs. A developer creates a custom component, tests it, modifies it, and publishes it to become available to business users. A business user then creates and configures an instance of the custom component on a project page in Studio.

Elements of a Custom Visualization Component

A Custom Visualization Component is made of the following:

- A JavaScript file to define the features, rendering, and interaction of the custom component with its data. You code this file to conform to the Custom Visualization Portlet JavaScript API.
- An EQL statement to provide one or more result sets for the component. This may include EQL token configuration to define variables in an EQL query.

You specify the JavaScript, the EQL, and the EQL token configuration on the **Custom Visualizations** page of Studio along with additional configuration of the custom component.

Installed libraries and external libraries

BDD has D3 version 3 and jQuery version 2.0.3 installed by default. If you need to access additional JavaScript libraries for use in your component, you specify them as a list of external JavaScript libraries when you create the component.

Role privileges

You must have Administrator privileges to access the **Custom Visualizations** page in Studio. Once you publish a Custom Visualization Component, any user with project access can create an instance of the custom component on a project page.

Reference API documentation

The Custom Visualization Portlet JavaScript API has generated JavaScript documentation that is available as part of the full BDD documentation set. You can use this documentation as a reference to code the JavaScript file for a custom component.

For details, see the *Custom Visualization Portlet JavaScript API Reference*.

Requirements for using the Custom Visualization Component

Before developing a Custom Visualization Component, make sure that you meet the following requirements.

Supported platforms

While Big Data Discovery is always deployed on a Linux system, you can develop a Custom Visualization Component on either a Windows or Linux system.

Required knowledge and skills

In order to work with a Custom Visualization Component, you should be familiar with the following:

- JavaScript development and charting libraries.
- Writing EQL statements to provide one or more result sets for a custom component. For details about writing EQL queries, see the *EQL Reference*.

Installing the Custom Visualization Component

No additional installation tasks are required.

Downloading the Custom Visualization Component Sample

On the Oracle Technology Network, you can download a ZIP file containing the Custom Visualization Component Sample.

The ZIP file contains a complete example of a custom component, including:

- A JavaScript file that illustrates how to use the Custom Visualization Component API. The sample code implements a Donut Pie chart visualization and the code provides a reference for building your own component.
- A text file of configuration settings that you use to populate fields on the **Custom Visualization** page of Studio. This provides the configuration settings for the sample JavaScript code and a tokenized EQL statement to generate results for the component.

2

Developing a Custom Visualization Component

This section describes how to create your own Custom Visualization Component and publish it for use in Studio.

Creating a Custom Visualization Component
: You create a component by coding a JavaScript file that uses the Custom Visualization Component JavaScript API to initiate queries and render the returned data to the component. You upload the file as part of the component configuration. Also you write one or more EQL statements to provide the result set for the component.

Using tokens in an EQL query
: The EQL queries for a Custom Visualization Component support token replacement in the EQL query. Tokens are simply variables in an EQL query that correspond to user-interface controls in the **Visualization Settings** panel the component. Controls include attributes (metrics or dimensions), views, data views and sorts. For example, a sort token in an EQL query creates an ASC or DSC sort control in the component configuration for a project user to select. A dimension token creates a drop down menu of attributes for a project user to select.

Editing JavaScript during development
: As a troubleshooting convenience, Studio provides an inline JavaScript editor so you can modify a component's JavaScript directly. You do not have to upload the file again using the Add Component wizard. You modify the JavaScript inline as part of debugging the component. This JavaScript editor is available only while a component is unpublished.

Publishing a Custom Visualization Component
: After you are satisfied that a Custom Visualization Component behaves as desired, you publish it to make it available on the Discover page of Studio to all Studio users. Unpublished components are available only to administrators.

Unpublishing a Custom Visualization Component
: To remove a Custom Visualization Component from the component menu, you unpublish it. Previously created instances of the component are still available on the Discover page, but business users cannot create new instances of the component after it has been unpublished.

Deleting a Custom Visualization Component
: Deleting removes the component from the component menu of the Discover page and removes the component from any project where it was used.

Creating a Custom Visualization Component

You create a component by coding a JavaScript file that uses the Custom Visualization Component JavaScript API to initiate queries and render the returned data to the component. You upload the file as part of the component configuration. Also you write one or more EQL statements to provide the result set for the component.

Before performing this task, you should code the JavaScript file for the custom component to conform to the Custom Visualization Component JavaScript API. You upload the file in the steps below. For details about coding the JavaScript file, see the *Custom Visualization Component JavaScript API Reference*.

When a business user creates a custom component on a project page in Discover, Studio loads the JavaScript code to render the component.

The JavaScript code has two major requirements:

- It must extend `Oracle.BDD.Portlets.Visualization.Renderers.BaseRenderer`.
- It must implement the `init()` function. This is executed on each Discover page load and it typically queries for a result set and directs the responds in the component.

To create a Custom Visualization Component:

1. In the Studio header, click the **Configuration Options** icon and select **Control Panel** > **Custom Visualizations**.
2. Click **+ Component**.
3. Specify a name for the component.

   This is the display name of the component as it displays on the Component menu of the **Discover** page.
4. Optionally, click **Browse** to locate an icon image for the component and then click **Upload** and **Ok**.

   This is the icon image for the component as it displays on the Component menu of the **Discover** page.
5. In **JavaScript File**, click **Browse** to locate the JavaScript file that implements your Custom Visualization Component and click **Open**.
6. In **Renderer class**, specify the fully qualified name of the JavaScript class that renders the component.

   For example, in the following JavaScript snippet, the name of the renderer class is `Oracle.BDD.Portlets.Visualization.Renderers.DonutPieChart`:

```
Oracle.BDD.Portlets.Visualization.Renderers.DonutPieChart =
Oracle.BDD.Portlets.Visualization.Renderers.BaseRenderer.extend({

    init: function() {

        /**
         * Get the queryConfig for the initial query
         */
```

```
        var queryConfig = this.getQueryConfig("eql");
...
```

7. Select a **Sort type** of one of the following:

   - **None** - Specifies that there is no sort option available for the business user to configure in the component.
   - **Defined** - Specifies that the sort option for the component is defined by token replacement in an EQL statement. A business user then sorts by value of the token in either an ascending or descending order.
   - **Per Dimension** - Specifies that a sort option is available for any token defined as a dimension.

8. Optionally, expand **Advanced Options** and specify the following:

   - **CSS** - Specifies visualization-specific CSS. Also note that the CSS is scoped to the visualization's DOM container.
   - **External CSS** - Specifies a list of URLs to external CSS files. Each URL should be line separated.
   - **External JavaScript** - Specifies a list of URLs to external JavaScript files. Each URL should be line separated. These JavaScript files might provide additional resources, such as third-party plug-ins, to support the JavaScript file.

   For example:

   ```
   https://cdnjs.cloudflare.com/ajax/libs/EventEmitter/4.2.11/EventEmitter.min.js
   https://cdnjs.cloudflare.com/ajax/libs/UAParser.js/0.7.9/ua-parser.min.js
   ```

9. Click **Next**.

10. In **EQL query name**, specify the name of the EQL query used in the JavaScript API function `getQueryConfig`.

    For example, if your Javascript file contains:

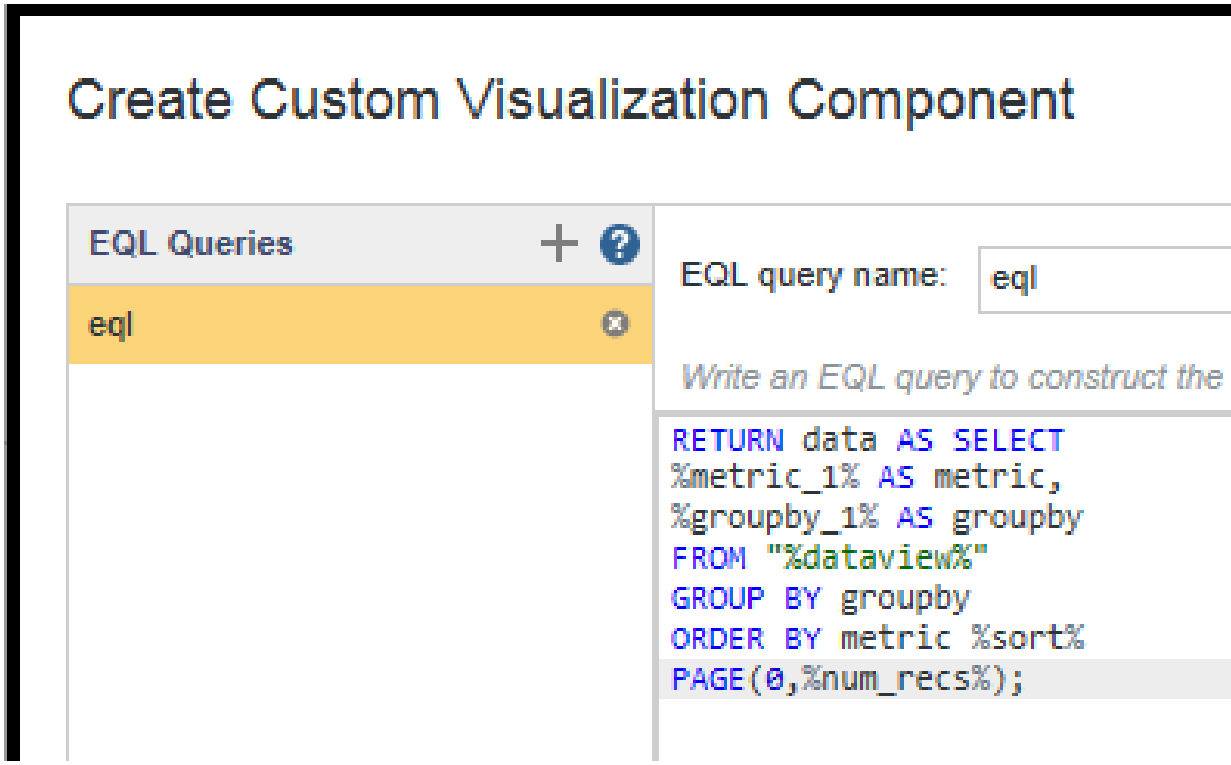
    ```
    var queryConfig = this.getQueryConfig("eql");

    ...
    ```

    then the EQL query name you specify is `eql`.

11. Specify the EQL query for your component in the text box.

    For example:



For more details about writing EQL statements, see the *EQL Reference*.

12. Optionally, you can add more EQL queries to the component by clicking **+** and specifying a new name and EQL query in the text box.

    Each query that you add generates an additional result set for the component to use.

13. Click **Detect Tokens**.

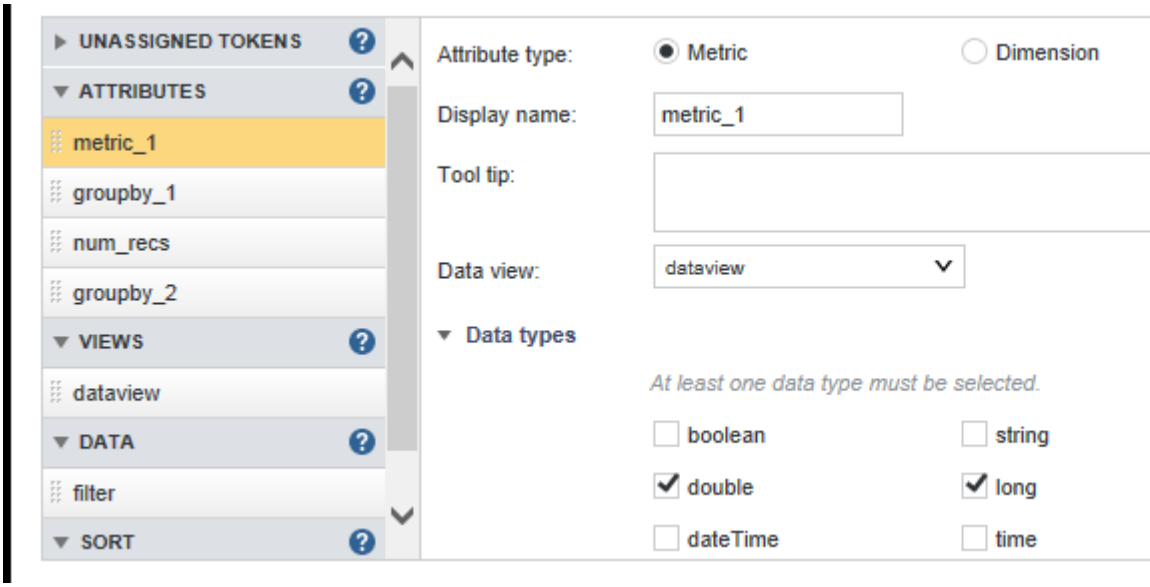
    Studio examines the tokens in the EQL query, and based on the syntax of how the tokens are used, Studio adds the tokens to **Attributes**, **Views**, **Data**, and **Sort** categories. For example, tokens in a `FROM` clause appear in **Views**. Tokens in a `WHERE` clause appear in **Data**.

14. If any of the tokens are incorrectly categorized, click the token and drag and drop it into the correct **Attributes**, **Views**, **Data**, or **Sort** category.

15. Click each token name and specify the details of how the token is used in the component.

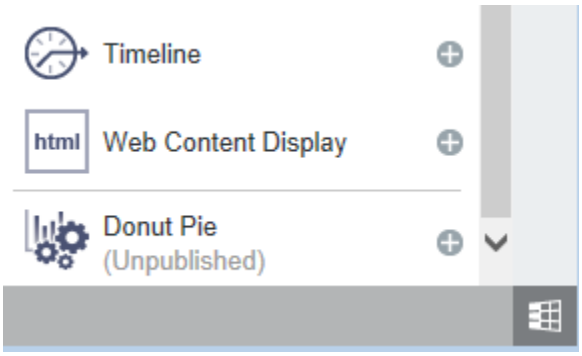
    This steps configures how a business user interacts with the token values on the Discover page. You specify whether a token is a metric or a dimension, its display name, its data view, its data type, and aggregation type.

    For example, the settings shown here configure the `metric_1` token:



16. Click **Save**.

The new component is available at the bottom of the **Add Component** menu on the **Discover** page. For example, here is the Donut Pie example used in the procedures:



The component is not available to business users until you publish it.

Before publishing, you should test the component by working with it as a business user would and adjusting the JavaScript or other configuration if necessary to modify the component's behavior. Once the component works correctly, you publish it for use by other Studio users.

Using tokens in an EQL query

The EQL queries for a Custom Visualization Component support token replacement in the EQL query. Tokens are simply variables in an EQL query that correspond to user-interface controls in the **Visualization Settings** panel the component. Controls include attributes (metrics or dimensions), views, data views and sorts. For example, a sort token in an EQL query creates an ASC or DSC sort control in the component configuration for a project user to select. A dimension token creates a drop down menu of attributes for a project user to select.

In EQL query, tokens are strings enclosed by percentage signs (%), for example, `%metric_1%`. Here is an example EQL statement that contains five tokens (`%metric_1%`, `%groupby_1%`, `%dataview%`, `%sort%`, `%num_recs%`):

```
RETURN data AS SELECT
%metric_1% AS metric,
%groupby_1% AS groupby
FROM "%dataview%"
GROUP BY groupby
ORDER BY metric %sort%
PAGE(0,%num_recs%);
```

Studio replaces the tokens with a value based on the user configuration in the **Visualization Settings** panel the component. That value is used when Studio runs the query to generate results for the component.

Token types

Tokens are distinguished by the type of data the token represents and by how the component acquires the token's value. The following table shows each token type and explains how Studio replaces the token with a value before running the EQL query.

Token type	Value	Example replacement value
Attribute (metric or dimension sub-types)	Attribute tokens represent either a metric or a dimension. A *metric* token represents menus of metric and aggregation functions. During component configuration, you select a data type for the metric token to determine which attributes are available for a project user to select. Similarly, during component configuration, you also select aggregation functions (e.g. SUM) to determine which attributes are available for a project user to select. A metric token is replaced when a project user selects an attribute and aggregation function from the **Visualization Settings** panel of the component. A *dimension* token represents a drop-down menu of attributes for a project user to select. During component configuration, you select a data type for the dimension token to determine which attributes are available for a project user to select. For example, if you select data type of string, only string attributes are available for selection. A dimension token is replaced when a user selects an attribute from the **Visualization Settings** panel of the component. Both types of metric and dimension tokens are associated with a View token that dictates which data view the attributes are taken from to populate the user-interface menus. Attribute token values may also be set with the JavaScript API.	SUM(p_price)

Token type	Value	Example replacement value
View	A view token represents a drop-down menu of data views for a project user to select. The token is replaced when a user selects a data view from the **Visualization Settings** panel of the component. View token values may also be set with the JavaScript API.	p_price
Sort	A sort token represents an ASC or DESC sort control in the component configuration for a project user to select. The token is replaced when a user selects a sort direction (ASC or DESC) from the **Visualization Settings** panel of the component. ASC is the default value. Sort token values may also be set with the JavaScript API.	ASC
Data	Data token values must be set with the JavaScript API.	

For additional details about tokens, see the *Custom Visualization Component JavaScript API Reference*.

Token substitution example

Here is an example EQL query with tokens:

```
RETURN data AS SELECT
%metric_1% AS metric,
%groupby_1% AS groupby
FROM "%dataview%"
GROUP BY groupby
ORDER BY metric %sort%
PAGE(0,%num_recs%);
```

Here is the same EQL query with values substituted for the tokens:

```
RETURN data AS SELECT
SUM(p_price) AS metric,
p_color AS groupby
FROM "wine_dataset"
GROUP BY groupby
ORDER BY metric ASC
PAGE(0,20);
```

Editing JavaScript during development

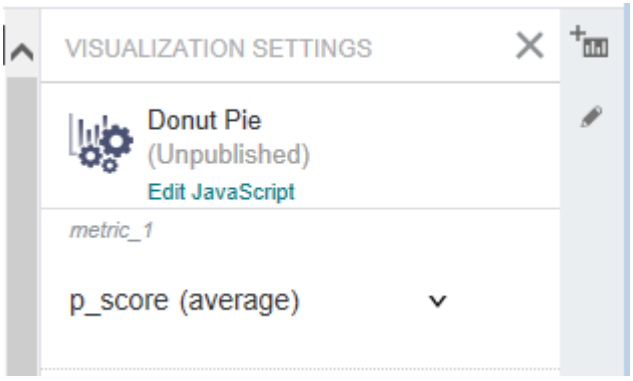
As a troubleshooting convenience, Studio provides an inline JavaScript editor so you can modify a component's JavaScript directly. You do not have to upload the file again using the Add Component wizard. You modify the JavaScript inline as part of debugging the component. This JavaScript editor is available only while a component is unpublished.

You must have already created a custom visualization component and added it to a project before you can edit the component's JavaScript.

To edit JavaScript during development:

1. Open a project and add your Custom Visualization Component to a Discover page.
2. Click the pencil icon to edit the visualization settings.
3. Click the **Edit JavaScript** link.

   For example:



4. In the Edit JavaScript editor, modify the component's code as necessary to adjust its behavior.
5. Click **Save and exit**.

The code changes take place immediately.

Publishing a Custom Visualization Component

After you are satisfied that a Custom Visualization Component behaves as desired, you publish it to make it available on the Discover page of Studio to all Studio users. Unpublished components are available only to administrators.

To publish a Custom Visualization Component:

1. In the Studio header, click the **Configuration Options** icon and select **Custom Visualizations**.
2. Locate the component you want to make available and click **Published**.

The component immediately becomes available on the component menu of the Discover page.

Unpublishing a Custom Visualization Component

To remove a Custom Visualization Component from the component menu, you unpublish it. Previously created instances of the component are still available on the Discover page, but business users cannot create new instances of the component after it has been unpublished.

An unpublished component is still stored in Studio, so a developer with the administrator role can modify it if necessary and publish it again later.

To unpublish a Custom Visualization Component:

1. In the Studio header, click the **Configuration Options** icon and select **Custom Visualizations**.

2. Locate the component that you want to make unavailable and deselect **Published**.
3. Click **Unpublish**.

Deleting a Custom Visualization Component

Deleting removes the component from the component menu of the Discover page and removes the component from any project where it was used.

To delete a Custom Visualization Component:

1. In the Studio header, click the **Configuration Options** icon and select **Custom Visualizations**.
2. Locate the component you want to remove and click **Remove**.
3. In the confirmation dialog, click **Delete**.

Index

C

E