Oracle® Analytics

Developer's Guide for Oracle Analytics Server

F92263-01
March 2025

Oracle Analytics Developer's Guide for Oracle Analytics Server,

F92263-01

Primary Author: Adam Donald

Contributing Authors: Stefanie Rhone, Hemala Vivek

Contributors: Oracle Analytics Server development, product management, and quality assurance teams

Contents

3 Create Oracle Analytics Visualization and Workbook Extensions

Preface

Learn how to develop and extend your Oracle Analytics instance with embedded content, and SDKs.

Topics:

Audience

This guide is intended for application developers and integrators who want to programmatically access and use the Oracle Analytics components to create applications or integrations with other components. You need to have knowledge of the following:

- Oracle Analytics Desktop
- Oracle Analytics Server

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documents

For a full list of guides, refer to the Books tab on Oracle Analytics Server Help Center.

- https://docs.oracle.com/en/middleware/bi/analytics-server/books.html

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Part I

Overview of Oracle Analytics Developer Resources

This part introduces you to the Oracle Analytics developer resources.

Topics:

1

Introduction to Oracle Analytics Server Developer Resources

Oracle allows you to develop and extend your Oracle Analytics Server products with REST APIs, custom extension plug-ins, and embedded content.

Developer Resources

Developer Resource	See
REST APIs	• Oracle Analytics Server REST APIs • REST APIs for Oracle Analytics Publisher in Oracle Analytics Server
Custom Extensions	Create and Manage Custom Extensions
Embedding content methods	Embed Content

Part II

Create and Manage Custom Extensions

This part explains how to create and manage custom data action, visualization, and workbook extensions.

Topics:

2

Create Custom Data Action Extensions

You can create custom data action extensions to use in Oracle Analytics.

Data action extensions extend Oracle Analytics and enable users to select data-points in visualizations and to invoke specific actions. Oracle Analytics provides a core set of data actions that cover many common use cases, but by writing your own data action extension, you can extend this functionality even further.

You must have a basic understanding of the following to create custom data action extensions:

- JavaScript
- RequireJS
- JQuery
- KnockoutJS

Topics:

About Data Action Extensions and the Data Actions Framework

Data action extensions leverage the data actions framework to provide custom, data-driven actions that are tightly integrated into the Oracle Analytics user interface.

When a user invokes a data action, the Data Action Manager passes the request context (for example, qualified data reference, measure values, filters and metadata) to the data action extension which is responsible for handling the request. Oracle provides four types of data action extensions: `CanvasDataAction`, `URLNavigationDataAction`, `HTTPAPIDataAction` and `EventDataAction`. You can extend these data action extension types along with their abstract base classes to provide your own data actions.

Topics:

Data Action Categories

The data action categories include Navigate to URL, HTTP API, Navigate to Canvas, and Event actions:

- **Navigate to URL**: Opens the specified URL in a new browser tab.
- **HTTP API**: Uses the `GET/POST/PUT/DELETE/TRACE` commands to target an HTTP API and doesn't result in a new tab. Instead the HTTP status code is examined and a transient success or failure message is displayed.
- **Navigate to Canvas**: Enables the user to navigate from a source canvas to a target canvas in either the same or a different visualization. Any filters that are in effect in the source canvas are passed to the target canvas as external filters. When the target canvas opens, it attempts to apply the external filters to the visualization. The mechanism by which external filters are applied isn't described here.
- **Event Actions**: Publishes an event using the Oracle Analytics event router. Any JavaScript code (for example, a third-party extension) can subscribe to these events and handle their custom response accordingly. This provides the maximum flexibility because the extension developer can choose how the data action responds. For example, they can choose to display a user interface or pass data to multiple services at once.

Both the **Navigate to URL** and **HTTP API** data action category types can use a token syntax to inject data or metadata from the visualization into the `URL` and `POST` parameters.

URL Token Replacement

HTTP data actions can replace tokens in URLs with values from the context passed to the data action. For example, qualified data reference values, filter values, username, workbook path, and canvas name.

Token	Notes	Replace With	Example	Result
`${valuesForColumn:COLUMN}`	NA	Column display values from the qualified data reference.	`${valuesForColumn:"Sales"."Products"."Brand"}`	`BizTech,FunPod`
`${valuesForColumn:COLUMN, separator:"/"}`	Any token that can potentially be replaced with multiple values supports the optional separator option. The `separator` defaults to a comma (,) but you can set it to any string. You can escape double quotes inside this string by using a backslash (\).	Column display values from the qualified data reference.	`${valuesForColumn:"Sales"."Products"."Brand"}`	`BizTech,FunPod`

Token	Notes	Replace With	Example	Result
`${valuesForColumn:COLUMN, separationStyle:individual}`	Any `separationStyle` defaults to `delimited` but you can set it to `individual` if the user needs to generate separate URL parameters for each value.	Column display values from the qualified data reference.	`&myParam=${valuesForColumn:"Sales"."Products"."Brand"}`	`&myParam=BizTech&myParam=FunPod`
`${keyValuesForColumn:COLUMN}`	NA	Column key values from the qualified data reference.	`${keyValuesForColumn:COLUMN}`	`10001,10002`
`${env:ENV_VAR}`	Supported environment variables are: `sProjectPath`, `sProjectName`, `sCanvasName`, `sUserID`, and `sUserName`.	An environment variable.	`${env:'sUserID'}`	`myUserName`

Data Action Context

You can define a context that is passed when the user invokes a data action.

You define how much of the context is passed to the data action when you create the data action.

Qualified Data Reference

When the data action is invoked a qualified data reference is generated for each marked data point using an array of `LogicalFilterTree` objects. A `LogicalFilterTree` consists of multiple `LogicalFilterNode` objects arranged in a tree structure. This object includes:

- The attributes on the row or column edges of the data layout.
- The specific measure on the measure edge that addresses each marked cell.
- The specific measure value for each marked cell.
- Key values and display values.

Environment Variables

In addition to the data and metadata describing each marked data point, certain data actions may need further context describing the environment from where the data action is invoked. Such environment variables include:

- Project Path
- Project Name
- Canvas Name
- User ID
- User Name

Data Action Code Design

You create data actions using API classes.

- There are four concrete classes of data action that inherit from the `AbstractDataAction` class:
  - `CanvasDataAction`
  - `URLNavigationDataAction`
  - `HTTPAPIDataAction`
  - `EventDataAction`
- You can create new types of data actions using the data action extension API.
- The registry of data action types is managed by the `DataActionPluginHandler`.
- Code that creates, reads, edits, deletes, or invokes instances of data actions does so by publishing events.
- Events are handled by the `DataActionManager`.

Data Action Model Classes

There are several different types of data action model classes.

AbstractDataAction

This class is responsible for:

- Storing the Knockout Model (subclasses are free to extend this with their own properties).
- Defining the abstract methods that subclasses must implement:
  - `+ invoke(oActionContext: ActionContext, oDataActionContext:DataActionContext) <<abstract>>`
    Invokes the data action with the passed context - should only be called by the `DataActionManager`.
  - `+ getGadgetInfos(oReport): AbstractGadgetInfo[] <<abstract>>`
    Constructs and returns the `GadgetInfos` responsible for rendering the user interface fields for editing this type of data action.
  - `+ validate() : DataActionError`
    Validates the data action and returns null if valid or a `DataActionError` if it's invalid.
- Providing the default implementation for the following methods used to render generic parts of the data action user interface fields:
  - `+ getSettings():JSON`
    Serializes the data action's Knockout Model to JSON ready to be included in the report (uses komapping.toJS(_koModel)).
  - `+ createNameGadgetInfo(oReport) : AbstractGadgetInfo`
    Constructs and returns the `GadgetInfo` that can render the data action's **Name** field.
  - `+ createAnchorToGadgetInfo(oReport) : AbstractGadgetInfo`
    Constructs and returns the `GadgetInfo` that can render the data action's **Anchor To** field.
  - `+ createPassValuesGadgetInfo(oReport) : AbstractGadgetInfo`

Constructs and returns the `GadgetInfo` that can render the data action's **Pass Values** field.

Subclasses may not need all of the `GadgetInfo`s that the base class provides so they may not need to call all of these methods. By separating out the rendering of each field in this way, subclasses are free to pick and choose the gadgets they need. Some subclasses may even choose to provide a different implementation of these common data action gadgets.

CanvasDataAction, URLNavigationDataAction, HTTPAPIDataAction, EventDataAction

These are the concrete classes for the basic types of data actions. These classes work by themselves to provide the generic user interface for these types of data action. They can also act as convenient base classes for custom data action plug-ins to extend.

- **CanvasDataAction**: Used to navigate to a canvas.
- **URLNavigationDataAction**: Used to open a web page in a new browser window.
- **HTTPAPIDataAction**: Used to make a `GET/POST/PUT/DELETE/TRACE` request to an HTTP API and handle the `HTTP Response` programatically.
- **EventDataAction**: Used to publish JavaScript events through the Event Router.

Each class is responsible for:

- Implementing the abstract methods from the base class.
  - `invoke(oActionContext: ActionContext, oDataActionContext:DataActionContext)`
    This method should invoke the data action by combining the properties defined in the KOModel with the specified `DataActionContext` object.
  - `getGadgetInfos(oReport): AbstractGadgetInfo[]`
    This method should:
    * Create an array containing `AbstractGadgetInfos`.
    * Call individual `createXXXGadgetInfo()` methods pushing each `AbstractGadgetInfo` into the array.
    * Return the array.
- Providing the additional methods for creating the individual gadgets that are specific to the particular subclass of data action.

Subclasses of these concrete classes may not need to use all of the gadgets provided by their superclasses in their custom user interfaces. By separating out the construction of each gadget in this way, subclasses are free to pick and choose the gadgets they need.

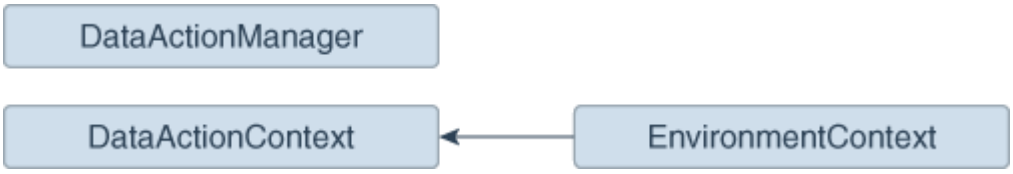
DataActionKOModel, ValuePassingMode

The `DataActionKOModel` class provides the base KOModel shared by the different subclasses of `AbstractDataAction`. See DataActionKOModel Class.

Data Action Service Classes

There are several different data action service classes.

DataActionManager



All communication with `DataActionManager` uses `ClientEvents.DataActionManager` which implements event handlers for:

- Managing the set of data actions defined in the current workbook.
- Invoking a data action.
- Retrieving all the data actions defined in the current workbook.
- Retrieving all the data actions that are applicable to the current marked data points.

DataActionContext, EnvironmentContext

When a data action is invoked, the `DataActionContext` class contains the context that's passed to the target.

- `getColumnValueMap()`
  Returns a map of attribute column values keyed by attribute column names. These define the qualified data reference for the data points that the data action is invoked from.
- `getLogicalFilterTrees()`
  Returns a `LogicalFilterTrees` object describing the qualified data references for the specific data points that the data action is invoked from (see the `InteractionService` for details).
- `getEnvironmentContext()`
  An instance of the `EnvironmentContext` class describing the source environment such as:
  - `getProjectPath()`
  - `getCanvasName()`
  - `getUserID()`
  - `getUserName()`
- `getReport()`
  Returns the report that the data action is invoked from.

DataActionHandler

The `DataActionHandler` class registers the various data action extensions. Its API is broadly consistent with the other extension handlers (for example, `VisualizationHandler`).

The `DataActionHandler` class provides the following public methods:

- `getClassName(sPluginType:String) : String`
  Returns the fully qualified class name for the specified data action type.
- `getDisplayName(sPluginType:String) : String`
  Returns the translated display name for the specified data action type.
- `getOrder(sPluginType:String) : Number`
  Returns a number used to sort lists of the types of data action into the preferred order.

The `DataActionHandler` class provides the following static methods:

- `getDependencies(oPluginRegistry:Object) : Object.<String, Array>`
  Returns a dependency map covering all the registered data action types.
- `getHandler(oPluginRegistry:Object, sExtensionPointName:String, oConfig:Object) : DataActionPluginHandler`
  Constructs and returns a new instance of the `DataActionHandler` class.

DataActionUpgradeHandler

The `DataActionUpgradeHandler` class is called by the `UpgradeService` when a report is opened.

The `DataActionHandler` class provides two main methods:

- `deferredNeedsUpgrade(sCurrentVersion, sUpgradeTopic, oDataActionJS, oActionContext) : Promise`
  Returns a `Promise` that resolves to a Boolean indicating whether the specified data action must be upgraded (`true`) or not (`false`). The method decides whether the data action must be upgraded by comparing the data action instance with the data action's constructor.
- `performUpgrade(sCurrentVersion, sUpgradeTopic, oDataActionJS, oActionContext, oUpgradeContext) : Promise`
  Carries out the upgrade on the specified data action and resolves the `Promise`. The upgrade itself is carried out by calling the `upgrade()` method on the data action (only the specific subclass of data action being upgraded is qualified to upgrade itself).
- `getOrder(sPluginType:String) : Number`
  Returns a number used to sort lists of the types of data action into the preferred order.

Data Action Code Interactions

A data action interacts with Oracle Analytics code when it creates a user interface field, and when a user invokes a data action.

Create the Field for a New Data Action Instance

This interaction starts when Oracle Analytics wants to render a data action user interface field. To do so, it:

1. Creates a `PanelGadgetInfo` that acts as the parent `GadgetInfo` for the `GadgetInfos` that the data action returns.
2. Calls `getGadgetInfos()` on the data action.
3. Adds the data action's `GadgetInfos` as children of the `PanelGadgetInfo` created in the first step.
4. Creates the `PanelGadgetView` that renders the `PanelGadgetInfo`.
5. Sets the `HTMLElement` that's the container of the `PanelGadgetView`.

6. Registers the `PanelGadgetView` as a child `HostedComponent` of a `HostedComponent` that's already attached to the `HostedComponent` tree.
   This renders the data action's gadgets inside the Panel gadget in the order they appear in the array returned by `getGadgetInfos()`.

Invoke a Data Action

This interaction starts when the user invokes a data action through the Oracle Analytics user interface (for example, from the context menu on a data point in a visualization).

In response to the user interaction, the code:

1. Publishes an `INVOKE_DATA_ACTION` event containing the data action's ID, the `DataVisualization` that the data action is invoked from, and a `TransientVizContext` object.
2. The `DataActionManager` handles this event by:
   a. Obtaining the data action instance from its ID.
   b. Obtaining the `LogicalFilterTrees` for the marked data points in the specified `DataVisualization`.
   c. Constructing a `DataActionContext` that contains all the information to pass to the data action's target.
   d. Calling `invoke(oDataActionContext)` on the data action.

Example Data Action plugin.xml File

This topic shows an example plugin.xml file for a `CanvasDataAction` data action.

Example plugin.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:obiplugin xmlns:tns="http://plugin.frameworks.tech.bi.oracle"
               xmlns:viz="http://plugin.frameworks.tech.bi.oracle/extension-
points/visualization"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               id="obitech-currencyconversion"
               name="Oracle BI Currency Conversion"
               version="0.1.0.@qualifier@"
               optimizable="true"
               optimized="false">


   <tns:resources>
      <tns:resource id="currencyconversion" path="scripts/
currencyconversion.js" type="script" optimizedGroup="base"/>
      <tns:resource-folder id="nls" path="resources/nls" optimizable="true">
         <tns:extensions>
            <tns:extension name="js" resource-type="script"/>
         </tns:extensions>
      </tns:resource-folder>
   </tns:resources>


   <tns:extensions>
```

```
      <tns:extension id="oracle.bi.tech.currencyconversiondataaction" point-
id="oracle.bi.tech.plugin.dataaction" version="1.0.0">
         <tns:configuration>
         {
            "resourceBundle": "obitech-currencyconversion/nls/messages",
            "properties":
            {
               "className": "obitech-currencyconversion/
currencyconversion.CurrencyConversionDataAction",
               "displayName": { "key" : "CURRENCY_CONVERSION", "default" :
"Currency Conversion" },
               "order": 100
            }
         }
         </tns:configuration>
      </tns:extension>
   </tns:extensions>


</tns:obiplugin>
```

Data Action Extension Files and Folders

The following files and folders are used to implement data action extensions.

`bitech/client/plugins/src/`

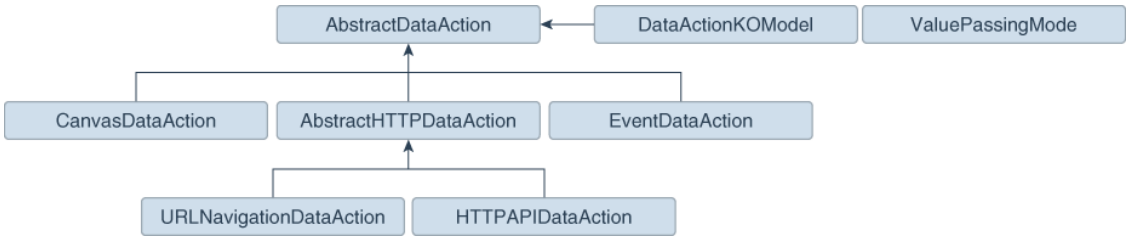
- `report`
  - `obitech-report`
    - `scripts`
      - `dataaction`
        - `dataaction.js`
        - `dataactiongadgets.js`
        - `dataactionpanel.js`
        - `dataactionupgradehandler.js`
- `obitech-reportservice`
  - `scripts`
    - `dataaction`
      - `dataactionmanager.js`
      - `dataactionhandler.js`

Choose the Best Data Action Class to Extend

Before you start writing your custom data action extension, decide which of the existing data action classes you want to extend. Choose the data action class that provides functionality that most closely matches what you want your data action to do.

Each data action inherits from the `AbstractDataAction` class as shown in the class diagram. The class diagram shows the two abstract data action classes (`AbstractDataAction` and

`AbstractHTTPDataAction`) and the four concrete data action classes (`CanvasDataAction`, `URLNavigationDataAction`, `HTTPAPIDataAction`, and `EventDataAction`) that you can extend. Each data action that you provide must extend one of these classes. Which class you extend depends on the behavior you want to implement when you invoke your data action. Most third-party data actions are likely to extend either `URLNavigationDataAction`, `HTTPAPIDataAction` or `EventDataAction`.



Regardless of which class you extend, when your data action is invoked, you're provided with metadata describing the full context of the data-point from which the data action is invoked. See Data Action Context.

AbstractDataAction Class

`AbstractDataAction` is the abstract base class from which all types of data action inherit. It's responsible for providing common functionality and default behavior that the subclasses can use.

AbstractDataAction

All types of data action are subclasses of the `AbstractDataAction` base class. It provides the core set of functionality common to all data actions. Unless you're creating a complex data action that carries out multiple types of action when invoked, or you need to do something not supported by the concrete classes, you shouldn't extend this class directly. If you need to create a complex data action then consider extending the concrete class that most closely provides the functionality you require.

AbstractDataAction Syntax

```
+ AbstractDataAction(oKOModel)


+ getKOViewModel():DataActionKOModel


+ createFromJS(fDataActionConstructor, sClassName, oDataActionKOModelUS) :
AbstractDataAction


+ invoke(oActionContext, oDataActionContext)
+ getGadgetInfos(oReport) : AbstractGadgetInfo[]
+ validate() : DataActionError


+ getSettings() : Object
+ requiresActionContextToInvoke() : Boolean
+ isAllowedHere() : Boolean


# createNameGadgetInfo(oReport) : AbstractGadgetInfo
# createAnchorToGadgetInfo(oReport) : AbstractGadgetInfo
# createPassValuesGadgetInfo(oReport) : AbstractGadgetInfo
```

DataActionKOModel Class

Each subclass of `AbstractDataAction` is likely to create its own subclass of `DataActionKOModel`. The `DataActionKOModel` base class provides the following properties:

DataActionKOModel, ValuePassingMode

- `sID:String`
  The unique ID given to the data action instance.
- `sClass:String`
  The class name of this specific type of data action.
- `sName:String`
  The display name given to the data action instance.
- `sVersion`
- `sScopeID`
- `eValuePassingMode:ValuePassingMode`
  The mode used when passing context values. The mode can be one of the `ValuePassingMode` values (`ALL`, `ANCHOR_DATA`, `NONE`, `CUSTOM`).
- `aAnchorToColumns: ColumnKOViewModel[]`
  The columns that this data action is anchored to. This is optional. If not supplied, then the data action is available on all columns.
- `aContextColumns : ColumnKOViewModel[]`

The columns that this data action includes in the context passed to the data action target when the data action is invoked. If not supplied, all marked columns are included in the context.

CanvasDataAction Class

`CanvasDataAction` is a subclass of the `AbstractDataAction` base class. You can extend this concrete class to provide the functionality you require.

CanvasDataAction

Use the `CanvasDataAction` class to navigate from a data point in a visualization to a different canvas. The canvas you're navigating to can be in the same workbook or a different one. All the active filters for the source visualization are passed to the target canvas along with new filters that describe the Qualified Data Reference of the data point itself. If your data action needs to navigate to a different canvas then this is the class your data action should extend.

```
+ CanvasDataAction(oKOModel)



+ create(s)ID_sName) : CanvasDataAction
+ upgrade(oOldDataActionJS) : Object



+ invoke(oActionContext: ActionContext, oDataActionContext:DataActionContext)
+ getGadgetInfos(oReport) : AbstractGadgetInfo[]
+ validate() : DataActionError


# createProjectGadgetInfo(oReport) : AbstractGadgetInfo
# createCanvasGadgetInfo(oReport) : AbstractGadgetInfo
```

EventDataAction Class

`EventDataAction` is a subclass of the `AbstractDataAction` base class. You can extend this concrete class to provide the functionality you require.

EventDataAction

Use the `EventDataAction` class to publish a client-side event. You can then register one or more subscribers that listen for that event and perform their own actions. Use this type of data action in more complex use cases where you've a large amount of code and can benefit from

keeping your data action code loosely coupled to the code that performs the necessary actions when the data action is invoked.

```
+ EventDataAction(oKOModel)



+ create(sID_sName) : EventDataAction
+ upgrade(oOldDataActionJS) : Object


+ invoke(oActionContext: ActionContext, oDataActionContext:DataActionContext)
+ getGadgetInfos(oReport) : AbstractGadgetInfo[]
+ validate() : DataActionError


# createEventGadgetInfo(oReport) : AbstractGadgetInfo
```

AbstractHTTPDataAction Class

`AbstractHTTPDataAction` is the abstract base class that the `URLNavigationDataAction` and `HTTPAPIDataAction` subclasses inherit common functionality and default behavior from.

AbstractHTTPDataAction

The `AbstractHTTPDataAction` abstract base class is shared by both the `URLNavigationDataAction` and `HTTPAPIDataAction` classes. If your data action needs to open a web page in a new browser tab you must extend `URLNavigationDataAction`. If your data action needs to invoke an HTTP API then you should extend `HTTPAPIDataAction`. You may decide it's better to extend `AbstractHTTPDataAction` directly.

```
+ HTTPDataAction(oKOModel)


+ validate() : DataActionError


# createURLGadgetInfo(oReport) : AbstractGadgetInfo
```

URLNavigationDataAction Class

`URLNavigationDataAction` is a subclass or the `AbstractHTTPDataAction` base class.

URLNavigationDataAction

Use the `URLNavigationDataAction` class to open a specific URL in a new browser tab. You compose the URL using tokens that are replaced with values derived from data points that the user selects when they invoke the data action. The data point values are passed as part of the data action context to the external web page. For example, create a data action invoked using

a `CustomerID` column that opens a customer's web page in your Customer Relations Management application such as Oracle Sales Cloud.

```
+ URLNavigationDataAction(oKOModel)


+ create(sID_sName) : URLNavigationDataAction
+ upgrade(oOldDataActionJS) : Object


+ invoke(oActionContext: ActionContext, oDataActionContext:DataActionContext)
+ getGadgetInfos(oReport) : AbstractGadgetInfo[]
```

HTTPAPIDataAction Class

`HTTPAPIDataAction` is a subclass or the `AbstractHTTPDataAction` base class. You can extend this concrete class to provide the functionality you require.

HTTPAPIDataAction

Use the `HTTPAPIDataAction` class to invoke HTTP APIs by creating an asyncronous `XMLHTTPRequest` (XHR) and submitting it to the specified URL. The HTTP response code enables a message to be displayed briefly on the canvas. For example, you can customize the request to send JSON or XML payloads to a REST or SOAP server and you can customize the response handler to show a custom user interface.

For the `HTTPAPIDataAction` data action to work, you must add the URL of the HTTP API you want to access to your list of Safe Domains and grant it **Connect** access. See Register Safe Domains.

```
+ HTTPAPIDataAction(oKOModel)


+ create(sID_sName) : HTTPAPIDataAction
+ upgrade(oOldDataActionJS) : Object


+ invoke(oActionContext: ActionContext, oDataActionContext:DataActionContext)
+ getGadgetInfos(oReport) : AbstractGadgetInfo[]


# createHTTPMethodGadgetInfo(oReport) : AbstractGadgetInfo
# createPostParamGadgetInfo(oReport) : AbstractGadgetInfo
```

Generate Data Action Extensions from a Template

You use a series of commands to generate a development environment and populate it with a HTTP API Data Action along with the necessary folders and files that you need to create a custom data action extension.

All extensions files follow the same basic structure. You can manually create the files and folders or you can generate them from a template. The tools to do this are part of the Oracle Analytics Desktop software development kit (SDK) which is included with Oracle Analytics Desktop.

Use these commands to generate your development environment and populate it with a HTTP API data action.

1. At a command prompt, specify the root folder of your Oracle Analytics Desktop installation:

   ```
   set DVDESKTOP_SDK_HOME=C:\Program Files\Oracle Analytics Desktop
   ```

2. Specify the location to store your custom extensions:

   ```
   set PLUGIN_DEV_DIR=C:\temp\dv-custom-plugins
   ```

3. Add the SDK command line tools to your path using:

   ```
   set PATH=%DVDESKTOP_SDK_HOME%\tools\bin;%PATH%
   ```

4. Create a folder for the directory used to store the custom extensions using:

   ```
   mkdir %PLUGIN_DEV_DIR%
   ```

5. Change the directory to the folder for storing custom extensions:

   ```
   cd %PLUGIN_DEV_DIR%
   ```

6. Create the environment variables:

   ```
   bicreateenv
   ```

7. Create the template files needed to start developing a custom HTTP API data action, for example:

   ```
   bicreateplugin -pluginxml dataaction -id company.mydataaction -subType httpapi
   ```

   Use the `-subType` option to specify the data action type that you want to create from: `httpapi`, `urlNavigation`, `canvasNavigation`, `event`, or `advanced`. The `advanced` option extends from the `AbstractDataAction` base class.

Generated Folders and Files

Your newly generated data action development environment contains these folders and files:

```
1    %PLUGIN_DEV_DIR%\src\customdataaction
2       company-mydataaction\
3          extensions\
4             oracle.bi.tech.plugin.dataaction\
5                company.mydataaction.json
6          nls\
7             root\
8                messages.js
9             messages.js
10          mydataaction.js
11          mydataactionstyles.css
12          plugin.xml
```

- **Line 2**: The `company-mydataaction` folder is the ID that you specify.
- **Line 6**: The `nls` folder contains the files for externalizing strings that enable your extension to provide Native Language Support.
- **Line 7**: The strings in the files under the `nls\root` folder are the default strings used when translations for a requested language aren't available.

- **Line 8**: The `messages.js` file contains externalized strings for your extension that you can add.
- **Line 9**: The `messages.js` file must contain an entry that you add for each additional language that you want to provide localized strings for. You must add a corresponding folder under the `nls` folder for each locale that you want to add translations for. Each folder must contain the same set of files, with the same file names as those added under the `nls\root` folder.
- **Line 10**: The `mydataaction.js` file is the newly generated JavaScript module template that provides a starting point to develop your custom data action.
- **Line 11**: The `mydataactionstyles.css` file can contain any CSS styles that you want to add, and which your data action's user interface can use.
- **Line 12**: The `plugin.xml` file registers your extension and its files with Oracle Analytics.

Extend a Data Action Base Class

Once you've chosen the subclass of data action that you want to extend and have generated the necessary folders and files, you're ready to start writing the code specific to your new data action.

You can find your newly generated data action code under `%PLUGIN_DEV_DIR%\src\dataaction`. See Generated Folders and Files for an explanation of the files and folder structure. The main file you must edit is the JavaScript file. For example, if your custom data action ID is `company.MyDataaction`, then the file you're looking for is `%PLUGIN_DEV_DIR%\src\dataaction\company-mydataaction\mydataaction.js`.

Extending Your Data Action's Knockout Model
If your data action has additional properties that need to be stored, then you must add them as observable properties to the Knockout Model. If your data action is given the ID `company.MyDataaction`, then the Knockout Model is called `mydataaction.MyDataActionKOModel` which is located near the top of `mydataaction.js`. By default, this Knockout Model is configured to extend the Knockout Model used by your data action's superclass so you only need to add additional properties to the model.

For a data action that's extending the `HTTPAPIDataAction` base class, use code similar to the following:

```
1 - mydataaction.MydataactionKOModel = function (sClass, sID, sName,
sVersion, sScopeID, aAnchorToColumns, eValuePassingMode, sURL,
        eHTTPMethod, sPOSTParams)
2 - {
3 - mydataaction.MydataactionKOModel.baseConstructor.call(this, sClass, sID,
sName, sVersion, sScopeID, aAnchorToColumns, eValuePassingMode, sURL,
eHTTPMethod, sPOSTParams);
4 - };
5 - jsx.extend(mydataaction.MydataactionKOModel,
dataaction.HTTPAPIDataActionKOModel);
```

- **Line 1**: This is the constructor for your Knockout Model. It accepts the properties that the model needs to store.
- **Line 3**: This is the superclass's constructor, otherwise known as the `baseConstructor` to which you pass the values for all of the properties that are handled by one of the Knockout Model's superclasses.

- **Line 5**: This sets the superclass for this Knockout Model class.

Use code similar to the following to add a string and an array to set properties that are persisted by the data action.

```
1   mydataaction.MydataactionKOModel = function (sClass, sID, sName,
sVersion, sScopeID, aAnchorToColumns, eValuePassingMode, sURL, eHTTPMethod,
sPOSTParams)
2   {
3   mydataaction.MydataactionKOModel.baseConstructor.call(this, sClass, sID,
sName, sVersion, sScopeID, aAnchorToColumns, eValuePassingMode, sURL,
eHTTPMethod, sPOSTParams);
4
5
6   // Set Defaults
7   sMyString = sMyString || "My default string value";
8   aMyArray = aMyArray || [];
9
10
11  // Asserts
12  jsx.assertString(sMyString, "sMyString");
13  jsx.assertArray(aMyArray, "aMyArray");
14
15
16  // Add observable properties
17  this.sMyString = ko.observable(sMyString);
18  this.aMyArray = ko.observableArray(aMyArray);
19  };
20  jsx.extend(mydataaction.MydataactionKOModel,
dataaction.HTTPAPIDataActionKOModel);
```

Choose Which Data Action Inherited Methods to Override

Each data action must implement various methods in order to function properly, so you only need to override those methods that implement behavior that you want to change.

If you're extending one of the concrete data actions classes, for example `HTTPAPIDataAction`, then most of the required methods are already implemented and you only need to override the methods that implement the behavior you want to change.

Generic Methods

This section describes the various methods and what's expected of them.

All types of data action must implement the methods that are described here.

create(sID, sName)

The `create()` static method is called when you're creating a new data action and select a **Data Action Type** from the drop-down menu. This method is responsible for:

- Constructing the Knockout Model class that your data action uses.
  The Knockout Model class must have the ID and name that's passed to the `create()` method along with sensible defaults for all other properties. For example, for a currency conversion data action you might want to set the default currency to convert into Dollars. The Knockout Model is the correct place to provide your default values.

- Constructing an instance of your data action from the Knockout Model.
- Returning the instance of your data action.

invoke(oActionContext, oDataActionContext)

The `invoke()` method is called when the user invokes your data action from the context menu for a data point in a visualization. The method passes the `DataActionContext` argument which contains metadata describing the selected data points, visualization, filters, workbook, and session. See Data Action Service Classes.

validate()

The `validate()` method is called on each data action when the user clicks **OK** in the Data Actions dialog. The `validate()` method returns a `null` to indicate that everything is valid or a `DataActionError` if something is invalid. If there's an error in one of the data actions in the dialog, the error prevents the dialog from closing and an error message is displayed to the user. This method validates the name of the data action using the `this.validateName()` method.

getGadgetInfos(oReport)

The `getGadgetInfos()` method is called to enable the user interface to display data action property fields. The method returns an array of `GadgetInfos` in the order you want them to appear in the user interface. Gadgets are provided for all of the most common types of fields (for example, text, drop-down, password, multi-select, radio button, check box) but you can create custom gadgets if you want more complicated fields (for example, where multiple gadgets are grouped together, or where different gadget fields display depending on which option you select). It's a best practice to create a method that constructs each `GadgetInfo` you want in your array, as it makes it easier for potential subclasses to pick and choose from the `GadgetInfo`s you've provided. If you follow this best practice there are already various methods implemented by the different data action base classes that can return a `GadgetInfo` for each of the fields that they use in their user interfaces. If you also need one of these `GadgetInfo`s then you call the corresponding `create****GadgetInfo()` method and push its return value into your array of gadgets.

isAllowedHere(oReport)

The `isAllowedHere()` method is called when the user right-clicks on a data-point in a visualization and the user interface starts to generate the context menu. If a data action exists that's relevant to the selected data-points, then the method returns `true` and the data action appears in the context menu. If the method returns `false`, then the data action doesn't appear in the context menu. Consider accepting the default behavior inherited from the superclass.

upgrade(oOldDataActionJS)

If you're creating your first data action then don't use the `upgrade(oOldDataActionJS)` method. Only use this method after you've created your first Knockout Model and are making significant changes to properties for a second version of your Knockout Model. For example, if the first version of your data action stores a URL in its Knockout Model, but you decide that the next version will store URL component parts in separate properties (for example, `protocol`, `hostname`, `port`, `path`, `queryString` and `bookmark`).

The second version of your Knockout Model code would request to open a data action that had been saved with the first version of your Knockout Model code which can cause problems. To resolve this issue, the system identifies that your current data action code version is newer than that of the data action being opened and it calls the `upgrade()` method on your new data action class and passes in the old data action Knockout Model (serialized to a JSON object). You can then use the old JSON object to populate your new Knockout Model and return an

upgraded version of the JSON object. This ensures that old data action metadata continues to work as you improve your data action code.

HTTPAPIDataAction Methods

If you're extending the `HTTPAPIDataAction` class, then it provides the following additional method that you may choose to override:

getAJAXOptions(oDataActionContext)

The `getAJAXOptions()` method is called by the data action's `invoke()` method. The `getAJAXOptions()` method creates the `AJAX Options` object that describes the HTTP request that you want your data action to make. The `getAJAXOptions()` method is passed the `oDataActionContext` object that contains the metadata describing the selected data-points, visualization, filters, workbook, and session. Set the `AJAX Options` as required by the HTTP API you're trying to integrate with and specify the functions you want to be called when the `HTTPRequest` is successful or results in an error. See the JQuery website for an explanation of the `jQuery.ajax` object and its properties.

The following implementation is inherited from the `HTTPAPIDataAction` class. You need to rewrite the inherited method to specify requirements. For example, forming the HTTP request, and the code that handles the HTTP response. This implementation is useful as it shows the parameters passed to the `getAJAXOptions()` function, the object that it's expected to return, and gives a clear example of how to structure the code inside the method.

```
1 /**
2  * This method returns an object containing the AJAX settings used when the
data action is invoked.
3  * Subclasses may wish to override this method to provide their own
behavior.
4  * @param {module:obitech-reportservices/
dataactionmanager.DataActionContext} oDataActionContext The context metadata
describing where the data action was invoked from.
5  * @returns {?object} A JQuery AJAX settings object (see http://
api.jquery.com/jQuery.ajax/ for details) - returns null if there is a
problem.
6  */
7 dataaction.HTTPAPIDataAction.prototype.getAJAXOptions = function
(oDataActionContext)
8 {
9    jsx.assertInstanceOfModule(oDataActionContext, "oDataActionContext",
"obitech-reportservices/dataactionmanager", "DataActionContext");
10
11   var oAJAXOptions = null;
12   var oKOViewModel = this.getKOViewModel();
13   var sURL = oKOViewModel.sURL();
14   if (sURL)
15   {
16      // Parse the URL
17      var sResultURL = this._parseURL(sURL, oDataActionContext);
18      if (sResultURL)
19      {
20         // Parse the POST parameters (if required)
21         var eHTTPMethod = oKOViewModel.eHTTPMethod()[0];
22         var sData = null;
23         if (eHTTPMethod ===
dataaction.HTTPDataActionKOModel.HTTPMethod.POST)
```

```
24         {
25            var sPOSTParams = oKOViewModel.sPOSTParams();
26            sData =
sPOSTParams.replace(dataaction.AbstractHTTPDataAction.RegularExpressions.LINE_
END, "&");
27            sData = this._parseURL(sData, oDataActionContext, false);
28         }
29         oAJAXOptions = {
30            type: eHTTPMethod,
31            url: sResultURL,
32            async: true,
33            cache: false,
34            success: function (/*oData, sTextStatus, oJQXHR*/)
35            {
36
oDataActionContext.getReport().displaySuccessMessage(messages.HTTP_API_DATA_AC
TION_INVOCATION_SUCCESSFUL.format(oKOViewModel.sName()));
37            },
38            error: function (oJQXHR/*, sTextStatus, sError*/)
39            {
40
oDataActionContext.getReport().displayErrorMessage(messages.HTTP_API_DATA_ACTI
ON_INVOCATION_FAILED.format(oKOViewModel.sName(), oJQXHR.statusText,
oJQXHR.status));
41            }
42         };
43         if (sData)
44         {
45            oAJAXOptions.data = sData;
46         }
47      }
48   }
49   return oAJAXOptions;
50 };
```

Test, Package, and Install Your Data Action

You use Oracle Analytics Desktop to test your data action from its source location before you install it.

1. If Oracle Analytics Desktop is currently running, close it.
2. If you're working behind a proxy, set the proxy settings in `%PLUGIN_DEV_DIR%\gradle.properties`. For information about accessing the web through HTTP proxy, see Gradle User Manual.
3. Run Oracle Analytics Desktop in SDK mode by using the command prompt you started in Choose Which Data Action Inherited Methods to Override and enter the following commands:

   ```
   cd %PLUGIN_DEV_DIR%
   ```

   ```
   .\gradlew run
   ```

   Oracle Analytics Desktop starts in SDK mode. Your data action extension appears in the Console | Extensions page.

Create a workbook and test your data action. If you find any issues, you can debug your code using your browser's built-in developer tools.

4. If you created an HTTP API data action:
   a. Go to the Console and display the Safe Domains page.
   b. Add each domain that you want to access.

      For example, if you need access to the apilayer.com APIs, add `apilayer.net` to the list of safe domains.
   c. Click the **Connect** column checkbox for the selected domain.
   d. Reload the Safe Domains page in your browser for the changes to take effect.
5. If you want to prepare your data action extension to distribute to other people or to install in Oracle Analytics:
   - Package all of the files into a single ZIP file containing the `%PLUGIN_DEV_DIR%\src\customdataaction` folder and its contents.
   - Name the zip using the same ID you gave to your data action extension when you created it.
6. Install your data action extension. See Manage Oracle Analytics Extensions.

Use an Upgrade Handler for Knockout Model Changes

For some Knockout Model changes you need to upgrade your data action extension using an upgrade handler.

When you're making improvements to your data action extension without making changes to the Knockout Model you normally edit your JavaScript or CSS files, create a new ZIP file, and replace the existing data action extension with the new ZIP file. However, if you've made changes to your data action's Knockout Model then you might need to change the data action `VERSION` property and provide an upgrade handler.

Decide whether you need to use an upgrade handler:

Upgrade Handler Required

- If you rename a property in your Knockout Model.
- If you combine multiple properties into a single property in your Knockout Model.
- If you split a single property into multiple properties in your Knockout Model.
- If you add a new property to the Knockout Model and the correct default value for it depends on other values in the Knockout Model.

Upgrade Handler Not Required

- If you add a new property to the Knockout Model and can provide a default value that's correct for all existing usages of your data action.
- If you remove a property from the Knockout Model because it's no longer used by your data action code.

Upgrade Data Action Extensions

Upgrade your data action extensions to improve the data action code or upgrade the metadata to enable existing data actions to work with new data action code.

Use an upgrade handler to upgrade a data action extension.

1. Increase the version number of your data action.

   For example, if your data action is called company.MyDataAction, then search `mydataaction.js` for the `mydataaction.MyDataAction.VERSION` property. If it's currently set to `1.0.0` then change it to `1.0.1`.

2. Add a `static upgrade(oOldDataActionJS)` method to your data action's class.

   If the `VERSION` property differs from the `sVersion` value stored in the data action metadata then the Data Action Manager calls the `static upgrade()` method on your data action's class.

3. Implement your `upgrade()` method by calling the `upgrade()` method on the superclass and capture its response.

4. Continue to implement your `upgrade()` method by making further edits to the partially upgraded data action JSON returned by the superclass, until the object matches the correct set of properties required by your latest Knockout Model.

5. To finish call `var oUpgradedDataAction = dataaction.AbstractDataAction.createFromJS(fDataActionClass, sFullyQualifiedDataActionClassName, oUpgradedDataActionJS).`

   This command constructs a new instance of your data action from the upgraded data action JSON and returns `oUpgradedDataAction.getSettings()`.

Data Action Extension File Reference

Each data action extension requires a plugin.xml file and each plugin.xml file can contain any number of data actions.

Topics:

Data Action plugin.xml File Example

The plugin.xml file has three main sections, `tns:obiplugin`, `tns:resources`, and `tns:extension`.

Example plugin.xml

This example shows a typical plugin.xml file for one data action.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <tns:obiplugin xmlns:tns="http://plugin.frameworks.tech.bi.oracle"
```

```
3                id="obitech-currencyconversion"
4                name="Oracle BI Currency Conversion"
5                version="0.1.0.@qualifier@"
6                optimizable="true"
7                optimized="false">
8
9
10   <tns:resources>
11      <tns:resource id="currencyconversion" path="scripts/
currencyconversion.js" type="script" optimizedGroup="base"/>
12      <tns:resource-folder id="nls" path="resources/nls" optimizable="true">
13         <tns:extensions>
14            <tns:extension name="js" resource-type="script"/>
15         </tns:extensions>
16      </tns:resource-folder>
17   </tns:resources>
18
19
20   <tns:extensions>
21      <tns:extension id="oracle.bi.tech.currencyconversiondataaction" point-
id="oracle.bi.tech.plugin.dataaction" version="1.0.0">
22         <tns:configuration>
23         {
24            "host": { "module": "obitech-currencyconversion/
currencyconversion" },
25            "resourceBundle": "obitech-currencyconversion/nls/messages",
26            "properties":
27            {
28               "className": "obitech-currencyconversion/
currencyconversion.CurrencyConversionDataAction",
29               "displayName": { "key" : "CURRENCY_CONVERSION", "default" :
"Currency Conversion" },
30               "order": 100
31            }
32         }
33         </tns:configuration>
34      </tns:extension>
35   </tns:extensions>
36
37 </tns:obiplugin>
```

Data Action plugin.xml File Properties Section - tns:obiplugin

The `tns:obiplugin` section defines properties common to all types of extensions.

Extension Properties

The `tns:obiplugin` section defines properties common to all types of extensions.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <tns:obiplugin xmlns:tns="http://plugin.frameworks.tech.bi.oracle"
3                id="obitech-currencyconversion"
4                name="Oracle BI Currency Conversion"
5                version="0.1.0.@qualifier@"
```

```
6                optimizable="true"
7                optimized="false">
```

- **Line 1**: The XML declaration.
- **Line 2**: The opening tag for the extension's root XMLElement and the declaration for the `tns` namespace that's used throughout plugin.xml files.
- **Line 3**: The extension's unique ID.
- **Line 4**: The extension's default display name (used when a localized version isn't available).
- **Line 5**: The extension's version number.
- **Line 6**: A boolean indicating whether or not the JS/CSS can be optimized (compressed).
- **Line 7**: A boolean indicating whether or not the JS/CSS has been optimized (compressed).

Data Action plugin.xml File Resources Section - tns:resources

The `tns:resources` section registers all of the files that contribute to your extension.

Resources

```
1 <tns:resources>
2    <tns:resource id="currencyconversion" path="scripts/
currencyconversion.js" type="script" optimizedGroup="base"/>
3    <tns:resource-folder id="nls" path="resources/nls" optimizable="true">
4       <tns:extensions>
5          <tns:extension name="js" resource-type="script"/>
6       </tns:extensions>
7    </tns:resource-folder>
8 </tns:resources>
```

You need to register each JavaScript, CSS, Image, and Translation Resource File here. The section is contained within the `<tns:resources>` element and contains any number of the following elements:

- `<tns:resource>`
  These elements are used to register a single file (for example, a JavaScript or CSS file).
- `<tns:resource-folder>`
  These elements are used to register all the files under a specified folder at the same time. For example, an image folder or the folder containing the resource files for Native Language Support.

More information on how to register each type of file is provided in the following sections.

JavaScript Files

Each JavaScript file in your extension must be registered with a line similar to the one shown below.

```
<tns:resource id="currencyconversion" path="scripts/currencyconversion.js"
type="script" optimizedGroup="base"/>
```

Where:

- **id** is the ID given to the file.
  Set the ID to match the JavaScript filename without the .js extension.
- **path** is the relative path to the JavaScript file from the plugin.xml file. JavaScript files should be stored under your extension's `scripts` directory.
  Use all lowercase for your JavaScript files with no special characters (for example, underscore, hyphen).
- **type** is the type of file being registered. It must be set to `script` for JavaScript files.
- **optimizedGroup** groups multiple JavaScript files into a single compressed file. Third-party extensions must leave this set to `base`.

CSS Files

Each CSS file in your extension must be registered with a line similar to the one shown below.

```
<tns:resource id="currencyconversionstyles" path="resources/
currencyconversion.css" type="css"/>
```

Where:

- **id** is the ID given to the file.
  Set the ID to match the CSS filename without the .css extension.
- **path** is the relative path to the CSS file from the plugin.xml file. CSS files should be stored under your extension's `resources` directory.
  Use all lowercase for your CSS files with no special characters (for example, underscore, hyphen).
- **type** is the type of file being registered. It should always be set to `css` for CSS files.

Image Folders

If your extension has images that you need to refer to from within your JavaScript code, then put them in a `resources/images` directory within your extension's directory structure and add a `<tns:resource-folder>` element to your `plugin.xml` as follows:

```
<tns:resource-folder id="images" path="resources/images" optimizable="false"/>
```

If your images are only referenced by your CSS files, then you don't need to add this `<tns:resource-folder>` element to your `plugin.xml` file. In this case, you must still add them to the `resources/images` directory so that you can then refer to them using a relative path from your CSS file.

Native Language Support Resource Folders

Oracle Analytics implements Native Language Support. This requires developers to externalize the strings they display in their user interface into separate JSON resource files. You can then provide different localized versions of those files in a prescribed directory structure and Oracle Analytics automatically uses the correct file for the user's chosen language. You can provide as many translated versions of the resource files as needed. A Native Language Support resource folder points Oracle Analytics to the root of the prescribed Native Language Support directory structure used by your extension. All extensions that use Native Language Support

resource files must have a `<tns:resource-folder>` entry that looks exactly like the example below.

```
1 <tns:resource-folder id="nls" path="resources/nls" optimizable="true">
2    <tns:extensions>
3       <tns:extension name="js" resource-type="script"/>
4    </tns:extensions>
5 </tns:resource-folder>
```

See Generated Folders and Files for details about the contents of the files and the prescribed directory structure that you should follow.

Data Action plugin.xml File Extensions Section - tns:extension

For each data action you want your extension to provide, you must register a data action extension using a `<tns:extension>` element similar to this:

```
<tns:extension id="oracle.bi.tech.currencyconversiondataaction" point-
id="oracle.bi.tech.plugin.dataaction" version="1.0.0">
   <tns:configuration>
   {
      "host": { "module": "obitech-currencyconversion/currencyconversion" },
      "resourceBundle": "obitech-currencyconversion/nls/messages",
      "properties":
      {
         "className": "obitech-currencyconversion/
currencyconversion.CurrencyConversionDataAction",
         "displayName": { "key" : "CURRENCY_CONVERSION", "default" :
"Currency Conversion" },
         "order": 100
      }
   }
   </tns:configuration>
</tns:extension>
```

Where:

- **id** is the unique ID you give to your data action.
- **point-id** is the type of extension you want to register. For data action extensions, this must be set to `oracle.bi.tech.plugin.dataaction`.
- **version** is the extension API version that your extension definition uses (leave this set to **1.0.0**).

The `<tns:configuration>` element contains a JSON string that defines:

- **host.module** - This is the fully qualified name of the module containing your data action. This fully qualified module name is formulated as `%PluginID%/%ModuleName%`, where:
  - `%PluginID%` must be replaced with the extension ID you specified in the id attribute of the `<tns:obiplugin>` element.
  - `%ModuleName%` must be replaced with the resource ID you specified in the id attribute of the `<tns:resource>` element for the JavaScript file containing your data action.

- **resourceBundle** - This is the Native Language Support path to the resource file that contains this data action's localized resources. If your resource files are named messages.js and stored correctly in the prescribed `nls` directory structure, then set this property to `%PluginID%/nls/messages` (where `%PluginID%` must be replaced with the extension ID you specified in the id attribute of the `<tns:obiplugin>` element at the top of the `plugin.xml` file).
- **properties.className** - This is the fully qualified class name given to the data action you're registering. This fully qualified class name is formulated as `%PluginID%/%ModuleName%.%ClassName%`, where:
  - `%PluginID%` must be replaced with the extension ID you specified in the id attribute of the `<tns:obiplugin>` element.
  - `%ModuleName%` must be replaced with the resource ID you specified in the id attribute of the `<tns:resource>` element for the JavaScript file containing your data action.
  - `%ClassName%` must be replaced with the name you gave to the data action class in your JavaScript file.
- **properties.displayName** - This property contains an object and two further properties:
  - **key** is the Native Language Support message key that can be used to lookup the data action's localized display name from within the specified `resourceBundle`.
  - **default** is the default display name to use if for some reason the localized version of the display name can't be found.
- **properties.order** - This property enables you to provide a hint that's used to determine the position that this data action should appear when shown in a list of data actions. Data actions with lower numbers in their order property appear before data actions with higher numbers. When there's a tie, the data actions are displayed in the order they're loaded by the system.

3

Create Oracle Analytics Visualization and Workbook Extensions

This chapter describes how to set up your development environment to create and test custom visualization and workbook extensions.

Topics:

About the Oracle Analytics Extension Development Environment

After you set up your extension development environment, you use the scripts and SDK provided with Oracle Analytics Desktop to create, develop, and test custom visualization and workbook extensions.

Topics:

Workflow to Set Up the Oracle Analytics Extension Development Environment

Here are the tasks you need to complete to set up your extension development environment. You can begin creating your extensions after you've successfully set up your environment.

Task	Description	More Information
Install Oracle Analytics Desktop	Provides the scripts you need to create your environment and create an extension skeleton. Oracle Analytics Desktop also functions as a local environment where you run and test your extensions.	Install Oracle Analytics Desktop on Mac Install Oracle Analytics Desktop on Windows
Install Java JDK	Provides the Java tools and libraries required to build your extensions.	Install Java JDK on Mac Install Java JDK on Windows

Task	Description	More Information
Set variables on your computer	Ensures that the Oracle Analytics Desktop scripts work properly. For Mac you configure bash profile, and for Windows you set user variables.	Update Bash Profile or ZSHRC File and Create the Development Directory on Mac Set User Variables and Create a Development Directory on Windows
Add a directory to contain your development environment	Provides a location where you create your development environment.	Update Bash Profile or ZSHRC File and Create the Development Directory on Mac Set User Variables and Create a Development Directory on Windows
Create your extension development environment	Provides the framework and resources you use to create and develop extensions.	Create the Extension Development Environment on Mac Create the Extension Development Environment on Windows

Oracle Analytics Extensions Development Scripts

Your installation of Oracle Analytics Desktop includes the scripts you use to create your development environment and create and work with extensions.

- **bicreateenv** - Run this script to create the environment where you develop your extensions.
- **bicreateplugin** - Run this script to create a skeleton extension. For information about the types of extensions that you can create with this script, see Types of Oracle Analytics Extensions.
- **bideleteplugin** - Run this script to delete an extension from your development environment.
- **bivalidate** - Run the `gradlew validate` command to call this script. The bivalidate script validates that the JSON configuration files are properly formatted and contain the appropriate extension configuration.

Types of Oracle Analytics Extensions

This topic lists the types of extensions you can create when you run the `bicreateplugin` script.

Visualization Extensions

Runing the `bicreateplugin` command creates a folder containing the files that you use to develop your visualization extension. The entry point of the visualizations is the `render()` method on the file <vizName>.js. The `render()` method is invoked during the creation of the visualization and during events like resize, data update, and so on.

You can create the following types of visualization extensions.

- **basic** - Creates a visualization that doesn't use any data from Oracle Analytics or any data model mapping. This is like the Image and Text visualization types delivered with Oracle Analytics.

For example, you can use this visualization type to show an image or some text that's coded into the extension or from a configuration. You can use this type of visualization to improve formatting.

- **dataviz** - Creates a visualization that renders data from data sources registered with Oracle Analytics into a chart or table or some other representation.
- **embeddableDataviz** - Creates a visualization that renders data from data sources registered with Oracle Analytics into the cells of a trellis visualization.

Workbook Extension

You can create a workbook extension.

- **workbook** - Creates the base structure that you use to develop a workbook-scoped extension. The extension's entry point is the `performMainAction` method. This code exists in the .js file created by the extension command, for example workbook.js.

Oracle Analytics Extension Development Resources

Oracle Analytics provides information to help you develop your extensions.

- **circlePack sample** - The circlePack sample is included in your development environment to help you learn how to develop a visualization extension. You can deploy and use this sample immediately. You can also copy the sample and use it as a template for the visualization extensions that you want to create.

  The circlePack sample is located in your development environment, for example `<your_development_directory>\src\sampleviz\sample-circlepack`
- **Extensions library** - Extensions are available for you to download from the Oracle Analytics Extensions library.
- **JS API documentation** - The API documentation contains JavaScript reference information that you need to develop an extension.
- **Oracle Analytics product documentation** - These resources contain information about how to create workbooks and visualizations:

  Begin to Build a Workbook and Create Visualizations

  Get Started with Visualizations video

Oracle Analytics Extensions Limitations

The extensions that you create are custom code and may not work properly in all browsers or on all devices.

When creating an extension, you as the developer must test all of the browsers and devices that you want the extension to render on.

Also, in some cases extensions may not work in the Oracle Analytics mobile application due to application restrictions that don't apply to browsers.

Set Up the Oracle Analytics Extension Development Environment on Mac

This topic describes the tasks you need to perform to set up and use your Oracle Analytics extension development environment.

Topics:

Install Oracle Analytics Desktop on Mac

Oracle Analytics Desktop provides the scripts needed to create your development environment and extension skeletons, and a local test environment.

Install or upgrade to the latest version of Oracle Analytics Desktop.

See Workflow to Set Up the Oracle Analytics Extension Development Environment .

1. Go to Oracle Analytics Desktop Installation Download, click **Download** and log into your Oracle Cloud account.
2. In the Oracle Software Delivery Cloud page, click **Platforms** and select **Apple Mac OS X**.
3. Review and accept the license agreement. Click the Oracle Analytics Desktop ZIP file to download it.
4. Go to the download location on your computer, click the ZIP file, and click Oracle_Analytics_Desktop_<version>_Mac.pkg and perform the installation.
5. Navigate to the Applications folder and confirm the installation created these applications:
   - Oracle Data Visualization for Desktop
   - Oracle Data Visualization for Desktop Configure Python

Install Java JDK on Mac

Use a Java JDK version that's compatible with your macOS and processor. All examples in this chapter were developed with Java JDK 8u201.

See Workflow to Set Up the Oracle Analytics Extension Development Environment .

1. Open Terminal and enter this command to check if you have Java JDK installed.

   ```
   java -version
   ```

2. If one or more Java JDK is installed, confirm that one is compatible with your macOS and processor.
3. If you need to install Java JDK, go to Java SE 8 Archive Downloads.

4. In the table, click the macOS tab. Locate and download the install file compatible with your macOS and processor.
5. Locate and run the downloaded installation file.
6. After the installation completes, in Terminal enter this command to check that the Java JDK version you picked installed successfully:

   ```
   java -version
   ```

Update Bash Profile or ZSHRC File and Create the Development Directory on Mac

Modify your bash profile or ZSHRC file to include the variables required by the Oracle Analytics Desktop scripts. Then create the development directory to contain your development environment.

See Workflow to Set Up the Oracle Analytics Extension Development Environment .

1. To modify your Bash Profile, go to the home directory and check if bash_profile is visible. If not, press **Command + Shift + .** to make bash_profile visible.

   To modify your ZSHRC file, open Terminal and run this command: .

   ```
   open ~/.zshrc
   ```

2. Add these lines to bash_profile or ZSHRC.

   In `PLUGIN_DEV_DIR` specify the location of the development directory, for example `/Users/<username>/Documents/dv-custom-plugins`.

   ```
   export DVDESKTOP_SDK_HOME=/Applications/dvdesktop.app/Contents/Resources/app.nw

   export PLUGIN_DEV_DIR=/Users/<username>/Documents/dv-custom-plugins

   export PATH=${DVDESKTOP_SDK_HOME}/tools/bin:$PATH

   export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk-1.8.jdk/Contents/Home/
   ```

3. For bash_profile, open Terminal and run this command to apply the changes:

   ```
   source ~/.bash_profile
   ```

   For the ZSHRC file, open Terminal and run this command to apply the changes:

   ```
   source ~/.zshrc
   ```

4. To create the extension development directory, open Terminal and run this command:

   ```
   mkdir $PLUGIN_DEV_DIR
   ```

Create the Extension Development Environment on Mac

After you configure bash profile, you run the `bicreateenv` script to create the development environment that contains the resources you need to create extensions.

For information about the options available for running this script, see the script's command-line help, for example:

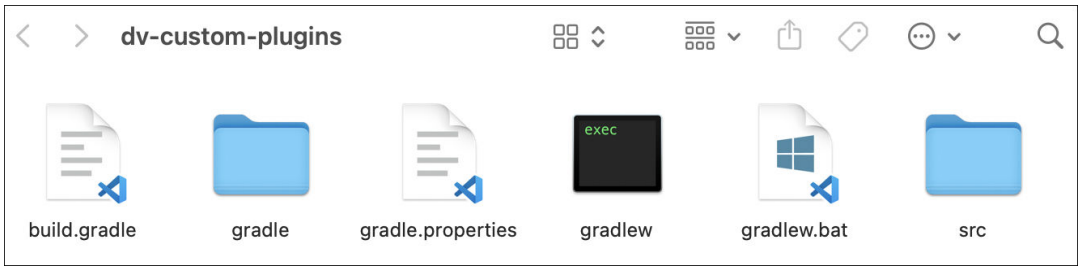
```
cd $PLUGIN_DEV_DIR
bicreateenv -help
```

See Workflow to Set Up the Oracle Analytics Extension Development Environment .

1. In Finder, navigate to the extension directory and run the `bicreateenv` script to create the environment:

   ```
   cd $PLUGIN_DEV_DIR
   bicreateenv
   ```

2. Navigate to the directory that you created and confirm that its contents look like this:



3. Open `build.gradle` and search for `-pluginDevDir`. If the `-pluginDevDir` argument contains capital letters, change them to lowercase letters. The modified argument should look like this:

   ```
   args '-sdk', "-plugindevdir=${projectDir}${File.separator}src"
   ```

4. Optional: If you're working behind a web proxy, open `gradle.properties` and add system properties that point to your proxy.

   Use the following example to set your system properties:

   ```
   systemProp.https.proxyHost=www-proxy.somecompany.com
   systemProp.https.proxyPort=80
   systemProp.https.nonProxyHosts=*.somecompany.com|*.companyaltname.com
   ```

Create a Skeleton Extension on Mac

Use the `bicreateplugin` script to create an Oracle Analytics extension skeleton.

For information about the extensions you can create when you run the `bicreateplugin` script, see Types of Oracle Analytics Extensions.

Running the script creates a folder in your PLUGIN_DEV_DIRECTORY environment. This folder contains the files that you use to develop the extension. The <extension_name>.js render method is the entry point where you can start writing code.

The `bicreateplugin` script uses the following syntax:

```
bicreateplugin viz -subType <subtypename> -id <com.company.yourVizName>
```

Where:

`subType` is the type of visualization extension you want to create. Valid values are `basic`, `dataviz`, and `embeddableDataviz`. Don't include `subType` when you create a workbook extension.

`id` is the name of the extension. The name you specify must be in this format: `<com.company.yourVizName>`.

1. In Terminal, navigate to your extension development directory, run the `bicreateplugin` script.

   This example shows how to create a dataviz skeleton extension:

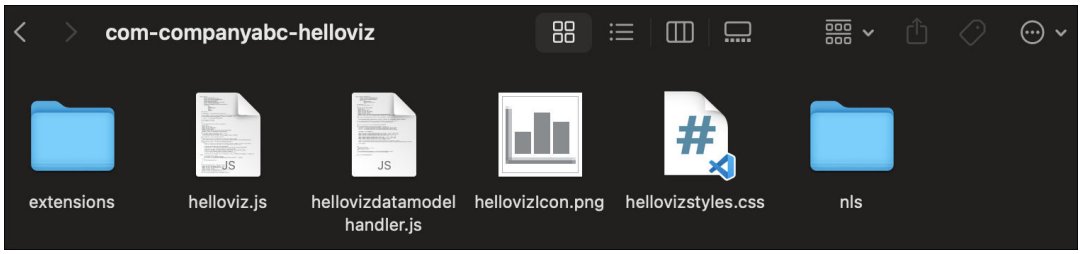
   ```
   bicreateplugin viz -subType dataviz -id com.companyabc.helloviz
   ```

   This example shows how to create a workbook skeleton extension:

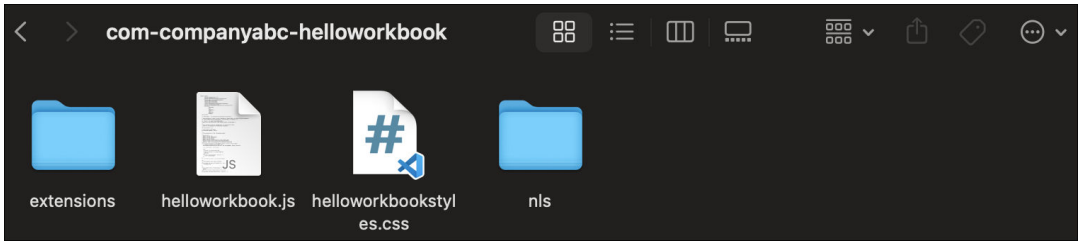
   ```
   bicreateplugin workbook -id com.companyabc.helloworkbook
   ```

2. In Finder, navigate to the `src/customviz` folder and confirm that a new folder was created and that its name matches the extension name you specified when you ran the script.

   This example shows a dataviz extension's directory:



   This example shows a workbook extension's directory:



Test Your Visualization and Workbook Extensions on Mac

Use Terminal to run Oracle Analytics Desktop in SDK mode to test your Oracle Analytics visualization and workbook extensions. Running Oracle Analytics Desktop in SDK mode opens Oracle Analytics Desktop in the browser.

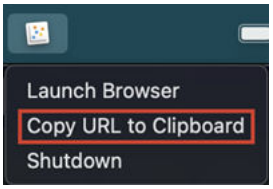
For information about creating workbooks and adding visualizations to workbooks, see the *Oracle Analytics product documentation* section in Oracle Analytics Extension Development Resources.

You must build and package a workbook extension before you can upload it to Oracle Analytics Desktop to test it. See Build and Package an Extension.

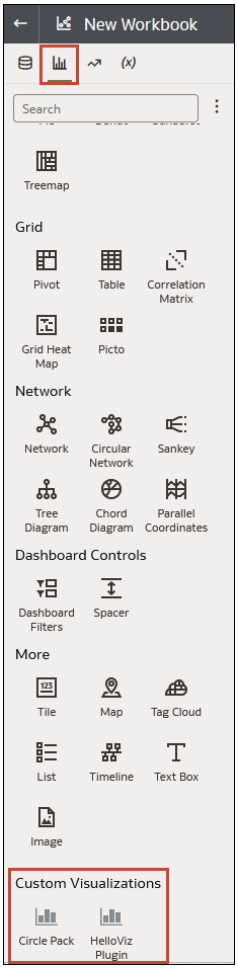
1. In Terminal run this command to invoke Oracle Analytics Desktop in the browser:

   ```
   ./gradlew run
   ```

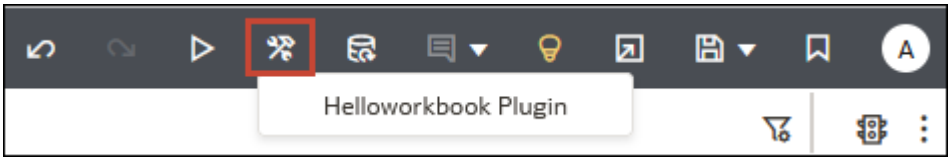
2. If after you run the command Oracle Analytics Desktop opens and then closes, you can use the Mac menu bar to manually open Oracle Analytics Desktop in a browser.
   a. Go to the Mac menu bar and locate and click the Oracle Analytics Desktop icon.
   b. Select **Copy URL to Clipboard**.

   c. In a browser, paste the copied URL and press Enter.
3. To test a visualization extension:
   a. In Oracle Analytics Desktop, open or create a workbook.
   b. In the workbook's Data Panel, click **Visualizations** and scroll to the bottom of the Visualizations list to locate the Custom Visualizations section containing the custom visualizations you created.



4. To test a workbook extension:
   a. In Oracle Analytics Desktop, click **Navigator** and then click **Console**. Go to the **Extensions and Enrichments** section and click **Extensions**.
   b. Click **Upload Extensions** and browse for and select the workbook extension ZIP file. Click **Open**.
   c. In Oracle Analytics Desktop, open or create a workbook.
   d. In the Toolbar click **Custom Workbook Extension** to view a list of the workbook extensions that you uploaded to your instance.



Set Up the Oracle Analytics Extension Development Environment on Windows

This topic describes the tasks you need to perform to set up and use your Oracle Analytics extension development environment.

Topics:

Install Oracle Analytics Desktop on Windows

Oracle Analytics Desktop provides the scripts needed to create your development environment and extension skeletons, and a local test environment.

Install or upgrade to the latest version of Oracle Analytics Desktop.

See Workflow to Set Up the Oracle Analytics Extension Development Environment .

1. Go to Oracle Analytics Desktop Installation Download, click **Download** and log into your Oracle Cloud account.
2. In the Oracle Software Delivery Cloud page, click **Platforms** and select **Microsoft Windows x64**.
3. Review and accept the license agreement. Click the Oracle Analytics Desktop ZIP file to download it.
4. Go to the download location on your computer, double-click the ZIP file, and double-click Oracle_Analytics_Desktop_<version>_Win.exe and perform the installation.
5. Navigate to `C:\Program Files\Oracle Analytics Desktop` to confirm the installation.

Install Java JDK on Windows

Use a Java JDK version that is compatible with your Windows and processor. All examples in this chapter were developed with Java JDK 8u201.

See Workflow to Set Up the Oracle Analytics Extension Development Environment .

1. Open Command Prompt and enter this command to check if you have Java JDK installed:

   ```
   java -version
   ```

2. If one or more Java JDK is installed, confirm one is compatible with your macOS and processor.
3. If you need to install Java JDK, go to Java SE 8 Archive Downloads.

4. Locate and download the JDK install file compatible with your Windows and processor.
5. After the installation completes, open Command Prompt and enter this command to check that the Java JDK version you picked installed successfully:

```
java -version
```

Set User Variables and Create a Development Directory on Windows

Create or modify the user variables required by the Oracle Analytics Desktop scripts. Then create the development directory to contain your development environment,

In this procedure, you create or update these required user variables:

- **PLUGIN_DEV_DIR** - The location of your development directory, for example `C:\PLUGIN_DEV_DIR`.
- **DVDESKTOP_SDK_HOME** - The location of your Oracle Analytics Desktop installation, for example `C:\Program Files\Oracle Analytics Desktop\dvdesktop`.
- **JAVA_HOME** - The location of your JDK 1.8 installation, for example `C:\Program Files\Java\jdk-1.8`.
- **Path** - The location of your Oracle Analytics Desktop bin directory, for example `C:\Program Files\Oracle Analytics Desktop\tools\bin`. This variable already exists in Windows. When you update it, make sure that you don't delete or modify any of the variable's existing paths.

See Workflow to Set Up the Oracle Analytics Extension Development Environment .

1. Open File Explorer, right-click **This PC**, and then click **Properties**. Click **Advanced System Settings**, and in the Advanced tab click **Environment Variables**.
2. In Environment Variables, click **New** under **User variables for <computer name>**. In the New User Variable dialog, go to **Variable name** and enter the name of the variable, and then browse for or enter the directory location. See the list at the top of this task for variable name and value requirements. Click **OK**.
3. In Environment Variables, under **User variables for <computer name>** click the Path variable, and then click **Edit**. Browse for or enter the location of your Oracle Analytics Desktop bin directory. Click **OK**.
4. In the Environment Variables dialog, click **OK**.
5. To create the development directory, open the Command Prompt and run this command:

```
cd C:\
mkdir $PLUGIN_DEV_DIR
```

Create the Extension Development Environment on Windows

After you configure user variables, you run the `bicreateenv` script to create the development environment that contains the resources you need to create extensions.

For information about the options available for running the script, see the script's command-line help, for example:

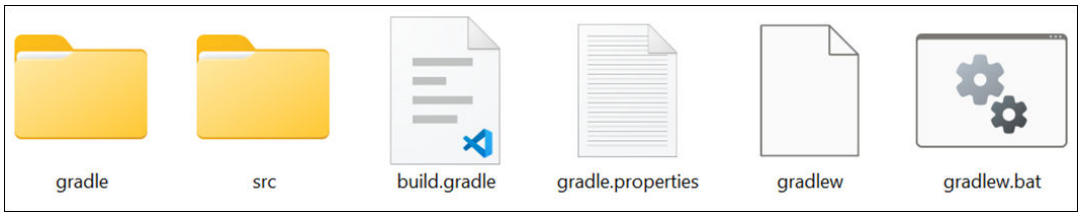
```
cd $PLUGIN_DEV_DIR
bicreateenv -help
```

See Workflow to Set Up the Oracle Analytics Extension Development Environment .

1. Open Command Prompt, and run the `bicreateenv` script to create the environment, for example:

```
cd $PLUGIN_DEV_DIR
bicreateenv
```

2. In File Explorer, navigate to the directory that you created and confirm that its contents look like this:



3. Optional: If you're working behind a web proxy, open `gradle.properties` and add system properties that point to your proxy.

   Use the following example to set your gradle.properties:

```
systemProp.https.proxyHost=www-proxy.somecompany.com
systemProp.https.proxyPort=80
systemProp.https.nonProxyHosts=*.somecompany.com|*.companyaltname.com
```

Create a Skeleton Extension on Windows

Use the `bicreateplugin` script to create an Oracle Analytics extension skeleton.

The `bicreateplugin` script uses the following syntax:

```
bicreateplugin viz -subType <subtypename> -id <com.company.yourVizName>
```

Where:

`subType` is the type of visualization extension you want to create. Valid values are `basic`, `dataviz`, and `embeddableDataviz`. Don't include `subType` when you create a workbook extension.

`id` is the name of the extension. The name you specify must be in this format: `<com.company.yourVizName>`.

For information about the extensions you can create when you run the `bicreateplugin` script, see Types of Oracle Analytics Extensions. The examples used in this topic show you how to create the dataviz and workbook skeleton extensions.

Running the script creates a folder in your PLUGIN_DEV_DIRECTORY environment. This folder contains the files that you use to develop the extension. The <extension_name>.js render method is the entry point where you can start writing code.

1. In Command Prompt, run the `bicreateplugin` script in your development directory.

   ```
   cd $PLUGIN_DEV_DIR
   bicreateplugin viz -subType <subtypename> -id <com.company.yourVizName>
   ```

   This example shows how to create a dataviz skeleton extension:

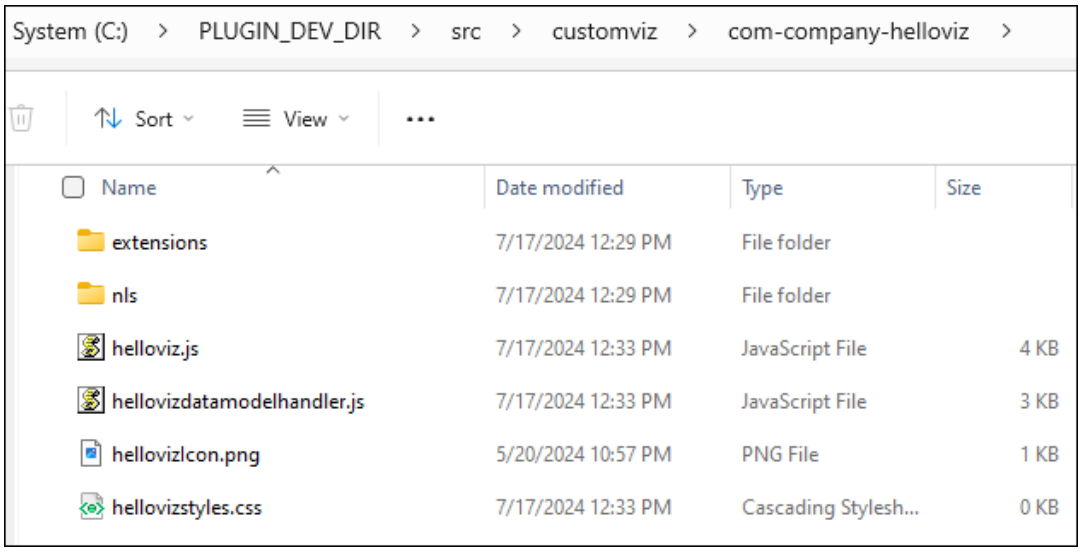
   ```
   bicreateplugin viz -subType dataviz -id com.companyabc.helloviz
   ```

   This example shows how to create a workbook skeleton extension:

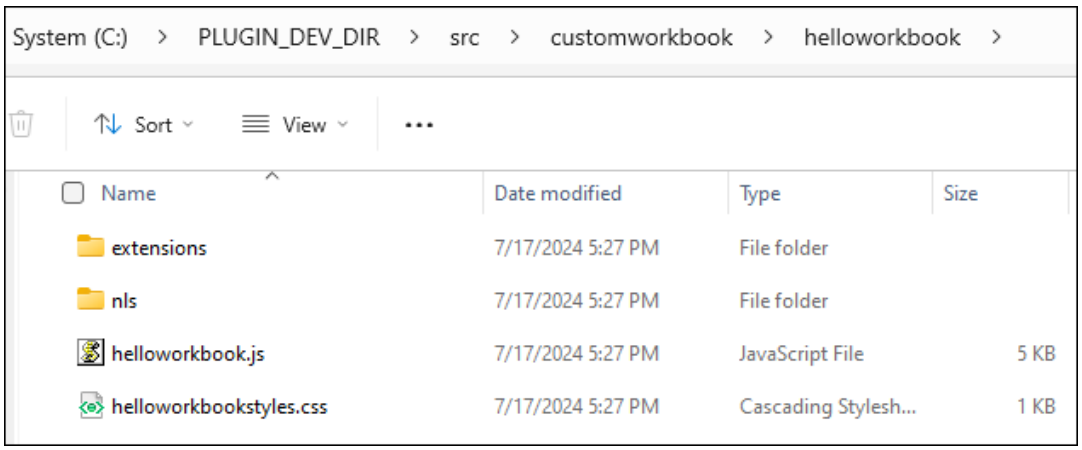
   ```
   bicreateplugin workbook -id com.companyabc.helloworkbook
   ```

2. In File Explorer, navigate to your development environment and extension directory, for example `C:\PLUGIN_DEV_DIR\src\customviz`, and confirm that a new folder was created and that its name matches the extension name you specified.

   This example shows a dataviz extension's directory:



   This example shows a workbook extension's directory:



Test Your Visualization and Workbook Extensions on Windows

Use Command Prompt to run Oracle Analytics Desktop in SDK mode to test your Oracle Analytics visualization and workbook extensions. Running Oracle Analytics Desktop in SDK mode opens Oracle Analytics Desktop in a browser.

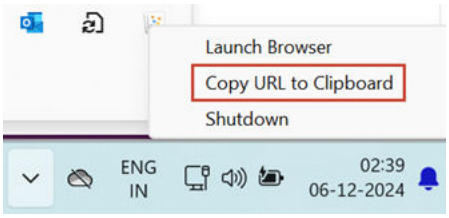
For information about creating workbooks and adding visualizations to workbooks, see the *Oracle Analytics product documentation* section in Oracle Analytics Extension Development Resources.

You must build and package a workbook extension before you can upload it to Oracle Analytics Desktop to test it. See Build and Package an Extension.

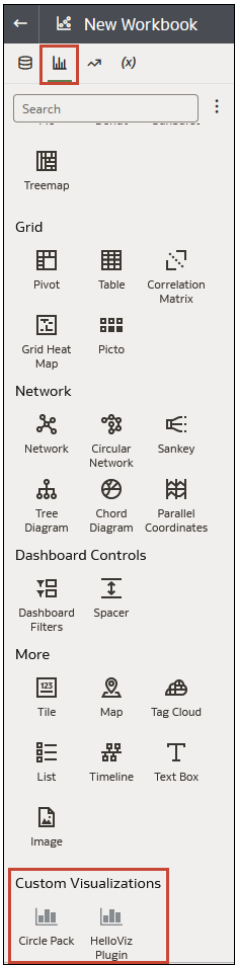
1. In Command Prompt, change to PLUG_IN_DEV_DIR and run this command to invoke Oracle Analytics Desktop in a browser:

   ```
   cd $PLUGIN_DEV_DIR
   ./gradlew run
   ```

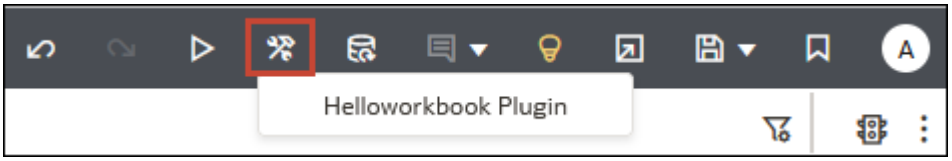
2. If after you run the command Oracle Analytics Desktop opens and then closes, you can use the Windows task bar to manually open Oracle Analytics Desktop in a browser.
   a. Go to the Windows task bar and click **Show hidden icons**. Locate and right-click the Oracle Analytics Desktop icon.
   b. Select **Copy URL to Clipboard**.

   c. In a browser, paste the copied URL and press Enter.
3. To test a visualization extension:
   a. In Oracle Analytics Desktop, open or create a workbook.
   b. In the workbook's Data Panel, click **Visualizations** and scroll to the bottom of the Visualizations list to locate the Custom Visualizations section containing the custom visualizations you created.



4. To test a workbook extension:
   a. In Oracle Analytics Desktop, click **Navigator**, and then click **Console**. Go to the **Extensions and Enrichments** section and click **Extensions**.
   b. Click **Upload Extensions** and browse for and select the workbook extension ZIP file. Click **Open**.
   c. In Oracle Analytics Desktop, open or create a workbook.
   d. In the Toolbar click **Custom Workbook Extension** to view a list of the workbook extensions and you uploaded to your instance.



Work with Extensions

This topic describes some of the tasks you perform when you develop your Oracle Analytics extensions.

Topics:

Build and Package an Extension

Run the `gradlew clean build` command to build and package an extension into a ZIP file. You upload this file to your Oracle Analytics production environment, or to your Oracle Analytics Desktop test environment.

After you run the command, Oracle Analytics Desktop adds a build directory to your development environment, for example `C:/PLUGIN_DEV_DIR/build/distributions`. This directory contains a ZIP file for each extension in your development directory.

- Navigate to your development directory and run the `gradlew clean build` command. For example,

```
cd $PLUGIN_DEV_DIR
./gradlew clean build
```

Upload an Extension to Oracle Analytics

After you build and package your extension to a ZIP file in your Oracle Analytics Desktop development environment, you upload the ZIP file to your Oracle Analytics production environment.

See Build and Package an Extension.

After you upload a visualization extension to Oracle Analytics and create or open a workbook, the Visualization tab displays the visualization extension in the Custom Visualizations section. From there you can drag and drop the custom visualization to your workbook. The extension displays as an option for every workbook on the instance where you uploaded the extension. All visualization extension types are available for you to add to workbooks. See Types of Oracle Analytics Extensions.

After you upload a workbook extension to Oracle Analytics, and create or open a workbook, the **Custom Workbook Extension** icon is displayed in the workbook's toolbar. Click this icon to view a list of the uploaded workbook extensions. Click an extension from the list to invoke it. The workbook extensions display for every workbook on the instance where you uploaded the extension.

1. Open Oracle Analytics. On the Home page click **Navigator**. In Navigator click **Console**.
2. Under **Extensions and Enrichments**, click **Extensions**.
3. In Extensions, click **Upload Extensions**, and select the ZIP file containing the extension. Then click **Open**.

Delete Extensions from the Oracle Analytics Development Environment

You can use the `bideleteplugin` script to delete any extension from your Oracle Analytics development environment.

The build and package process includes all of the visualizations contained in your development directory. To exclude any unwanted visualizations from the build, you can delete them before you perform the build and package process.

Use the information in this table to help you delete an extension:

Action	Command
Delete an extension	`cd $PLUGIN_DEV_DIR` `bideleteplugin viz -id <id_of_extension>`
Delete an unclassified extension	`cd $PLUGIN_DEV_DIR` `bideleteplugin unclassified -id <id_of_extension>`
Delete a skin extension	`cd $PLUGIN_DEV_DIR` `bideleteplugin skin -id <id_of_extension>`

4

Manage Oracle Analytics Extensions

You can upload, download, search for, and delete extensions. Extensions are custom visualizations, workbooks, or data actions that you or a developer create externally and then import into Oracle Analytics.

 LiveLabs Sprint

For example, you can upload an extension that provides a custom visualization that you can add to workbooks.

1. On the Home page, click the **Navigator**, and then click **Console**.
2. Click **Extensions**.
3. To upload an extension, click **Upload Extension**, browse to the extension ZIP file, and click **Open** to upload the extension.
4. Perform any of the following tasks.
   - To search for an extension, enter your search criteria in the **Search** field and click **Return** to display search results.
   - To delete an extension, click **Options** on the extension and select **Delete**, and click **Yes** to delete the extension.
     If you delete a visualization type that's used in a workbook, then that workbook displays an error message in place of the visualization. Either click **Delete** to remove the visualization, or upload the same extension so that the visualization renders correctly.
   - To download an extension, click **Options** on the extension and select **Download**.

Part III
Embed Content

This part explains how to embed content into applications and web pages.

Topics:

5

Get Started Embedding Content into Applications and Web Pages

This chapter contains information that you need to know before you embed content into applications and web pages.

Topics:

About Embedding Oracle Analytics Content into Applications and Web Pages

You can embed Oracle Analytics content into an application, custom application, or portal web page.

When you embed analytics, you put information where users need it to make business decisions. Embedded analytics delivers fast time-to-insight and increases user productivity.

There are two analytics content embedding methods:

- Use the analytics content item's URL. Typically this method uses an iFrame. See Embed Oracle Analytics Content With iFrames.
- Use the JavaScript embedding framework when you need an integrated way to embed analytics content. This method provides greater flexibility than the iFrame embedding method. For example, use this method when you want to embed visualizations into a custom web application. See Typical Workflow to Use the JavaScript Embedding Framework with Oracle Analytics Content.

Register an Application as a Safe Domain

Before you can embed Oracle Analytics content into another application, your administrator must register the application's domain as safe.

For security reasons, you’re not allowed to add analytics content to an applications unless your administrator considers it safe to do so.

See Register Safe Domains.

Web browsers have become more restrictive about dealing with third party cookies. This restriction can impact embedding projects where the browser won't display your embedded analytics content.

To work around this issue make sure that the Oracle Analytics instance where you embed analytics content is on a subdomain of the host web page or web application.

Use this information if you're using JavaScript to embed analytics content:

- Due to CORS safeguarding, you can't open your HTML file containing embedded analytics content directly in a browser. To work around this issue you must register the web server (either localhost or another web server) as a safe domain.
- If you use a localhost web server for testing, then you may need to add references to http://localhost:<port> and http://127.0.0.1:<port>.

You must be an administrator to perform this task.

1. Go to Oracle Analytics, click **Navigator**, and click **Console**.
2. Click **Safe Domains**.
3. Click **Add Domain** and enter the domain.
4. Select **Embedding**.
5. If using compatibility mode with embedding, select **Allow Frames**.

6

Embed Oracle Analytics Content With iFrames

This chapter explains how to use iFrames to embed Oracle Analytics content into applications and web pages.

Topics:

Considerations for Embedding Oracle Analytics Content With iFrame

This topic describes issues that you might encounter when you use iFrame to embed Oracle Analytics content into applications and web pages.

Users can open embedded Oracle Analytics content from an application if single sign-on is set up, or if there's already an active session for the embedded Oracle Analytics in the same browser.

If you're using the Safari browser and the embedded analytics content doesn't display as expected, try disabling Safari's **Prevent cross-site tracking** preference.

Use iFrame to Embed Analytics Content into an Application or Web Page

You can embed your analytics content into an application or web page by adding the target analytics content's URL into an application or portal's iFrame. For example, you can use this method to embed analytics content into Microsoft Teams.

> **Note:**
>
> If you need an integrated way to embed analytics content, then use the JavaScript embedding framework. This method provides greater flexibility than the iFrame embedding method. See Typical Workflow to Use the JavaScript Embedding Framework with Oracle Analytics Content.

Before you perform this task, confirm that you've registered the domain that you want to embed your analytics content into as a safe domain. See Register an Application as a Safe Domain.

If you need to manually build the URL, for example to create a URL that includes parameters, be sure to properly escape any characters. All special characters in the URL need to be URL-encoded. For example, use `%2C` to encode commas and `%20` to encode spaces.

1. On the Home page, click **Navigator**, and then click **Catalog**.
2. Locate the item that you want to embed and click **Actions**. Click **Open**.

3. Go to the browser's address bar and copy the item's URL. These are examples of URLs:

   - **Report** - `http://example.com/analytics/saw.dll?PortalGo&path=%2Fshared%2FRevenuehttp://example.com/analytics/saw.dll?PortalGo&Action=prompt&path=%2Fshared%2FSaled%2FSales%20by%20Brand`
   - **Dashboard** - `http://example.com/analytics/saw.dll?Dashboard&PortalPath=%2Fshared%2FSales%2F_portal%2FQuickStart&page=Top%20Products`
   - **Workbook** - `http://example.com/ui/dv/home.jsp?pageid=visualAnalyzer&reportmode=full&reportpath=%2Fshared%2FMySalesWorkbook`
   - **Canvas** - `https://example.com:8080/ui/dv/?pageid=visualAnalyzer&reportmode=full&reportpath=%2F%40Catalog%2Fusers%2Fadmin%2FOAC%20Demo%20Samples%2FCost%20Management%20Analytics%20copy&canvasname=canvas!2`.

     See Share a Workbook URL with a Specific Canvas Selected.

4. Alternatively, manually build and then copy the URL to insert into an iFrame.

   This is an example of how to construct a URL containing parameters:

   ```
   https://example.com/ui/dv/ui/project.jsp?pageid=visualAnalyzer&reportmode=full&reportpath=%2F%40Catalog%2Fshared&p1n=pCustomerSegment&p1v=Corporate&p2n=pCity&p2v=Bristol%2CCardiff%2CAustin
   ```

5. Open the target application or portal, locate an iFrame, and paste the analytics content's URL into it.

7

Embed Oracle Analytics Content With the JavaScript Embedding Framework

This chapter explain how to use the JavaScript embedding framework to embed Oracle Analytics content into applications and web pages.

Topics:

Typical Workflow to Use the JavaScript Embedding Framework with Oracle Analytics Content

If you're using the JavaScript embedding framework to embed Oracle Analytics content into an application or web page, then follow these tasks as a guide.

> **Note:**
>
> You can also embed Oracle Analytics content using the analytic content item's URL. Typically this method uses an iFrame. See Embed Oracle Analytics Content With iFrames.

Task	Description	More Information
Add safe domains	Use the Console to register as safe the development, production, and test environments domains.	Register an Application as a Safe Domain
Enable Developer options	Use the Developer's page to find the <script> tag, HTML, and column's expression that you need to embed analytics content.	Enable Oracle Analytics Developer Options

Task	Description	More Information
Create the HTML page	Create the HTML page where you'll embed analytics content. Steps include: reference the embedding.js JavaScript source and the embedded workbook's URL, specify filters and parameters, and specify how to refresh data.	Prepare the HTML Page for Embedded Oracle Analytics Content Pass Filters to the HTML Page for Embedded Oracle Analytics Content Pass Parameters to the HTML Page for Embedded Oracle Analytics Content Refresh Data in the HTML Page for Embedded Oracle Analytics Content
Specify the embedding mode	Your application uses JET or another technology to embed analytics content.	Embed Oracle Analytics Content into a Custom Application that Uses Oracle JET Embed Oracle Analytics Content into a Custom Application That Doesn't Use Oracle JET
Understand how authentication works	Learn about login prompt authentication and how to customize the login message that users see.	Use Login Prompt Authentication With Embedded Oracle Analytics Content

Enable Oracle Analytics Developer Options

Enable the developer's options to access the Oracle Analytics Developer's page. Use the Developer's page to find the <script> tag, HTML, and column's expression that you need to embed Oracle Analytics content into an application or web page.

1. Go to the top toolbar and click your user name.
2. Click **Profile** and in the Profile window, click **Advanced**.
3. Click the **Enable Developer Options** icon and click **Save**.

Find the Javascript and HTML for Embedding Oracle Analytics Content

Oracle Analytics generates the analytics content's <script> tag and HTML for you to copy and paste in to your custom application or portal web page's HTML page.

If the **Developer** option isn't displayed in the workbook's **Menu**, then you need to enable it. See Enable Oracle Analytics Developer Options.

1. Go to Oracle Analytics and open the workbook containing the analytics content you want to embed.
2. Click the workbook's **Menu** and then click **Developer**.
3. In the Developer window, click the Embed tab.
4. Locate the **Embedding Script to Include** field and click **Copy** to copy the <script> tag to paste in to the HTML page.

5. Optional: If you want the embedded workbook to show the workbook's default view, then locate the **Default** field, click **Copy** to copy the HTML, and paste it in to the HTML page.
6. Optional: If you want the embedded workbook to show an item such as a specific canvas, then locate the item's field, click **Copy** to copy the HTML, and paste it in to the HTML page.

Prepare the HTML Page for Embedded Oracle Analytics Content

To embed Oracle Analytics content, you must create or update the HTML page to include the required DOCTYPE declaration, dir global attribute, and reference the embedding.js JavaScript source and the embedded workbook's URL. You must also specify the embedding mode (JET or standalone), an authentication method, and add any attributes.

This topic contains the following information:

Doctype Declaration

Set the doctype declaration to `<!DOCTYPE html>`. Unpredictable behavior such as the page not rendering properly results if you use a doctype declaration other than `<!DOCTYPE html>`, or if you forget to include a doctype declaration.

Dir Global Attribute

Set the `dir` global attribute as required by the web page's locale. The `dir` global attribute indicates the embedded analytics content's layout direction.

> **Note:**
>
> If you need to support multiple locales, then use JavaScript to set the attribute.

The attribute's value options are:

- `rtl` - Use for right to left layout direction.
- `ltr` - Use for left to right layout direction.
- `auto` - Don't use. This value isn't supported by Oracle Analytics.

<script> Tag and JavaScript Source Reference

> **Note:**
>
> Oracle Analytics generates the `<script>` tag and JavaScript source's URL that you need to include.

Add a `<script>` tag that references embedding.js to your HTML page.

The JavaScript source's URL structure is:

- `"https://<instance>/public/dv/v1/embedding/<embeddingMode>/embedding.js"`. The examples in this document use this URL.
- For older deployments, use: `"http://<instance>/ui/dv/v1/embedding/<embeddingMode>/embedding.js"`.

Where `<embeddingMode>` must be either `jet` or `standalone`:

- Use `jet` if you're embedding analytics content within an existing Oracle JET application. If you use `jet`, then the version of Oracle JET that the application uses must match the same major version of Oracle JET that Oracle Analytics uses. For example, if Oracle Analytics uses JET 11.0.0, then your custom application must use JET 11.0.0 or 11.1.0. Oracle Analytics uses Oracle JET version 11.1.10.

  To find the version of JET that Oracle Analytics uses, log into Oracle Analytics, open the browser console, and run this command:

  ```
  requirejs('ojs/ojcore').version
  ```

  If the embedding application uses Oracle JET, Oracle Analytics extends the application with the components it needs. See Embed Oracle Analytics Content into a Custom Application that Uses Oracle JET.

  Oracle JET is a set of Javascript-based libraries used for the Oracle Analytics user interface.

- Use `standalone` when embedding visualization content in a generic application that doesn't use Oracle JET.

  If the embedding application doesn't use Oracle JET, then Oracle Analytics brings its JET distribution to the page with additional components. See Embed Oracle Analytics Content into a Custom Application That Doesn't Use Oracle JET.

Authentication

You need an authenticated session to view the embedded analytics content. See Use Login Prompt Authentication With Embedded Oracle Analytics Content.

<oracle-dv> Element

To embed a workbook, you must add the following HTML snippet with attribute values inside an appropriately sized element. Oracle Analytics generates the HTML that you need to include.

```
<oracle-dv project-path="" active-page="" active-tab-id="" filters=""></oracle-dv>
```

Supported attributes — These attributes support static strings and properties defined within a Knockout model. Knockout is a technology used in Oracle JET.

> **Note:**
>
> See Embed Oracle Analytics Content into a Custom Application That Doesn't Use Oracle JET for an example of binding these attributes to a Knockout model.

- `project-path`: Specifies the path of the workbook that you want to render.
- `active-page`: (Optional) Specifies whether an insight other than the default is rendered. When you specify `active-page`, you also use `active-tab-id` to specify the exact Present canvas that you're showing. Valid value is `insight`.

  > **Note:**
  >
  > The `active-page` value `canvas` is deprecated. Oracle recommends that you modify your embedding code that uses `canvas` to `insight`. Existing embedded analytics content that uses `canvas` will continue to work and a warning displays in the browser console.

- `active-tab-id`: (Optional) Specifies the ID of the Present canvas that you're showing.
- `filters`: (Optional) Allows the programmatic passing of filter values to an embedded workbook.
- `project-options`: (Optional) In this attribute, *project* refers to *workbook*. Allows you to pass these options:
  - `bDisableMobileLayout`: Disables or enables the mobile layout. Mobile layout refers to the summary card layout available only on phone devices. Value should be `true` or `false`.
  - `bShowFilterBar`: Shows or hides the filter bar. Value should be `true` or `false`.
  - `showCanvasNavigation`: Shows or hides the canvases in the workbook according to the canvas navigation setting in the workbook's Present tab. Value should be `true` or `false`.

  For example, `<oracle-dv project-path="{{projectPath}}" active-page="canvas" active-tab-id="1" filters="{{filters}}" project-options='{"bDisableMobileLayout":true, "bShowFilterBar":false}'></oracle-dv>`
- `brushing-type`: Controls how brushing works. The value you specify overrides all other settings, including system defaults and settings in the saved workbook. Value should be the string `on`, `off`, or `auto`.
  - `on`: Use to issue brushing queries with normal priority. Brushing queries and visualization queries are mixed and run at the same time.
  - `auto`: Default. Use to issue brushing queries with low priority. When a user interacts with a visualization, there may be a delay showing marks in other visualizations until the brushing queries complete.

- `compatibility-mode`: Use when different major versions of Oracle JET are present. This creates an iFrame at runtime to sandbox the embedded analytics content. Value should be the string `yes`, `no`, or `auto`.

  > **Note:**
  >
  > When setting this attribute, note these two items:
  >
  > If using compatibility mode, confirm that **Allow Frames** is selected for the application your administrator registered as a safe domain. See Register an Application as a Safe Domain.
  >
  > To find the version of JET that Oracle Analytics uses, log into Oracle Analytics, open the browser console, and run this command:
  >
  > ```
  > requirejs('ojs/ojcore').version
  > ```

  - `yes`: Use when you always want to sandbox the analytics embedded content. This is useful when embedding into Oracle APEX applications.
  - `no`: Default. Use when you don't want to create an iFrame.
  - `auto`: Use to automatically detect major differences in Oracle JET version between the host embedding application and Oracle Analytics. You can use this when embedding into Oracle APEX.

Example

In this example, all instances of *project* refer to *workbook*.

You can get the exact URL of the embedding.js file from the Embed tab in the Developer window of the workbook.

```
<!DOCTYPE html>
<html dir="ltr">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
        <title>Embedded Oracle Analytics Workbook Example</title>
        <script src="https://<instance>/public/dv/v1/embedding/<embedding
mode>/embedding.js" type="application/javascript">
        </script>

    </head>
    <body>
        <h1>Embedded Oracle Analytics Workbook</h1>
        <div style="border:1px solid black;position: absolute; width:
calc(100% - 40px); height: calc(100% - 120px)" >
            <!--
            The following <oracle-dv> tag is the tag that will embed the
specified workbook.
            -->
            <oracle-dv
               project-path="<project path>"
               active-page="insight"
               active-tab-id="snapshot!canvas!1">
```

```
            </oracle-dv>
        </div>
        <!--
        Apply Knockout bindings after DV workbook is fully loaded.  This
should be executed in a body onload handler or in a <script> tag after the
<oracle-dv> tag.
        -->
        <script>
        requirejs(['knockout', 'ojs/ojcore', 'ojs/ojknockout', 'ojs/
ojcomposite', 'jet-composites/oracle-dv/loader'], function(ko) {
        ko.applyBindings();
        });
        </script>
    </body>
</html>
```

Pass Filters to the HTML Page for Embedded Oracle Analytics Content

You can pass numeric and list filters to the HTML page where you're embedding Oracle Analytics content. You can filter any type of data with these filter types.

The filters payload is a Javascript array containing one filter Javascript object per array item.

In this example, all instances of *project* refer to *workbook*. Rendering a workbook while applying filters looks like this:

```
<oracle-dv project-path="{{projectPath}}" filters="{{filters}}">
</oracle-dv>

<script>
requirejs(['knockout', 'ojs/ojcore', 'ojs/ojknockout', 'ojs/ojcomposite',
'jet-composites/oracle-dv/loader'], function(ko) {
   function MyProject() {
      var self = this;
      self.projectPath =  ko.observable("/users/weblogic/EmbeddingStory");
      self.filters = ko.observableArray([{
         "sColFormula": "\"A - Sample Sales\".\"Products\".\"P2  Product
Type\"",
         "sColName": "P2  Product Type",
         "sOperator": "in", /* One of in, notIn, between, less, lessOrEqual,
greater, greaterOrEqual */
         "isNumericCol": false,
         "bIsDoubleColumn": false,
         "aCodeValues": [],
         "aDisplayValues": ['Audio', 'Camera', 'LCD']
      },{
         "sColFormula": "\"A - Sample Sales\".\"Base Facts\".\"1- Revenue\"",
         "sColName": "Rev",
         "sOperator": "between", /* One of in, notIn, between, less,
lessOrEqual, greater, greaterOrEqual */
         "isNumericCol": true,
         "bIsDoubleColumn": false,
         "aCodeValues": [],
```

```
        "aDisplayValues": [0, 2400000] /* Because the operator is "between",
this results in values between 0 and 2400000 *
/
  }]);
}
   ko.applyBindings(MyProject);
});
</script>
```

Supported attributes — Each filter object within the filters payload must contain the following attributes:

- `sColFormula`: Specifies the three-part formula of the column to filter. The column formula must include three parts.

  If you're unsure of the formula, create a workbook that uses that column, and then in the Visualize tab, click the workbook's **Menu** and then click **Developer**. In the Developer page, click the JSON tab to view the column's expression. For example, `sColFormula": "\"A - Sample Sales\".\"Base Facts\".\"1- Revenue\"" .`

  If the **Developer** option isn't displayed in the workbook's **Menu**, then you need to enable it. See Enable Oracle Analytics Developer Options.

- `sColName`: (Required) Specifies a unique name for this column.
- `sOperator`: Use `in`, `notIn`, `between`, `less`, `lessOrEqual`, `greater`, or `greaterOrEqual`.
  - `in` and `notIn` - Apply to list filters.
  - `between`, `less`, `lessOrEqual`, `greater`, and `greaterOrEqual` - Apply to numeric filters.
- `isNumericCol`: Specifies if the filter is numeric or list. Value should be `true` or `false`.
- `isDateCol`: (Required) Indicates whether the column is a date column. Value should be `true` or `false`. Use `true` if the column is a date, but not for year, month, quarter, and so on. If set to `true`, then specify date or dates in the `aDisplayValues` attribute.
- `bIsDoubleColumn`: Specifies if the column has double column values behind the display values. Value should be `true` or `false`.
- `aCodeValues`: When `bIsDoubleColumn` is `true`, this array is used.
- `bHonorEmptyFilter`: (Optional) Indicates whether an empty filter is honored (for example, empty `aCodeValues/aDisplayValues` based on the `bIsDoubleColumn` flag). This attribute applies to all column filters: list filters, number range filters, and date range filters. Value should be `true` or `false`.
  - If set to `true` and the user passes empty `aCodeValues/aDisplayValues`, then all values are part of the filter.
  - If set to `false` and user passes empty `aCodeValues/aDisplayValues`, then the attribute won't be applied and there is no change in filter values.
  - If this attribute isn't present, then the default value is `false`.
- `aDisplayValues`: When `bIsDoubleColumn` is `false`, then this array is used to filter and to display values within the user interface.

  When `bIsDoubleColumn` is `true`, then the values in this array are used for display in the user interface while the values in `aCodeValues` are used for filtering. When `bIsDoubleColumn` is `true`, there must be the same number of entries in this array as there are in the `aCodeValues` array and the values must line up. For example, suppose

`aCodeValues` has two values `1` and `2`, then `aDisplayValues` must have two values `a` and `b`, where `1` is the code value for `a`, and `2` is the code value for `b`.

If `isDateCol` attribute is set to `true`, then specify the `aDisplayValues` array with dates. If either no time zone in the time stamp or no time stamp is provided, then time is set with the local time zone. Use any of the following formats:

- mm/dd/yyyy (For example, 12/31/2011.)
- yyyy-mm-dd (For example, 2011-12-31.)
- yyyy/mm/dd (For example, 2011/12/31.)
- mm/dd/yyyy or yyyy/mm/dd, hh:mm:ss (For example, 12/31/2011 or 2011/12/31, 23:23:00.)
  Note: Use a 24 hour format. You can use a space as the separator.
- mm/dd/yyyy or yyyy/mm/dd, hh:mm:ss AM/PM (For example, 12/31/2011 or 2011/12/31, 11:23:00 PM.)
  Note: Use a 12 hour format. You can use a space as the separator.
- <3 letter month name> dd yyyy (For example, Mar 25 2015.)
- dd <3 letter month name> yyyy (For example, 25 Mar 2015.)
- Fri Sep 30 2011 05:30:00 GMT+0530 (India Standard Time)
- ISO Date Format - 2011-10-05T14:48:00.000Z

Pass Parameters to the HTML Page for Embedded Oracle Analytics Content

You can pass parameter values to the HTML page where you're embedding Oracle Analytics content. The parameter values that you pass can be utilized within query expressions and various parts of the product.

The parameters payload is a Javascript Object containing paired attributes of parameter names and values.

In this example, all instances of *project* refer to *workbook*. Rendering a project while applying parameters look like this:

```
<oracle-dv project-path="{{projectPath}}" active-page="canvas" active-tab-
id="3" parameters="{{parameters}}"
project-options='{"bDisableMobileLayout":false, "bShowFilterBar":false}'>
</oracle-dv>

<script>
requirejs(['knockout', 'ojs/ojcore', 'ojs/ojknockout', 'ojs/ojcomposite',
'jet-composites/oracle-dv/loader'], function(ko) {

   function MyProject() {
      var self = this;
      self.projectPath = ko.observable("/users/weblogic/EmbeddingStory");
      self.parameters = ko.observable({
         "p1n": "Office",
         "p1v": "Bristol Office",
         "p2n": "Year",
         "p2v": [2023, 2022]
      });
```

```
    }
    ko.applyBindings(MyProject);
});
</script>
```

Supported attributes — Each parameter object within the parameters payload must contain the following attributes:

- `p <number> n`: (Required) Specifies the parameter's name as defined in the workbook. For example, "Office" or "Year".
- `p <number> v`: (Required) Specifies the parameter value that you want to pass. For example "Bluebell Office" or "10" or [2023, 2022].
- `p <number> d`: (Optional) Use with parameters with double columns. Specifies the parameter's display value corresponding to `p <number> v`. For example, "My Office".

Refresh Data in the HTML Page for Embedded Oracle Analytics Content

In the HTML page where you're embedding Oracle Analytics content, you can specify how to refresh the embedded workbook's data.

To refresh data without reloading a workbook, the `refreshData` function is attached to all `<oracle-dv>` elements. Invoking it forces all visualizations under that element to refresh.

This is the code to refresh data for a single embedded workbook. In this code, all instances of *project* refer to *workbook*.

```
<oracle-dv id="project1" project-path="{{projectPath}}">
</oracle-dv>

<script>
   function refreshProject() {
      $('#project1')
  [0].refreshData();
}
</script>
```

This is the code to refresh data for multiple embedded workbooks. In this code, all instances of *project* refer to *workbook*.

```
<script>
   function refreshProject()
   {
      $('oracle-dv').each(function() {
         this.refreshData();
         });
}
</script>
```

Embed Oracle Analytics Content into a Custom Application that Uses Oracle JET

If the custom application uses Oracle JET, then the embedded Oracle Analytics content extends the application with the component it needs.

Before you begin to embed analytics content, confirm that the custom application uses the same major version of JET that Oracle Analytics uses. For example, if Oracle Analytics uses JET 11.0.10, then your custom application must use JET 11.x.x.
To find the version of JET that Oracle Analytics uses, log into Oracle Analytics, open the browser console, and run this command:

```
requirejs('ojs/ojcore').version
```

Your JET application must also use the same style that Oracle Analytics uses, which is Alta.

For information about creating an Oracle JET quick start application where you'll embed analytics content, see Oracle JET Get Started.

This procedure uses an example embedding application named OAJETAPP.

1. Follow the instructions to install the Oracle JET quickstart application and name the embedding application OAJETAPP using `--template=navdrawer`.
2. Edit the index.HTML file of the embedding application (for example, `OAJETAPP/src/index.html`) and include embedding.js.

   ```
   <script src="https://<instance>/public/dv/v1/embedding/jet/embedding.js"
   type="text/javascript">
   </script>
   ```

3. Include `<oracle-dv>` in the appropriate section (for example `OACJETAPP/src/js/views/dashboard.html`). Here `project-path` specifies the workbook's path.

   ```
   <div class="oj-hybrid-padding" style="position: absolute; width: calc(100%
   - 40px); height: calc(100% - 120px)">
    <h3Dashboard Content Area</h3>
     <oracle-dv id="oracle-dv" project-path="/Shared Folders/embed/test-
   embed">
     </oracle-dv>
   </div>
   ```

4. Run the quick start application using these commands.

   ```
   ojet build
   ojet serve
   ```

Embed Oracle Analytics Content into a Custom Application That Doesn't Use Oracle JET

If the custom application uses a technology other than Oracle JET, then the embedded Oracle Analytics content adds its Oracle JET distribution and all additional components into the page.

If the **Developer** option isn't displayed in the workbook's **Menu**, then you need to enable it. See Enable Oracle Analytics Developer Options.

1. Include the standalone version of embedding.js.

   ```
   <script src=https://<instance>/public/ui/dv/v1/embedding/standalone/
   embedding.js type="text/javascript"> </script>
   ```

2. Find and include `<oracle-dv>` under an appropriately sized `<div>`. To find this tag:
   a. Go to Oracle Analytics and open the workbook containing the analytics content you want to embed.
   b. Click the workbook's **Menu** and then click **Developer**.
   c. Click the Embed tab.
   d. Locate the item you want to embed and click **Copy** to copy it.

   Example

   Here `project-path` specifies the workbook's path.

   ```
   <div style="position: absolute; width: calc(100% - 40px); height:
   calc(100% - 120px)">
       <oracle-dv project-path="/@Catalog/users/admin/workbook_name">
       </oracle-dv>
   </div>
   ```

3. Apply Knockout bindings after the visualization is fully loaded. This should be placed inside of a `<script>` tag after the `<oracle-dv>` tag, or executed in an onload body handler.

   ```
   requirejs(['knockout', 'ojs/ojcore', 'ojs/ojknockout', 'ojs/ojcomposite',
   'jet-composites/oracle-dv/loader'], function(ko) {
       ko.applyBindings();
   });
   ```

Complete Example

```
<!DOCTYPE html>
<html dir="ltr">
   <head>
      <title>AJAX Standalone Demo</title>
      <script src="https://<instance>/public/dv/v1/embedding/standalone/
embedding.js""
type="text/javascript">
      </script>
   </head>
   <body>
      <h1>AJAX Standalone Demo</h1>

      <div style="position: absolute; width: calc(100% - 40px); height:
```

```
calc(100% -
120px)" >
         <oracle-dv project-path="/shared/embed/test-embed">
         </oracle-dv>
      </div>

   <script>
requirejs(['knockout', 'ojs/ojcore', 'ojs/ojknockout', 'ojs/ojcomposite',
'jet-composites/oracle-dv/loader'], function(ko) {   ko.applyBindings();
});
      </script
   </body
</html
```

Add Authentication to an Application or Web Page Containing Embedded Oracle Analytics Content

Use the topics in this section to add an authentication method to your web application or portal web page that contains embedded Oracle Analytics content.

Topics:

Use Login Prompt Authentication With Embedded Oracle Analytics Content

Login prompt authentication is the default authentication method for Oracle Analytics content embedded in a web application or portal web page.

When users access embedded analytics content, they're presented with a login screen where they enter login name and password before they can view data. If there is no common identity management between Oracle Analytics and the web application or portal web page, then users are presented with this login screen, even if they've already logged in to the web application or portal web page containing the embedded analytics content

Customize the Login Prompt Authentication Message

Add attributes to the `<oracle-dv>` tag to customize the login prompt authentication messages. The following attributes are supported:

- `auth-message-prefix`: Specifies the prefix text for the login message. The default value is `"Oracle Analytics"`.
- `auth-message-link`: Specifies the text for the login link. The default value is `"Login"`.
- `auth-message-suffix`: Specifies the suffix text for the login message. The default value is `"Required"`.
- `auth-needed-message`: Specifies the text for the authentication needed message. The default value is `"Requires Authentication"`.
- `auth-message-prefix-small`: Specifies the prefix text for the login message. The default value is `"Oracle Analytics"`. Applicable only if the embedded container size is smaller than 215 pixels.
- `auth-message-link-small`: Specifies the text for the login link. The default value is `"Login"`. Applicable only if the embedded container size is smaller than 215 pixels.

- `auth-message-suffix-small` - Specifies the suffix text for the login message. The default value is the empty string. Applicable only if the embedded container size is smaller than 215 pixels.
- `auth-needed-message-small`: Specifies the text for the authentication needed message. The default value is `"Requires Authentication"`. Applicable only if the embedded container size is smaller than 160 pixels.