PUBLIC

SAP BusinessObjects Business Intelligence platform

Document Version: 4.3 Support Package 4 – 2023-12-07

SAP BusinessObjects BI Developer's Guide for Web Intelligence and the BI Semantic Layer

THE BEST RUN SAP

Content

1 Document Version History. . . . 4

2 About This Guide. . . . 5

3 Audience. . . . 6

4 Conventions in This Guide. . . . 7

5 Opening a Web Intelligence document using OpenDocument URL. . . . 8

6 Customizing Web Intelligence User Interfaces. . . . 9

6.1 Customizing Web Intelligence interface elements by user groups and folders. . . . 9

Customization interface. . . . 11

Customization rules. . . . 11

To customize the Web Intelligence interface appearance. . . . 12

6.2 Web Intelligence content alignment. . . . 13

7 Customizing Web Intelligence User Interface with Extensions 14

7.1 About Extensions 14

Contributions. . . . 14

Deployment Scenarios. . . . 15

JavaScript APIs 15

7.2 UI Extension Points Task Sequence 16

7.3 Prerequisites. . . . 16

7.4 Writing Your Extension 16

Create the Web Application Folder 17

Declare the Extension 17

Creating a Target Page 18

Generating a WAR File 20

7.5 Deploying the Extension 20

To Deploy the Extension Using Apache Tomcat Manager. . . . 21

Deploying the Extension on the Application Server 21

Registering Extension in the CMC 21

7.6 Differences with SAP BusinessObjects BI 4.2 22

Extension UI 22

Non-OSGI Deployment 22

Asynchronous Mode 23

Changes in Web Intelligence JavaScript API 24

7.7 About the Web Intelligence Extension Sample 25

8 Creating Web Intelligence Visualizations with a Custom Element Service 26
8.1 About the Custom Element Service APIs. 28
8.2 Getting the Supported Formats. 29
8.3 Getting the Visualization Types. 30
8.4 Getting the Visualization Icon. 31
8.5 Getting the Visualization Feed Definitions. 32
8.6 Rendering the Visualization. 33
8.7 Getting the Rendering Settings. 37

9 Adding External Functions to Web Intelligence Formula Language. 43
9.1 To create an external function for Web Intelligence. 43
9.2 External function folder. 44
9.3 Defining the XML function declaration. 44
9.4 Defining the XML catalog declaration. 47
9.5 Writing the C++ source files. 48
9.6 Compiling the C++ source files. 49
9.7 Deploying the external library. 50
9.8 Error messages. 51
9.9 Trace log message errors. 51
9.10 Using the Web Intelligence samples files. 52

10 Exposing Web Intelligence Features with REST Web Services. 54

11 Consuming BI Semantic Layer Universes with REST Web Services. 55

12 Developing Applications to Design and Administrate Universes. 56

13 Creating a Data Access Driver with JavaBean. 57

1 Document Version History

The following table provides an overview of the most important document changes.

Version	Date	Change
SAP BusinessObjects BI Developer's Guide for Web Intelligence and the BI Semantic Layer 4.3 SP2	December 2021	Added chapter about UI extension.
SAP BusinessObjects BI Developer's Guide for Web Intelligence and the BI Semantic Layer 4.3 SP1	December 2020	Added chapter about UI customization. See Customizing Web Intelligence interface elements by user groups and folders [page 9]
SAP BusinessObjects BI Developer's Guide for Web Intelligence and the BI Semantic Layer 4.3	April 2020	Initial release Chapter 7 contains updated information about using a JavaBean driver. Creating a Data Access Driver with JavaBean [page 57]

2 About This Guide

The SAP BusinessObjects BI platform 4.3 is the key foundation for your analytics applications. It comes with a comprehensive set of tools from which you can pick up the one that suits the technologies you are ready to use and your business objectives. To this end, the *SAP BusinessObjects BI Developer's Guide for Web Intelligence and the BI Semantic Layer* is your new entry point to learn how to develop applications, using SDKs, samples, and extension framework, to enforce and take advantage of the Web Intelligence and BI Semantic Layer capabilities.

This guide provides information and references about:

- How to open Web Intelligence documents through OpenDocument URL
- How to customize Web Intelligence user interface via the Central Management Console
- How to extend Web Intelligence user interface using extensions
- How to create your visualizations with custom elements
- How to create external functions for Web Intelligence formula language
- How to use REST APIs to work with Web Intelligence documents and reports in non-SAP client tools
- How to use REST APIs to access universes and run queries in non-SAP client tools
- How to create, edit, secure and deploy universes with the BI Semantic Layer Java SDK
- How to create a JavaBean driver or an Open driver with the Driver Development Kit

3 Audience

As it serves as en entry point to the Web Intelligence and BI Semantic Layer customization area, the *BI Developer's Guide* is intended for various readers.

This guide is for you if:

- You are an SAP BusinessObjects administrator who wants to use Web Intelligence in their corporate portal
- You are a JavaScript developer responsible for developing extensions to Web Intelligence user interfaces
- You are a Java developer responsible for developing applications that perform creation, editing, and publication tasks on UNX and UNV universes
- You are a developer responsible for writing programs that access and consume the BI platform web services
- You are a developer responsible for developing data access drivers to help the BI platform to communicate with your company's data sources
- You are an SAP consultant who wants to help SAP partners and customers in their BI platform customization project.
- You are an SAP partner who would like to provide customizations and extensions of Web Intelligence to your customer

4 Conventions in This Guide

In this guide, the placeholder `<bip-install-dir>` is the install root path of the SAP BusinessObjects Bi platform. On Microsoft Windows, the default `<bip-install-dir>` stands for the `C:\Program Files (x86)\SAP Business Objects\SAP BusinessObjects Enterprise XI 4.0` directory.

The placeholder `<tomcat-dir>` stands for the `C:\Program Files (x86)\SAP BusinessObjects\tomcat` directory.

5 Opening a Web Intelligence document using OpenDocument URL

You can use the OpenDocument URL to create a hyperlink for content in the SAP BI Platform repository, including a folder, a Web Intelligence document, report, or block. Share this hyperlink with other users who can access the content without navigating in the repository after authentication.

If you are developing an application, you can also utilize Web Intelligence content by generating an OpenDocument URL to open a meaningful document.

The OpenDocument URL syntax offers numerous options for defining both the content to open and how to open it. For example, you can refresh a document or answer to prompts.

For more details, refer to *Viewing Documents Using OpenDocument* guide.

6 Customizing Web Intelligence User Interfaces

You can simplify the appearance of the web client by hiding some functionalities through the CMC.

6.1 Customizing Web Intelligence interface elements by user groups and folders

Customization allows you to hide multiple interface elements to simplify the way end-users interact with the application, depending on user groups and folders containing Web Intelligence documents. You can hide data sources types, edit mode toggle, turn-off the auto refresh feature, and many more.

By default, every interface element is enabled. If you want to hide them, you can do so in the Central Management Console. The table below lists the user interface elements you can hide.

Features List	Description
Mode	Hides the available modes the user can access through the drop-down button. • Reading To hide the Reading mode from the drop-down button. • Design To hide both the Design and Structure modes from the drop-down button. • Data To hide the Data mode from the drop-down button. If all modes are disabled, then documents can only be opened in Reading mode.
Location	Hides a whole category of data sources. Categories you can disable are: • BI Platform Repository • Local (only available in Rich Client) • Web Services • Google Drive • Microsoft OneDrive

Features List	Description
Data Source	In Design mode, you can restrict the data sources available in the *Select a Data Source* and the *Change Source* dialog boxes. The data sources you can disable are: • Universes • Web Intelligence documents • Excel files • Text files • SAP BW • SAP HANA views • Free Hand SQL queries • OData • Google Spreadsheet
Query	• Refresh In Reading mode, hides the *Data* section in the toolbar. In Design mode, hides the *Refresh* dropdown menu, the *Refresh All* command, the *Run* button and its dropdown menu in the Query Panel. • Advanced Refresh In Design mode, hides the *Advanced Refresh* command in the *Refresh* dropdown menu. • Auto-Refresh Hides the *Auto-Refresh* option in the Presentation Mode. • Change Source In Design mode, hides the ability to change the document's data sources.
Data	In Data mode, hides the Combine Cubes features.
Analysis	• Drill In both Reading and Design mode, hides the *Drill* checkbox in the Analyze section of the toolbar, drill filters in the *Filter Bar*. Also, in the report, values that can be drilled are not displayed as hyperlinks and the drill actions and icons available for these values are hidden. In Design mode, hides the drill filters in the *Build* panel, under *Data Filters*. • Track Data Changes In both Reading and Design mode, hide the *Track Data Changes* and *Show Changes* from the toolbar.
Documents	• *New*, *Open*, *Save*, *Favorites*, *Presentation Mode*. Hides the corresponding buttons from the toolbar. • Comments In both Reading and Design modes, hide the *Comments* tab in the side panel, and the *Comments* command in the contextual menu. • Shared Elements In Design mode, hide the *Shared Elements* tab in the side panel, and the *Shared Elements* command in the *Insert* section of the toolbar.

Features List	Description
Export To	In any modes, hides the possibility to export documents report and cubes to: • Excel • PDF • HTML • TXT • CSV
Generate Link	In Design mode, hides the ability to create OpenDocument link and generate OData links for queries and individual report elements from contextual menus.
Schedule & Publishing	Hides the possibility to schedule and publish documents to TXT, XLS, PDF, HTML, MHTML, and CSV.

6.1.1 Customization interface

You can select individual folders so that the documents they contain automatically benefit from the customization. Simply select one or more folders in the *Customized folders* area, and move on to the *Features* tab to start customizing. By default, the customization applies to every document in the folder you have selected.

The *Features* tab lists all the features you can enable or disable. Use the dedicated checkboxes to toggle them on or off.

6.1.2 Customization rules

The following rules are used to define customizations to apply to a user:

- If the user belongs to different groups, only the customization defined to the group whose ID is lower applies. The customization defined for the other groups containing the user does not apply.
- For nested folder structure, the immediate parent folder of the document that has been added in the list of customized folders defines customizations for the document for user interface elements, features, and extensions.
- The customization defined for Default Folders applies for the documents stored in Personal Documents and Inboxes, and for documents for which the parent folder is not customized.
- The customization defined for user interface elements have priority over customization defined for features as feature is only a shortcut to enable all user interface elements.
- Scenario: When the customization elements are displayed as a tree list and you disable a node on a system. Here, if you upgrade this system with a newer version of the product having new items in the nodes, then by default these items are activated even if the upper node is disabled.

6.1.3 To customize the Web Intelligence interface appearance

You can customize the appearance of the Web Intelligence user interface by hiding menu items, subitems, and features for a selected user group and document folder.

1. Log into the CMC as an Administrator.
2. From the *Organize* list, select *Users and Groups*.
3. In the *Group Hierarchy* list, select a user group.
4. In the *Actions* list, select *Customization*.
5. In the *Customized folders* section, do one of the following:

Option	Description
To define a default customization	1. Select *Default Folders* in the *Customized folders* area.
To add the document folders for which you want to apply customization for the selected user group	1. Click *Add Folder*. 2. Select the folders. The folders displays in the *Customized folders* area.
To avoid redefining the same customization for other folders	1. In the *Customized folders* area, select the folder from which you want to copy the customization. 2. In the dropdown list, click *Duplicate Customization*. 3. Select the folder to which you want define the customization. 4. Click *Paste Customization*. 5. Go to step 7.
To remove the customization for a specific folder	1. In the *Customized folders* area, select the folder. 2. In the dropdown list, click *Remove Folder*. 3. Go to step 7. **Note** You can't remove *Default Folders*.

6. Select or deselect items in the *Features* tab to display or hide them in Web Intelligence.

   If you deselect all children of a parent item, the parent item is also deselected and hidden in Web Intelligence. For more information, check out Customizing Web Intelligence interface elements by user groups and folders [page 9].
7. Click *Save & Close*.

When you save the customization, all users of the selected group will see these changes the next time they log on to BI launch pad and open Web Intelligence.

Note

We recommend that you log on to BI launch pad as a user from the group you have just customized, start Web Intelligence, and verify that the interface corresponds to your customization settings.

6.2 Web Intelligence content alignment

Choose the way document content will be aligned (left-to-right or right-to-left) when users create Web Intelligence documents.

For the Rich Client interface, the content alignment is determined by the locales set in the BI launch pad preferences:

- The system uses right-to-left alignment only when both the Preferred Viewing Locale and Product Locale are set to right-to-left languages.
- In all other cases, the content alignment is left-to-right.

> ⓘ Note
>
> For information about how to set locales, see the *Business Intelligence Launch Pad User Guide*.

> ⓘ Note
>
> Content alignment applies only at document creation time, and doesn't affect existing documents.

7 Customizing Web Intelligence User Interface with Extensions

Starting SAP BusinessObjects BI 4.3 Support Package 2, Web Intelligence supports again user interface extension that allows you to extend the Web Intelligence interface with your own code.

The Web Intelligence JavaScript API that exposes Web Intelligence capabilities is also back in SAP BusinessObjects BI 4.3 SP2. You can use it to write Web Intelligence extensions and Custom Elements.

> ⚠ Restriction
>
> Web Intelligence extensions and Web Intelligence JavaScript API are not supported in Web Intelligence Rich Client.
>
> Some adaptations are required to re-use Web Intelligence extensions written for SAP BusinessObjects BI 4.2.

Related Information

Creating Web Intelligence Visualizations with a Custom Element Service [page 26]
Differences with SAP BusinessObjects BI 4.2 [page 22]

7.1 About Extensions

You can customize the interfaces of SAP BusinessObjects Web Intelligence using extensions.

7.1.1 Contributions

An extension can contribute to the Web Intelligence interface as:

- A sub-tab in the *Main* > *Extensions* tab.
- A button or toggle button in a dedicated drop-down button menu in the toolbar.
- A new perspective available in the *Mode* selection drop-down button menu. Such perspective has no side panels and no toolbar. It only displays the *Mode* selection drop-down button menu. The extension code is run inside this custom perspective.

You can develop several extensions to contribute to several perspectives. Perspectives are added to the drop-down button menu in their registration order.

7.1.2 Deployment Scenarios

As opposed to SAP BusinessObjects BI 4.2, Web Intelligence extensions are only supported in non-OSGI mode. When writing the extension, you must provide a JSON file that declares the extension's properties.

You must then package the extension code and its declaration file as a web application in a WAR (Web Application Archive) file.

You can deploy this WAR file either on the SAP BusinessObjects BI application server (Apache Tomcat) or on an external application server by copying it into the application server folder or via its application server manager.

As an Administrator, you make the extension available through a URL registered on the CMC.

> ⚠ Restriction
>
> Web Intelligence extensions and Web Intelligence JavaScript API are not supported in Web Intelligence Rich Client.

Related Information

Differences with SAP BusinessObjects BI 4.2 [page 22]

7.1.3 JavaScript APIs

You make the extension work with Web Intelligence by using its JavaScript API. This API proposes two main sets of functions:

- Web Intelligence Application API: To set up the Web Intelligence interface to make your extensions work with documents and reports. You can for example listen to and dispatch events, update the Web Intelligence client context, display wait cursor and dialog boxes.
- Web Intelligence Service Document API: To leverage Web Intelligence reporting and analysis capabilities in your extension.

Your extension must contain a target page such as an HTML or a JSP page that includes the appropriate JavaScript code.

Related Information

https://help.sap.com/doc/2b701d4cd19b43289805d095e164b21a/4.3.2/en-US/index.html

7.2 UI Extension Points Task Sequence

The steps to create and deploy an extension to customize the Web Intelligence interface are:

1. Build your development environment.
2. Write the extension:
    1. Create a declaration file to define your extension.
    2. Implement your extension's behavior with the help of the Web Intelligence JavaScript API.
    3. Generate the extension as a WAR file.
3. Deploy the WAR file on the application server.
4. In the CMC, add your extension URL to the list of **Authorized URLs**.
5. In the CMC, register and enbale your extension URL for the group(s) of users.
6. Test and debug your extension.

Related Information

Prerequisites [page 16]
Writing Your Extension [page 16]
Deploying the Extension [page 20]

7.3 Prerequisites

- To deploy and run Web Intelligence extensions, you must have at least SAP BusinessObjects BI 4.3 SP2 Patch 11 or SAP BusinessObjects BI 4.3 SP3 Patch 3 that correct several issues.
- An application server, such as Apache Tomcat 9, is required to run the extension. You can use the one running your SAP BusinessObjects BI 4.3 server; otherwise, you need to install another one.
- If your extension uses Java, a Java version supported by the application server.
- To help you write your extension, you can use an Eclipse IDE for Enterprise Java Developers, or any Integrated Development Environment tool you use to write JavaScript/Java code.

7.4 Writing Your Extension

You can manually create your WAR file or use your favorite Integrated Development Environment tool to do so. If you are using Eclipse, you may declare a Java Virtual Machine and a Tomcat Server in your environment to debugging.

Refer to your IDE documentation for more details.

7.4.1 Create the Web Application Folder

Create the following folder structure that will be saved when generating the WAR file.

Sample Code

```
WebContent
  extension
    <ProviderName>
      <ExtensionName>
        assets
          css
          img
          js
```

where:

- `<ProviderName>` is the name of the company.
- `<ExtensionName>` is the name of your extension.

These two names identify the extension and are displayed in the CMC when you register it.

The extension content is located in the `WebContent/extension/<ProviderName>/<ExtensionName>` folder.

7.4.2 Declare the Extension

To identify your extension, you must provide a JSON declaration file.

1. Create a `declaration.json` file in the folder `WebContent/extension/<ProviderName>/<ExtensionName>`
2. This declaration.json file may contain the following lines:

Sample Code

```
{
  "properties":
  {
    "id": "WebI_Extension_Sample",
    "provider": "<ProviderName>",
    "name": "<ExtensionName>",
    "version": "1.0.0",
    "languages":
    {
      "en":
      {
        "title": "Extension sample",
        "description": "Web Intelligence extension sample"
      },
      "fr":
      {
        "title": "Exemple d'extension",
        "description": "Exemple d'extension pour Web Intelligence "
      }
    }
```

```
  },
  "contributions":
  [
    {
      "type": "sidepanebutton",
      "name": "SidePanelContributionSample",
      "iconURL": "assets/img/sample.png",
      "targetPage": "index.html",
      "perspectives": ["reading", "design"],
      "languages":
      {
        "en":
        {
          "title": "Side panel contribution",
          "description": "This is a side panel contribution"
        },
        "fr":
        {
          "title": "Contribution au panneau latéral",
          "description": "Ceci est une contribution au panneau latéral"
        }
      }
    }
  ]
}
```

Where the supported values for the contribution "type" are:

- `"sidepanebutton"` to add a sub-tab in the Main side panel > Extensions tab.
- `"statusbarbutton"` to add a button in the drop-down menu button.
- `"statusbartogglebutton"` to add a toggle button in the drop-down menu button.
- `"perspectivebutton"` to add a perspective in the mode selection drop-down menu button.

3. Copy the contribution icon `sample.png` in the `WebContent///assets/img` folder
4. In the `declaration.json` file, declare this icon path in the `iconURL` parameter of the `contributions` element.
5. Create the contribution target page `index.html` in the `WebContent//` folder.
6. In the `declaration.json` file, declare this target page path in the `targetPage` parameter of the `contributions` element.

Related Information

Creating a Target Page [page 18]

7.4.3 Creating a Target Page

Fill the target page with the code to run when this page is accessed through the extension. Use asynchronous calls when calling Web Intelligence JavaScript API functions (synchronous calls are no more supported).

1. Edit the index.html file created in the `WebContent/extension/<ProviderName>/<ExtensionName>` folder.

2. Reference the packages containing the functions of the Web Intelligence JavaScript API to use:
   - `<script type="text/javascript" src="../../../js/extension/webi.services.js"></script>`
   - `<script type="text/javascript" src="../../../js/extension/webi.application.js"></script>`
   - `<script type="text/javascript" src="../../../js/extension/webi.application.bar.js"></script>`
   - `<script type="text/javascript" src="../../../js/extension/webi.application.dialogbox.js"></script>`
   - `<script type="text/javascript" src="../../../js/extension/webi.application.sidepane.js"></script>`
3. Write your code extension. For example, you may add the following code to display the number of reports in the extension page:

Sample Code

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" type="text/css" href="assets/css/
sample.css" />
        <script type="text/javascript" src="../../../js/extension/
webi.services.js"></script>
        <script type="text/javascript" src="assets/js/sample.js"></script>

        <script type="text/javascript">
            function init() {
                getNbReports(updateLabelCallback)
            }

            function updateLabelCallback(nbReports) {
                document.getElementById("nbReportsId").innerHTML = "Number
of reports: " + nbReports
            }
        </script>
    </head>
    <body onload="init()">
        <label class="nbReportsLabel" id="nbReportsId"></label>
    </body>
</html>
```

4. In the `<head>` section of this HTML file,Create a file `sample.js` in the `WebContent/extension/<ProviderName>/<ExtensionName>/assets/js` folder. In this file, add the definition of the JavaScript function that calls the Web Intelligence JavaScript API to get the report number:

Sample Code

```
function getNbReports(callback) {
    WebiServices.document.getReports((response) => {
        callback(response.reports.report.length)
    })
}
```

5. Create a file `sample.css` in the `WebContent/extension/<ProviderName>/<ExtensionName>/assets/css` folder to define the extension's styles. For example, you may add the following content to this file:

Sample Code

```
.nbReportsLabel {
  font-family: Arial;
  font-weight: bold;
}
```

6. Write any additional code and pages that implement the extension behavior.

Related Information

https://help.sap.com/doc/2b701d4cd19b43289805d095e164b21a/4.3.2/en-US/index.html
Asynchronous Mode [page 23]

7.4.4 Generating a WAR File

Once the extension code is ready, you must package it in a WAR file to deploy it on application servers.

1. Use your operating system command-line interface to go inside the directory of your project.
2. Type the following command:

Sample Code

```
jar -cvf <ExtensionName>.war *
Where:
-c is used to create file.
-v is used to generate the verbose output.
-f is used to specify the archive name <ExtensionName>.
```

7.5 Deploying the Extension

You do not need to stop Apache Tomcat to deploy your extension.

1. Copy the WAR file from the destination folder to `<tomcat-dir>\webapps`.

   The application server unzips the WAR file automatically.
2. Check that your extension is running on the application server.

   For example, you can call the `index.html` file of your extension into a web browser. If the content of `index.html` displays, then the WAR file has been deployed successfully.

To Deploy the Extension Using Apache Tomcat Manager

You can also deploy your extension by using Apache Tomcat Manager. You need to stop the application server and add a specific Tomcat user to communicate with the Manager.

1. Stop Apache Tomcat.
2. Open the `<tomcat-dir>\conf\tomcat-users.xml` file for editing.
3. Uncomment the following line to add a Tomcat user with username and password.

```
<user username="<user>" password="<password>" roles="standard,manager"/>
```

4. Save the file.
5. Restart Apache Tomcat.
6. Launch the manager from `http://<server-name>:<port>/manager/`.
7. Upload the WAR file.

7.5.1 Deploying the Extension on the Application Server

You may deploy this WAR in:

- The application server attached to your IDE.
- The process to deploy it depends on your application server. For exemple, for Tomcat, simply copy this WAR in the webapps folder.
- The application server running the SAP BusinessObjects BI application.

The process to deploy it depends on your application server. For exemple, for Tomcat, simply copy this WAR in the `webapps` folder.

7.5.2 Registering Extension in the CMC

To access the extension in Web Intelligence, you must register it in the CMC.

1. Log into the CMC.
2. Open the *Applications* page and double-click *Web Intelligence* to open the *Properties: Web Intelligence* page.
3. In the *Authorized URLs* section, click the *Add a new URL* button in the toolbar.
4. Add the URL of the server that exposes the extension service, for example:

   `http://<host>:<port>/`

   Where `<host>` and `<port>` are the name and port of the machine hosting the extension. The URL may also contain the path of the extension.
5. Click *OK* to close the *Add URL* window.
6. Close the *Authorized URLs* window.

7. Open the *Users and Groups* page.
8. In the *Group List*, double click the group you want to enable the extension for.
9. Go to *Customization* section and select the *Extensions* tab.
10. Click *Add...*.
11. Enter the extension URL, for exemple:

    ```
    http://<host>:<port>/<Path>/extension/<ProviderName>/<ExtensionName>/
    declaration.json
    ```
12. Read the *Security Warning*, select the *I accept the risk* checkbox and click *OK*.
13. Ensure that the extension is listed and select the checkbox beside its name to enable it.
14. Click *Save & Close*.

    The extension is now available for all the users of the selected group.

    When they start Web Intelligence, the user interface is modified depending on this extension's contributions.

7.6 Differences with SAP BusinessObjects BI 4.2

Extensions written for SAP BusinessObjects BI 4.2 require some adaptations to be reused in SAP BusinessObjects BI 4.3.

7.6.1 Extension UI

In SAP BusinessObjects BI 4.3, the extension entry points in the user interface are slightly different from SAP BusinessObjects BI 4.2:

- Left side panel tabs are displayed as sub-tabs in the *Main* side panel > *Extensions* tab.
- Status bar buttons and toggle buttons are displayed in a button drop-down menu in the main toolbar.
- Perspectives buttons are displayed in the *Mode* selection button drop-down menu.
- The `welcome` and `pdf` perspectives are no more supported since they do not exist anymore in SAP BusinessObjects BI 4.3.

7.6.2 Non-OSGI Deployment

- The OSGI deployment is no more supported for Web Intelligence extension.
- Extensions must be generated as a WAR file and deployed on a web application as a non-OSGI.
- The corresponding URL must be authorized and registered in the CMC..

Related Information

Deployment Scenarios [page 15]

7.6.3 Asynchronous Mode

The XMLHttpRequest specification has deprecated the requests to servers in synchronous mode.

In SAP BusinessObjects BI 4.2, it is recommended to use asynchronous calls when using the Web Intelligence JavaScript API in your extension code. In SAP BusinessObjects BI 4.3, these calls must be asynchronous. If you use synchronous calls in your call, you may receive `InvalidAccessError` exceptions when running the extension.

If you have previously written your extension using synchronous calls, your code is still valid and can be re-used. You simply need to use a callback function as additional argument when using any functions of the Web Intelligence JavaScript API returning data. This callback function will receive the response asynchronously. For example

Synchronous Mode (No more supported):

```
function getConnections()
{
    var res = WebiServices.getConnections();

    if (!res.error)
    {
       var cnx = res.connections.connection;
       if (!isArray(cnx))
          cnx = [ cnx ];

       var table = document.getElementById("connections-table");
       table.options.length = 0;
       for (var i = 0; i < cnx.length; i++)
       {
          var entry = cnx[i];
          table.options.add(new Option(entry.name, JSON.stringify(entry)));
       }
    }
    else
       alert(res.error.message);
 }
...
var res = WebiServices.document.addDataProvider(
 {
    dataprovider:
      {
        name: xlsEntry.name,
        dataSourceId: xlsEntry.id,
        properties: properties
      }
 });
```

Asynchronous Mode (Supported):

```
function getConnections()
 {
    WebiServices.getConnections( function(res) {
```

```
       if (!res.error)
       {
          var cnx = res.connections.connection;
          if (!isArray(cnx))
             cnx = [ cnx ];

          var table = document.getElementById("connections-table");
          table.options.length = 0;
          for (var i = 0; i < cnx.length; i++)
          {
             var entry = cnx[i];
             table.options.add(new Option(entry.name, JSON.stringify(entry)));
          }
       }
       else
          alert(res.error.message);
    });
 }
...
function createExcelDataProvider()
 {
    ...
    var arg =
    {
       dataprovider:
         {
           name: xlsEntry.name,
           dataSourceId: xlsEntry.id,
           properties: properties
         }
    };

    WebiServices.document.addDataProvider(arg, function(res) {
       if (res.success)
       {
           WebiServices.document.refreshDocument( function(response) {
             var docContext = WebiApplication.getContext();
             WebiApplication.loadReport({
                 reportId:docContext.selectedReportId},function()
                   {
                     alert("Report loaded!");
                   });
          });
        }
        else
           alert(res.error.message);
    });
}
```

:

7.6.4 Changes in Web Intelligence JavaScript API

In addition to the mandatory asynchronous calls, some changes have also been done in the Web Intelligence JavaScript functions compared to SAP BusinessObjects BI 4.2:

- It is no more possible to retrieve the token of the current user and reuse it.
- The `export` functions do not return an URL anymore but download the exported resource in a local folder.
  - `<static> exportDocumentAsZip`
  - `<static> exportReportAsZip`

    - `<static> exportReportElement`
    - `<static> exportReportElementAsImage`
- The GetContext function returns new parameters:
    - `isInstance`: Is true if the current document is an instance generated by a schedule
    - `path`: Contains the current document's path.

Related Information

https://help.sap.com/doc/2b701d4cd19b43289805d095e164b21a/4.3.2/en-US/index.html

7.7 About the Web Intelligence Extension Sample

The SAP BusinessObjects BI 4.3 server installation contains a WAR sample exposing a Web Intelligence extension and a Web Intelligence custom element. This WAR file is named webisample.war and is available in the Samples/webi/ folder.

It contains the source code for the extension and the custom element and can be used as a model to write your own extension.

To deploy and register this extension:

1. Deploy the WAR file in your application server. For Tomcat, simply copy it in the `webapps` folder.
2. Log into the CMC.
3. Add the application server URL to the list of Web Intelligence authorized URLs.
4. Register and enable the extension to a group of users by providing its URLs: `http://<host>:<port>/webisample/extension/SAP/ICExtension/declaration.json`
   Where `<host>` and `<port>` are the server and port of the machine running the application server.

The extension is now available for all the users of the selected group.

When these users use Web Intelligence, this extension is displayed as a sub-tab in the *Main* side panel > *Extensions* tab. It illustrates how to use the functions to get and set Web Intelligence input controls.

Related Information

Deploying the Extension [page 20]

8 Creating Web Intelligence Visualizations with a Custom Element Service

You can extend the list of visualizations supported by SAP BusinessObjects Web Intelligence through custom elements.

Custom elements are a type of visualizations whose rendering is delegated to a third-party service. You develop this service (or ask a service provider to develop it for you) by implementing REST API calls, in order to allow the third-party server to communicate with Web Intelligence servers.

A custom element service must provide useful information for Web Intelligence, such as the size of the custom element, how to display the data in the chart, and the rendering settings.

You can develop as many custom element services as you need. A service can also host several custom elements, hence several visualizations.

What You Need to Do

1. Develop or ask a service provider to develop the custom element service. The service must implement the REST APIs calls.
2. Deploy your service on your third-party server.
3. As an administrator, add the custom element service URL to the list of trusted URLs in the CMC. See .
4. As an administrator, connect to the CMC and add the name and URL of the custom element service you have deployed. Click *Test* to make sure the service respects the API contract.

> ⓘ Note
>
> This is the responsibility of the administrator to ensure that data exchange between the service and the BI platform does not affect the security of the production system.

Users can now use custom elements in their reports. For more information, see *Managing Web Intelligence settings* in the *Business Intelligence Platform Administrator Guide*.

How Custom Elements Display in Web Intelligence

Once a custom element service has been added to the Web Intelligence settings in the CMC, the visualizations provided by the service display as available charts in the Web Intelligence client interfaces. The end-user can select one of them to display their data as in any other charts supported by Web Intelligence.

A custom element is defined as any other report block by the following properties:

- Size, position, and borders
- Formulas on feeds

- Rendering settings

For more information, see *Working with charts in reports* in the *SAP BusinessObjects Web Intelligence User's Guide*.

> **Note**
>
> To export a custom element into a PDF document or a Microsoft Excel file, the service must return the custom element as an image. Otherwise, the custom element displays as blank.

> **→ Remember**
>
> Make sure your configuration allows access to the third-party server from the BI platform server. Otherwise, the custom element won't display. In the case of a Web Intelligence Rich Client not connected to the CMS repository, the custom element does not display either.

How the Service Works

When an administrator configures a custom element service

To confirm that the service is up and running and to select an output format for the custom element visualizations, the administrator tests the service in the Central Management Console. See Getting the Supported Formats [page 29] for more information.

When an end-user inserts a custom element visualization in a Web Intelligence document:

The name of the custom element service is the name chosen by the administrator when configuring this service. In the interface, the end-user can select a visualization in the list offered by the service. Since the 4.2 SP4 release, the service can also display a thumbnail of each visualization to better illustrate its content. See Getting the Visualization Types [page 30] and Getting the Visualization Icon [page 31] for more information.

The end-user can assign data to the visualization he has selected according to the feeding defined by the custom element service. See Getting the Getting the Visualization Feed Definitions [page 32] for more information.

When an end-user customizes the parameters of a custom element visualization (since 4.2 SP3):

The end-user can modify the settings values supported by the custom element service for the selected visualization. See Getting the Rendering Settings [page 37] for more information.

When a report that contains a custom element is displayed:

Based on end-user choices, the Web Intelligence server performs the calculation and sends the computed data and their metadata to the service. The service then generates an image or an HTML document as output of the request. Finally, the HTML output is added to an <iframe> tag and the image output to a standard tag to the report, which is displayed in the Web Intelligence interface.

When publishing or printing a Web Intelligence document displaying custom elements:

The Web Intelligence server requests an image output from the custom element service, independently of the output format selected by the administrator in the Central Management Console.

8.1 About the Custom Element Service APIs

You implement the custom element service APIs to display your visualization through a custom element in Web Intelligence.

Using these REST APIs, you allow Web Intelligence to:

- Get the formats supported by the service
- Get the types of visualization provided by the service
- Get the feeding definitions of a specific custom visualization
- Display the visualization properly
- Get the rendering settings of the visualization

Base URL

The base URL to use within the call requests must refer to the server where the service is deployed. For example:

```
http://myvisualizationserver.domain.com:8095
```

Response Status and Error Messages

HTTP Codes and Descriptions

Code	Description
200	Successful request
500	Failed request

If the request has failed, the call returns a response in JSON format as follows:

```
{
 "code":"XXX 00001",
 "message":"Invalid feeding."
}
```

Where:

- `code` specifies an error code for a type of issue. The preferred format consists of 3 letters indicating your service name and a 5-digit number separated by a space character.
- `message` is a descriptive error message.

Localization

Although the service must use the English locale by default for responses, it should also support other languages according to user preferences. To this end, you can use the `locale` query parameter that contains the country and optionally the language code in requests. For example:

```
api/visualizations?locale=en
api/vizualisations/vizID/feeds?locale=fr_FR
```

8.2 Getting the Supported Formats

Returns the list of media types supported by the custom element service.

The Web Intelligence server needs this information to know what media types the service supports.

It can be a selection of the following types:

- `text/html`
- `image/png`
- `image/jpg`
- `image/gif`
- `image/bmp`

The preferred media type is `text/html`, which allows interactivity in SAP BusinessObject Web Intelligence interfaces and a better user experience. The preferred image output is `image/png`.

The custom element service can return several media types. For example, it can return `text/html` to display the custom elements in a Web Intelligence report and `image/png` to export that report into a PDF document or a Microsoft Excel file.

Request

URI: `api/formats`

HTTP Method: *GET*

Request Parameters: None

Response

Format: *JSON*

Response Example:

```
{
```

```
 "formats": [
     "text/html",
     "image/png"
    ]
}
```

8.3 Getting the Visualization Types

Returns the list of visualizations supported by the Custom Element service.

The Web Intelligence server needs this information to know what visualizations the service supports.

Request

URI: `api/visualizations`

HTTP Method: *GET*

Request Parameter:

Parameter	Required	Data Type	Description	Parameter Type
`locale`	No	String	The user locale as a language and/or country string. For example: • `en` • `fr_FR`	Query parameter

Response

Format: *JSON*

Response Example:

GET `/api/visualizations?locale=en`

```
{
 "visualizations": [
   {
    "id": "bullet-chart",
    "name": "Bullet chart",
    "description": "This a bullet chart"
   },
   {
    "id": "funnel",
    "name": "Funnel chart",
    "description": "This is a funnel chart"
   }
  ]
```

```
}
```

Output	Description
`id`	The visualization identifier. Must be unique in the scope of the service.
`name`	The visualization name as it appears in Web Intelligence. Its translation can be returned, depending on the `locale` query parameter.
`description`	The visualization description as it appears in Web Intelligence. Its translation can be returned, depending on the `locale` query parameter.

8.4 Getting the Visualization Icon

Returns a picture that illustrates the visualization type identified by identifier.

You get the visualization identifier with the `api/visualizations` call.

Web Intelligence calls this API to display the visualization thumbnail in the dialog box listing all available charts. The picture's requested dimensions (width and height) are passed in the request. In this case, the requested dimensions are 50x50 pixels and the requested format is picture/png.

Request

URI: `api/visualizations/<vizID>/sample`

HTTP Method

URI: *POST*

Request Headers:

Header	Required	Value
Accept	Yes	The output format. For example, image/png
Content-type	Yes	application/json

Request Parameters:

Parameter	Required	Data Type	Description	Parameter Type
height	Yes	Numeric	The sample height	Request body
width	Yes	Numeric	The sample width	Request body

Request Example:

POST `/api/visualizations/funnel/sample?height=50&width=50`

Response

Format: a 50x50 pixels .png image representing the visualization sample.

8.5 Getting the Visualization Feed Definitions

Returns the list of data feeds of a visualization type specified by its identifier.

The Web Intelligence server needs this information to know the feed definition that a specific visualization type supports. You get the visualization identifier with the `api/visualizations` call.

Request

URI: `api/visualizations/<vizID>/feeds`

HTTP Method: *GET*

Request Parameters:

Parameter	Required	Data Type	Description	Parameter Type
`<vizID>`	Yes	String	The visualization type identifier	Path
`locale`	No	String	The user locale as a language and/or country string. For example: • `en` • `fr_FR`	Query

Response

Format: *JSON*

Response Example:

GET `/api/visualizations/funnel/feeds?locale=en`

```
{
 "feeds": [
    {
     "id": "category-axis",
```

```
      "name": "Category Axis",
      "description": "The primary chart axis",
      "axis": "0",
      "type": "dimension",
      "min": "1",
      "max": "-1"
     },
     {
      "id": "region-color",
      "name": "Color",
      "description": "This feeds is used to customize bar colors",
      "type": "dimension",
      "axis": "1",
      "min": "0",
      "max": "-1"
     },
     {
      "id": "primary-values",
      "name": "Values",
      "description": "Feeds for measures",
      "type": "measure",
      "min": "0",
      "max": "-1"
     }
    ]
}
```

Output	Description
`id`	The unique identifier in the scope of the current visualization type
`name`	The human-readable name of the feed. Can be localized using the `locale` query parameter
`description`	The human-readable description of the feed. Can be localized using the `locale` query parameter.
`type`	The type of data provided by the feed: `measure` or `dimension`, which includes the BI Semantic Layer types dimension, detail, hierarchy, and level
`axis`	The axis type: primary (`0`) or secondary (`1`). Mandatory for feeds of type `dimension`
`min`	The minimum number of formulas required by the feed
`max`	The maximum number of formulas that the feed can manage. `max=-1` means there is no limit to the number of formulas.

8.6 Rendering the Visualization

Renders the visualization in the specified format.

The request contains the following inputs:

- The output format details (height, width, and dpi)
- The default font and palette settings to use in the visualization
- The end-user settings
- The Web Intelligence expressions applied by the end-user to each feed

- The metadata of the visualization and the dataset

You get the end-user settings from the call Getting the Rendering Settings [page 37]. Rendering setting values are saved in the Web Intelligence document.

Request

URI: `api/visualizations/<vizID>/render`

HTTP Method: *POST*

Request Headers:

Header	Required	Value
`Accept`	Yes	The output format. For example: `image/png`
`Content-Type`	Yes	`application/json`

Request Parameters:

Parameter	Required	Data Type	Description	Parameter Type
`<vizID>`	Yes	String	The visualization type identifier	Path
`locale`	No	String	The user locale as a language and/or country string. For example: • `en` • `fr_FR`	Query
`width`	Yes	Numeric	The visualization width	Request body
`height`	Yes	Numeric	The visualization height	Request body
`dpi`	Yes	Numeric	The visualization dpi	Request body
`font`	Yes	JSON object	The default font to use in the visualization. Must be installed on the third-party server.	Request body
`feed`	Yes	JSON array	The list of Web Intelligence formulas applied by the end-user to each feed. Each feed contains an array of expressions. Each expression contains a reference to an object of the `data` parameter based on its identifier.	Request body

Parameter	Required	Data Type	Description	Parameter Type
`data`	Yes	JSON array	The metadata of the visualization. Each `data` entry contains the following: • A unique identifier • An optional title • `dataType` that can be `string`, `double` or `date` • `dataStructure` that can be `tree` if it is based on hierarchical data or `simple` if it is flat data • The data itself • `cardinality` that specifies if the data is contained in an array (`1`) or a matrix (`2`).	Request body
`instances`	Yes	JSON object	The custom element source, specified by the following: • The Web Intelligence document CUID • The report identifier • The block identifier • The instance identifier, which can be used to differentiate two documents with the same CUID • The temporary reference to the report element instance (not valid after a refresh) • The unique reference identifier that can be used even after a refresh	Request body
`settings`	Yes	JSON array	The values defined by the end-user in the *Format Custom Element* dialog box in Web Intelligence. It contains only non-default values. You get these values from the call Getting the Rendering Settings [page 37]. The property identifier may contain a slash (`/`) if the property has subproperties.	Request body
`customProperties`	Yes	JSON array	The custom properties defined on the custom element.	Request body
`palette`	Yes	JSON array	The list of default hexadecimal colors to use in the visualization.	Request body

Request Example:

POST `/api/visualizations/funnel/render?locale=en_US`

```
{
 "width":400.0,
 "height":300.0,
 "dpi":96,
 "font":{
    "name":"Arial",
    "size":9,
    "color":"#333333",
    "isBold":false,
    "isItalic":false
    },
 "feeding":[
    {
    "id":"category-axis",
    "expressions":[
        {
         "dataId":3
        }
     ]
    },
    {
    "id":"region-color"
    },
    {
    "id":"primary-values",
    "expressions":[
        {
         "dataId":1
        }
     ]
    }
  ],
 "data":[
    {
    "id":"3",
    "title":"Year",
    "values":{
        "dataType":"string",
        "rawvalues":[
            "2004",
            "2005",
            "2006"
           ],
        "dataStructure":"simple",
        "cardinality":1
    },
    "type":"dimension"
    },
    {
    "id":"1",
    "title":"Sales revenue",
    "values":{
        "dataType":"double",
        "rawvalues":[
            8095814,
            1.3232246E7,
            1.50591428E7
           ],
        "dataStructure":"simple",
        "cardinality":1
    },
    "type":"measure"
    }
 ],
```

```
 "instance":{
    "documentId":"AVyTZhNVc9dHlaVGjuyOoqw",
    "instanceId":"7130cd02788a5cb8b1a44f409bfed81d58eb2c0c",
    "reportId":"4",
    "blockId":"5",
    "iref":"5.c",
    "uiref":"UIREF:RID=5:BID=38"
   },
 "settings":[
    {
     "property":"location",
     "region":"legend",
     "value":"top"
    }
 ],
 "customProperties":[
    {
     "property":"Custom_Prop2",
     "value":"{\"value1\":\"myvalue\",
\"value2\":\"myCustomJSONDefinedSetting\"}"
    },
    {
     "property":"Custom_Prop1",
     "value":"myvalue"
    }
 ],
 "palette":[

"#008fd3ff","#99d101ff","#f39b02ff","#9fcfecff","#4ba707ff","#f6d133ff","#cb4d2cf
f","#cac7baff
   ]
}
```

Response

Format: a stream containing the response.

- If the output format is `text/html`, it is an HTML document that will be loaded to the dedicated Iframe.
- If the output format is an image, it is the resulting image.

8.7 Getting the Rendering Settings

Returns the list of end-user settings supported by a type of visualization.

Web Intelligence needs this information to display the rendering settings returned by the service. You send the values actually set by the end-user by calling the service for rendering. Use these settings in the call Rendering the Visualization [page 33].

Settings are organized by regions such as title, legend, and axis. Each region contains several groups, and each group may contain several categories. Finally, a category contains a list of properties.

Settings are displayed in the *Format Custom Element* dialog box of Web Intelligence. Each region, group, and category of a setting is represented respectively as a left-pane button, section, and area of the dialog box. Web Intelligence assigns the appropriate widget to the setting according to its type (boolean, string, integer, and so on). A setting is displayed if at least one category is defined.

Request

URI: `api/visualizations/<vizType>/settings`

HTTP Method: *GET*

Request Headers:

Header	Required	Values
`Content-Type`	Yes	`application/json`

Request Parameters:

Parameter	Required	Data Type	Description	Parameter Type
`<vizType>`	Yes	String	The visualization type	Path
`locale`	No	String	The user locale as a language and/or country string. For example: • `en` • `fr_FR`	Query

Response

Format: *JSON*

Response Example:

GET `api/visualizations/funnel/settings?locale=en`

Rendering settings are the following:

- Region is Legend.
- Group is Design.
- Design categories are General and Layout.
- General properties are isVisible, font and background color. The font property has 5 subproperties (name, size, bold, italics, and underline).
- Layout properties are location and spacing. Location has only 4 possible values. Spacing has a set of parameters that restrict its possible values.

```
{
 "regions":[
    {
     "id":"legend",
     "name":"Legend",
     "groups":[
```

```
     {
      "name":"Design",
      "categories":[
      {
       "name":"General",
       "description":"General design properties",
       "properties":[
          {
           "id":"isVisible",
           "name":"Visible ?",
           "description":"Indicates if title of the visualization is displayed
or not",
           "type":"boolean",
           "default":"true"
          },
          {
           "id":"font",
           "name":"Font",
           "description":"Font of the legend.",
           "type":"font",
           "default":"",
           "properties":[
              {
               "id":"name",
               "name":"Font name",
               "description":"Font name",
               "type":"string",
               "default":"arial"
              },
              {
               "id":"size",
               "name":"Font size",
               "description":"Font size",
               "type":"integer",
               "default":"8"
              },
              {
               "id":"bold",
               "name":"Is bold?",
               "description":"Is bold?",
               "type":"boolean",
               "default":"false"
              },
              {
               "id":"italic",
               "name":"Is italic?",
               "description":"Is italic?",
               "type":"boolean",
               "default":"false"
              },
              {
               "id":"underline",
               "name":"Is underline?",
               "description":"Is underline?",
               "type":"boolean",
               "default":"false"
              },
              {
               "id":"color",
               "name":"Font color",
               "description":"Font color",
               "type":"color",
               "default":"#ffffffff"
              }
            ]
          },
          {
           "id":"backgroundColor",
```

```
            "name":"Background color",
            "description":"Background color",
            "type":"color",
            "default":"#d0d0d0cc"
          }
       ]
      },
      {
       "name":"Layout",
       "description":"Layout properties",
       "properties":[
          {
           "id":"location",
           "name":"Location",
           "description":"Indicates where the title should be displayed",
           "type":"state",
           "default":"right",
           "choices":[
              {
               "id":"top",
               "name":"Top",
               "description":"Display title at the top."
              },
              {
               "id":"left",
               "name":"Left",
               "description":"Display title on the left."
              },
              {
               "id":"bottom",
               "name":"Bottom",
               "description":"Display title at the bottom."
              },
              {
               "id":"right",
               "name":"Right",
               "description":"Display title on the right."
              }
           ]
          },
          {
           "id":"spacing",
           "name":"Spacing",
           "description":"Indicates space between items",
           "type":"integer",
           "default":"4",
           "parameters":{
              "min":"2",
              "max":"6",
              "step":"1"
           }
          }
       ]
      }
     ]
    }
   ]
  }
 ]
}
```

Regions

Output	Output Type	Description
`id`	String	The region identifier. Must be unique in the scope of the service.

Output	Output Type	Description
`name`	String	The region name. Is localized.
`groups`	JSON array	The groups of a region

Groups

Output	Output Type	Description
`name`	String	The group name. Is localized.
`categories`	JSON array	The categories of a group

Categories

Output	Output Type	Description
`name`	String	The category name. Is localized.
`description`	String	The category description. Is localized.
`properties`	JSON array	The properties of a category

Properties

Output	Output Type	Description
`id`	String	The property identifier. Is unique in the region and must contain only characters from [a-z], [A-Z], [0-9], and [. | - | _].
`name`	String	The property name. Is localized.
`description`	String	The property description. Is localized.
`type`	Enum	The property type. Is one of the following: • `boolean` • `color` • `double` • `font` • `integer` • `state` • `string`
`default`	String	The property default value. In the case of a color, default is a hexadecimal string. In the case of a font, default is an empty string.

Output	Output Type	Description
`parameters`	JSON object	The property parameters if any, for `int` and `double` type properties only. Specifies constraints to the property value. If a property describes a size, the parameter `isUnit` is added and set to `true`. Hence, all parameters are expressed in metrics so that Web Intelligence can convert them into the end-user unit defined by its preferred viewing locale.
`properties`	JSON array	The subproperties of a property if any. The property itself has no value in this case. For example, the property of type `font` has a set of subproperties.

Properties: state

The values of a property of type `state` are restricted to a set of specific values. The default value must be one of the identifiers of these values. A property of type `state` cannot have parameters.

Output	Output Type	Description
`choices`	JSON array	The list of the possible values of the property
`id`	String	The value identifier. Is unique in the property
`name`	String	The value itself. Is localized.
`description`	String	The value description. Is localized.

9 Adding External Functions to Web Intelligence Formula Language

The Web Intelligence formula language offers a broad range of functions and operators to leverage datasets retrieved from your data sources, including string manipulation, mathematical operations, Document and Data Sources properties. You can find these functions listed in the Functions section.

If the functions you require are not available or if you wish to optimize the processing of a formula by rewriting it, you can extend the formula language using external functions.

External functions are custom Web Intelligence functions that enhance the list of functions available in Web Intelligence formula language. External functions are visible and usable like the other Web Intelligence standard functions in variable definition or any formula.

To provide an external function, you must create a C++ external library that implements your own logic and deploy it on the machine where Web Intelligence is running. An external library can expose more than one external function and you can provide several external libraries.

9.1 To create an external function for Web Intelligence

1. Declare the external function in an XML file describing all external functions exposed by a library.
2. Declare your library in the XML catalog file that lists all external libraries.
3. Implement the library in C++ using a specific API for external function.
4. Compile the C++ source file. The file extensions of the generated library depend on the operating system:
   - DLL for Windows
   - SO for Linux
5. Copy the generated library and the XML files into the dedicated `WebiCalcPlugin` folder on any the machine (server or desktop) running Web Intelligence Rich Client. For exemple:
   - On Windows: `[INSTALLDIR]/SAP BusinessObjects Enterprise XI 4.0/win64_x86/WebiCalcPlugIn`
   - On Linux: `[INSTALLDIR]/enterprise_xi40/linux_64/WebiCalcPlugIn`
6. Restart the system to automatically add the external function to the list of Web Intelligence functions. This function will be available in **Formula Editor** and contextual help. The external function is based on a unique identifier. When it is used in a report, it cannot be misinterpreted even if you use several external libraries.

> → Tip
>
> We recommend to re-compile any library compiled with a release prior to SAP BusinessObjects BI 4.3 Support Package 2.

Related Information

External function folder [page 44]
Defining the XML function declaration [page 44]
Defining the XML catalog declaration [page 47]
Writing the C++ source files [page 48]
Compiling the C++ source files [page 49]
Deploying the external library [page 50]

9.2 External function folder

All files related to Web Intelligence external functions must be copied in the `[INSTALLDIR]\[VERSION]\[OS]_[PLATFORM]\WebiCalcPlugIn` folder.

- On any server of the SAP BusinessObjects BI cluster running Web Intelligence.
- On each desktop where Web Intelligence Rich Client is installed.

Where:

- `[VERSION]` is the version of the product, for example `BusinessObjects Enterprise XI 4.0`
- `[OS]` is the operating system, for example `win64` for Windows Operating System or `linux` for Linux Operating System.
- `[PLATFORM]` is the platform, for example `x86` on an Intel CPU.

This folder must contain the following files:

- One or more libraries containing the implementation of external functions.
- For each library, an XML file declaring the list of external functions exposed by this library.
- A single `externalcatalogs.xml` XML file containing the list of all XML function definition files.

Related Information

Defining the XML function declaration [page 44]
Defining the XML catalog declaration [page 47]

9.3 Defining the XML function declaration

The implementation of external functions is delivered through a C++ library, which may include multiple functions. For each library, an XML file must be provided to specify the functions exposed by that library.

The structure of this XML file is as follows:

```
<CATALOG>
  <LIBRARY file="<FileName>">
   <FUNCTION guid="<FunctionGUID>" name="<FunctionName>">
     <ARGLIST>
       <ARG type="<ArgumentType>" name="<ArgumentName>"/>
…
    </ARGLIST>
    <RETURN type="<ReturnType>"/>
    <CATEGORY type="<CategoryType>"/>
    <HINT value="<Hint>"/>
    </FUNCTION>
…
</LIBRARY>
</CATALOG>
```

This XML tags and attributes are described in the following table:

Tag	XML attribute	XML definition object
`<CATALOG>`	N/A	The XML root.
`<LIBRARY>`	`<file>`	The name of the library file that contains the function implementation code. The library extension should not be specified in this name.
`<FUNCTION>`	`<guid>`	The GUID that uniquely identifies the function. This GUID must be unique so the function cannot be reused or confused with other custom libraries. **→ Tip** Define all GUIDs in advance and make sure that all GUIDs are unique from a global point of view. For Windows you can use the GUID tool provided with Visual Studio or download it from the Microsoft website. For Linux, the tool `/usr/bin/uuidgen` can be found in the `libuuid1` (Debian) package.
	`<name>`	The function name that appears in the **Formula Editor**. The function name must: • be a simple, unique name for the function • start with a letter • use lower and upper case letters, number characters, or the _ character • not already exist in the Web Intelligence library **ⓘ Note** The name will not be translated to another language.

Tag	XML attribute	XML definition object
<ARGLIST>		The list of parameters supported by the function. The ARGLIST tag may contain ARG tag, one for each parameter. The number of parameters should be lower than or equal to five.
<ARG>	<type>	The parameter types. Only single value parameters are supported. Table parameters for instance aren't supported. The possible parameter types are: • Numeric • Boolean • Date • String
	<name>	The name of each parameter as it should appear in the *Formula Editor*. The name shows the prototype of the method to the user. Use only alphanumeric characters.
<RETURN>	<type>	The type of the values returned by the function. The available types are: • Numeric • Boolean • Date • String
<CATEGORY>	<type>	The category in which the function will appear in the *Formula Editor* . The available categories are: • Character • Date • Document • DP • Misc • Logical • Num For consistency, add the functions in the most logical category. For example, add functions related to strings manipulation in the Character category.
<HINT>	<value>	The hint that appears in the *Formula Editor*. The hint explains the use of the function.

Example

SampleMath.xml

Sample Code

```
<CATALOG>
<LIBRARY file="SampleMath">
<FUNCTION guid="CC3E9742-67A7-4844-9DBF-2CCD4F6ECABE" name="MySquareFct">
<ARGLIST>
<ARG type="Numeric" name="input_number"/>
</ARGLIST>
<RETURN type="Numeric"/>
<CATEGORY type="Num"/>
<HINT value="My square function."/>
</FUNCTION>
</LIBRARY>
</CATALOG>
```

Related Information

Defining the XML catalog declaration [page 47]

9.4 Defining the XML catalog declaration

All XML files declaration must be declared into one single XML catalog declaration file named `externalcatalogs.xml`.

If it does not exist, you may create it. Otherwise, add any the new file declaration into this file.

The structure of this XML file is:

Sample Code

```
<CATALOGS>
              <CATALOG file="<fileName>>
              ...
</CATALOGS>
```

Example

externalcatalogs.xml

Sample Code

```
<CATALOGS>
                <CATALOG file="SampleMath.xml"/>
</CATALOGS>
```

Related Information

Defining the XML function declaration [page 44]

9.5 Writing the C++ source files

The external function algorithm must be written in C++ source files. This implementation must be consistent with its XML function declaration.

1. In the file, add the `ibovariant.h` header.
2. For each method, start the declaration with the `BO_DECLARE_USER_FCT` macro.

   The macro includes:

   - the function name as it appears in the XML functions declaration file.
   - the return value object name
   - the parameter object name

   > **Note**
   >
   > The function returns a `BONOERROR` if everything is okay, otherwise the `#EXTERNAL` error message appears into the report.

Example: Square.cpp

```
// Headers file include of the WebI hearders
#include <ibovariant.h>
// To not repeat BOExtFunct::
using namespace BOExtFunct;
BO_DECLARE_USER_FCT (// Name of function as it was defined in the XML.
                         MySquareFct,
                     // Name of the return value object.
                     retVal,
                     // Name of the parameters object.
                     parameters
                     )
{
        try // Always used a try{}catch(…) to be sure no
            // exception was thrown outside this Web
            // Intelligence user function.
        {
           // Get the first parameter.
          const iBOValue&param0 = parameters[0];
           // Transform the parameter to the correct type.
          double valPar0(param0);
           // Assign value to the return value.
          retVal = valPar0 * valPar0;
        }
        catch(...)
        {
          return BOERROR; // Unkonwn exception so notify WebI
```

```
        }
        return BONOERROR; // It's OK
}
```

Related Information

Defining the XML function declaration [page 44]

9.6 Compiling the C++ source files

1. To create a project, go to *File* > *New* > *Project*.
2. In *Project types*, select *Visual C++* > *General*.
3. In *Templates*, select *Empty Project*.
4. Specify the name of the project.
5. Specify the destination folder for the project.
6. Click *OK*.
7. Right-click the project and select *Properties*.
8. In *Configuration*, select *All configurations*.
9. In *Configuration Properties* > *General* set *Configuration Type* to *Dynamic Library (.dll)*.
10. Click *OK*.
11. Right-click the project and select *Add* > *New Item*.
12. In *Category*, select *Code*.
13. In *Template*, select *C++ File (.CPP)* .
14. Specify the name of the CPP file.
15. Click *Add*.
16. Right-click the project and select *Properties*.
17. In *Configuration*, select *All configurations*.
18. In *Configuration Properties* > *C/C++* > *Additional Include Directories*, add the folder which contains the Business Objects file headers.
19. Click *Apply*.
20. In *Configuration*, select *Debug*.
21. In *Configuration Properties* > *C/C++* > *Code generation*, set *Runtime Library* to *Multi-threaded DLL (/MD)*.

> **Note**
>
> If you are running a machine where Microsoft Visual Studio is installed, you can use *Multi-threaded Debug DLL (/MDd)* instead of *Multi-threaded DLL (/MD)* to benefit from its debug environment.

22. Click *Apply*.
23. In *Configuration*, select *Release*.
24. In *Configuration Properties* > *C/C++* > *Code generation* set *Runtime Library* to *Multi-threaded DLL (/MD)* .

> **Note**
>
> If you are running a machine where Microsoft Visual Studio is installed, you can use *Multi-threaded Debug DLL (/MDd)* instead of *Multi-threaded DLL (/MD)* to benefit from its debug environment.

25. Click *OK*.
26. Add the code to the CPP file.
27. Compile.

9.7 Deploying the external library

The SAP BusinessObjects BI administrator must copy the XML files and related library files in the `WebiCalcPlugIn` folder on:

- Any server of the SAP BusinessObjects BI cluster running Web Intelligence.
- Each desktop where Web Intelligence Rich Client is installed.

> **Caution**
>
> - Replacing or adding a library in this folder can represent a threat to the system. Since the library is automatically loaded, an external library can access internal critical data or processes, putting the system in danger.
> - Make sure that the site administrator implements the appropriate security access to the related folder, so that only authorized people access it.

On a SAP BusinessObjects BI server, restart the Web Intelligence servers (WIProcessingServer) to automatically add the external functions to the list of the functions available in Web Intelligence (Formula Editor, contextual help...) On a desktop, restart Web Intelligence Rich Client.

Related Information

External function folder [page 44]

9.8 Error messages

When using the external function in Web Intelligence, a formula may return the #EXTERNAL error message and display it in the document. This error may be caused by the following problems:

- A formula refers to an external function for which no external library is available.
- The system cannot load a library or is missing information for an external function.
- The system has an inconsistent XML declaration, missing library, or duplicated function.
- An external function does not initialize the return value.
- An external function initializes the return type with bad type. For example, a double was set to a string.
- An external function returns an error code

Request your SAP BusinessObjects administrator to recompile and redeploy the library responsible for this function, ensuring they check for any errors during the process.

> → Tip
>
> We recommend to re-compile any library compiled with a release prior to SAP BusinessObjects BI 4.3 Support Package 2.

9.9 Trace log message errors

If an error appears during XML parsing/validation, a message appears to the user and errors are created in the trace logs.

Log type	Error messages
XML logs	File cannot be read or is missing. Bad XML structure due to: • Parent/Children relation invalid. • Missing field (ID function, name function). • Invalid field value.
DLL logs	File is missing. DLL cannot be loaded. Function is not found in the DLL.

Log type	Error messages
Function logs	Function name is already in use.
	Function ID is already used.
	Function name is missing.
	Return type is invalid.
	ID is invalid.
	Number of parameters is invalid.
Parameters logs	Parameter name is missing.
	Parameter type is invalid.
Runtime logs	The user function does not initialize the return value.
	The user function initializes the return value with a bad type.
	The user function returns the BOERROR error code.

9.10 Using the Web Intelligence samples files

Make sure to have the following applications installed:

- SAP BusinessObjects BI 4.3, including Web Intelligence
- Visual Studio C++ VS2015 or higher

An external function sample is provided when you install SAP BusinessObjects on Windows. It contains the files to generate `SampleString.DLL` and `SampleMath.DLL` libraries exposing external functions:

- The `MySquareFct.XML` function declaration that defines the `MySquareFct` external function.
- The `externalcatalogs.xml` catalog declaration.
- The `OpenSolution.bat` script to set the `<WEBICALCPLUGINAPI>` variable with the folder containing the C++ headers before opening Visual Studio.

You can use them as a model to write new external functions.

1. Retrieve the `Samples.zip` file located in the `[INSTALLDIR]\userlibs\WebI\Samples\` folder.
2. Unzip this `Samples.zip` file.
3. To open the samples in Visual Studio, launch the `OpenSolution.bat` script.

   The `OpenSolution.bat` can only work on a machine where Web Intelligence server or Rich Client is installed. If this is not the case, you can:

   1. From a Web Intelligence installation, copy the headers files `boplatform.h` and `ibovariant.h` located in the `[INSTALLDIR]\userlibs\WebI\API` folder.
   2. Paste them in a folder on the machine where you run Visual Studio.
   3. Set the `<WEBICALCPLUGINAPI>` variable to this folder.

4. Start Visual Studio.

10 Exposing Web Intelligence Features with REST Web Services

The Web Intelligence RESTful Web Service SDK provides a series of REST APIs that allows you to expose the Web Intelligence functionalities into your analytics applications.

Since the SDK is provided with the BI platform, you have nothing to install on the developer machine or where your application is deployed. The major benefit of the SDK is that you can use the REST APIs with any programming language that support the HTTP protocol, so that end-users can access the broad range of Web Intelligence features in many ways. A web service performs CRUD (Create, Read, Update, Delete) operations on data over HTTP, sending requests and receiving responses either in XML or JSON format. The way you implement these services is at your convenience. For example, you can automate batch operations on Web Intelligence documents. You can also make documents and reports available into non-SAP web applications.

The REST APIs expose features that relate to all Web Intelligence functional domains:

- Creating documents and building queries
- Creating reports with tables, sections, and charts
- Refreshing documents to get data
- Formatting reports
- Saving and exporting documents and reports
- Scheduling documents

Some Java samples are also provided to help you understand the REST APIs. They are supplied in the archive `<bip-install-dir>\Samples\webi\RaylightRESTWS_Samples.zip`.

> **Note**
>
> Before using the APIs, you need to logon to the BI platform and access the document or universe folder via the BI platform RESTful Web Service SDK.

Related Documentation	Description
See the *SAP BusinessObjects RESTful Web Service SDK User Guide for Web Intelligence and the BI Semantic Layer* on the SAP Help Portal.	The official guide for developing with the Web Intelligence RESTful Web Service SDK
See the *Business Intelligence platform RESTful Web Service Developer Guide* on the SAP Help Portal.	The official guide for developing with the BI platform RESTful Web Service SDK

> **Note**
>
> Access to these guides is restricted to customers with a logon to the SAP Support Portal.

11 Consuming BI Semantic Layer Universes with REST Web Services

The BI Semantic Layer RESTful Web Service SDK provides a series of REST APIs that allow you to access relational universes, browse universe metadata, create and execute queries. It supports UNV universes created with the universe design tool as well as UNX universes created with the information design tool.

Since the SDK is provided with the BI platform, you have nothing to install on the developer machine or where your application is deployed. The major benefit of the SDK is that you can use the REST APIs with any programming language that support the HTTP protocol. For example, you can rewrite a query panel using JavaScript. A web service performs CRUD (Create, Read, Update, Delete) operations on data over HTTP, sending requests and receiving responses either in XML or JSON format. The way you implement these services is at your convenience. Result sets are returned using the OData protocol.

The REST APIs expose features that relate to the main functional domains of the SAP BusinessObjects reporting tools:

- Browsing universes and retrieving metadata
- Building queries on universes and retrieving data

Some Java samples are provided to help you understand the REST APIs. They are supplied in the archive `<bip-install-dir>\SL SDK\SDK Samples\SLRESTWebService.zip`.

> **Note**
>
> Before using the APIs, you need to logon to the BI platform and access the universe folder via the BI platform RESTful Web Service SDK.

Related Documentation	Description
See the *SAP BusinessObjects RESTful Web Service SDK User Guide for Web Intelligence and the BI Semantic Layer* on the SAP Help Portal.	The official guide for developing with the BI Semantic Layer RESTful Web Service SDK
See the *Business Intelligence platform RESTful Web Service Developer Guide* on the SAP Help Portal.	The official guide for developing with the BI platform RESTful Web Service SDK

> **Note**
>
> Access to these guides is restricted to customers with a logon to the SAP Support Portal.

12 Developing Applications to Design and Administrate Universes

The BI Semantic Layer Java SDK allows you to access the features of the information design tool within a program of your own. You can develop Java applications to design the UNX universe resources (data foundations, business layers, and connections), to publish them in a CMS repository, and to configure security settings on published universes.

Some samples are also provided to help you understand the Java SDK APIs. They are supplied in the archive `<bip-install-dir>\SL SDK\SDK Samples\com.sap.sl.sdk.authoring.samples.source.jar`. For instructions on how to use the samples, see the BI Semantic Layer Java SDK Developer Guide.

Similarly, the Universe Design Tool COM SDK gives you access to the features of the universe design tool. You can develop applications to design and manage UNV universes using the provided COM objects.

13 Creating a Data Access Driver with JavaBean

A data access driver is a software component that runs with the data access service of the BI platform called Connection Server to perform requests to data sources and retrieve data for UNV and UNX universes. SAP BusinessObjects applications use a wide range of data access drivers to communicate with database middleware. In addition to the supplied data access drivers, you can create your own JavaBean drivers and use them as data access drivers for your data sources. Refer to the Data Access Guide for more information.

Writing a JavaBean

The JavaBean driver requires to write a JavaBean (ie a Java class) where some methods defined by the developer are exposed as stored procedures. To the data driver's consumer, it looks like a database that exposes only stored procedures.

For more details about how to use the JavaBean, refer to Section 5.5 of the Data Access Guide.

In functional terms the output of the driver is a set of independent tables (the results of the stored procedures), which are not manipulated using SQL. For the developer, SQL support does not need to be implemented, so such drivers are much easier and faster to write.

This JavaBean class must:

- Have a constructor that takes no parameters
- Have an initialize method, that takes some parameters to configure and initiate the connection to the data source
- Implement methods that are exposed as stored procedure if:
  - They are public methods
  - They return a java.sql.ResultSet
  - They have only scalar datatypes parameters: String, integer, float, double, ...

You can create several JavaBean drivers, but you need to write one class for each connection.

Procedure

1. Include ConnectionServer.jar in your project in order to use the class Context. This JAR can be found in the folder:
   `<INSTALL DIR>\SAP BusinessObjects\SAP BusinessObjects Enterprise XI 4.0\java\lib\`
2. Write a Java class constructor that does not need any argument. This constructor will give the name of your JavaBean. For example:

Sample Code

```
public class MyBean {
        public MyBean ()
   }
```

3. Implement the initialize method that connects to the data source.

Sample Code

```
public void initialize(Context context, String initString, Properties
properties) {
   // Add your own code here
}
```

This method must take the following arguments:

- Context: That contains the Connection Server configuration
- String: The string that the user can pass to the wizard
- Properties: Contains parameters like the authentication mode or the user's credentials

4. Implement the methods that retrieve and return dataset and are exposed as stored procedures. These methods must be public, returns a ResultSet and have non-scalar arguments.

Sample Code

```
public ResultSet myStoredProcMethod ( String stringArgument ) {
   // Add your own code here
}
```

5. Build the JAR file(s) for your JavaBean. Make sure to include potential dependencies.

To use this driver, in all server and client machines that need to access this data source:

- Create a dedicated folder in the `<INSTALL DIR>\SAP BusinessObjects\SAP BusinessObjects Enterprise XI 4.0\dataAccess\connectionServer\javabean` folder.
- Copy the JAR file(s) in this folder.

JavaBean Sample

A sample JavaBean is delivered with your SAP BI installation and can be used to query Google Cloud database. You need to generate a unique key from your Google account and use this key as the URL for authentication.

This JavaBean is installed when you install the BI platform. It is located in the

`<INSTALL DIR>\SAP BusinessObjects\SAP BusinessObjects Enterprise XI 4.0\dataAccess\connectionServer\javabean\BigQuery` folder.

Its code is located in the folder:

`<INSTALL DIR>\SAP BusinessObjects\SAP BusinessObjects Enterprise XI 4.0\dataAccess\connectionServer\DDK\examples\BigQueryJavabean\BigQuerySample`

You can use this code to create your own JavaBean, by following the indentation and comments provided in this sample.

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering an SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

www.sap.com/contactsap

THE BEST RUN SAP