



# Custom Data Provider Plug-in Developer Guide

## Copyright

© 2009 SAP® BusinessObjects™. All rights reserved. SAP BusinessObjects and its logos, BusinessObjects, Crystal Reports®, SAP BusinessObjects Rapid Mart™, SAP BusinessObjects Data Insight™, SAP BusinessObjects Desktop Intelligence™, SAP BusinessObjects Rapid Marts®, SAP BusinessObjects Watchlist Security™, SAP BusinessObjects Web Intelligence®, and Xcelsius® are trademarks or registered trademarks of Business Objects, an SAP company and/or affiliated companies in the United States and/or other countries. SAP® is a registered trademark of SAP AG in Germany and/or other countries. All other names mentioned herein may be trademarks of their respective owners.

2009-06-16



# Contents

<b>Chapter 1</b>	<b>Preface</b>	<b>5</b>
	About this Document.....	6
	Who Should Read this Document?.....	6
<b>Chapter 2</b>	<b>Introduction</b>	<b>7</b>
	Introduction to the Custom Data Source Framework.....	8
<b>Chapter 3</b>	<b>System Workflow</b>	<b>11</b>
	Workflow of the System.....	12
	Data Source Creation Phase.....	14
	Data Provider Phase.....	17
<b>Chapter 4</b>	<b>Getting Started</b>	<b>21</b>
	Getting Started with Plug-in Development.....	22
	CustomDSExtension Interface.....	23
	CustomDSComponent Interface.....	26
	CustomDataSource Interface.....	26
	CustomDataProvider Interface.....	31
	Extension Types.....	33
<b>Chapter 5</b>	<b>Configuring and Deploying the Plug-in</b>	<b>35</b>
	Overview.....	36
	Configuring the Plug-in.....	36
	Deploying the Plug-in.....	39
	Deploying Samples.....	39

## Contents

<b>Chapter 6</b>	<b>Points to Consider while Developing and Deploying the Plug-in</b>	<b>43</b>
<b>Chapter 7</b>	<b>Utility Class</b>	<b>47</b>
	Limitations of the Utility Class.....	48
<b>Chapter 8</b>	<b>Limitations of the CDS Framework</b>	<b>51</b>
<b>Chapter 9</b>	<b>Best Practices</b>	<b>53</b>
<b>Chapter 10</b>	<b>Abbreviations</b>	<b>55</b>
<b>Appendix A</b>	<b>More Information</b>	<b>57</b>

Preface

1

chapter

## About this Document

The *Custom Data Provider Plug-in Developer Guide* is intended to help plug-in developers gain a better understanding of Custom Data Source Framework. It describes how to create a Web Intelligence document (.wid) using a Custom Data Source by developing a plug-in and using it with Web Intelligence Rich Client (WRC). This document also describes how to develop, configure, and deploy a plug-in for the Custom Data Source (CDS) Framework. The CDS Framework plug-in uses a Custom Data Source, which can be any source of data besides the data sources supported by Business Objects universes. The CDS Framework provides the necessary information to facilitate the creation of a .wid document.

Before BusinessObjects XI R3, users were allowed to create Web Intelligence documents on universes. In BusinessObjects XI R3, users can create Web Intelligence documents based on a custom data source, such as a text file or an Excel file. Starting with this release, users can create a .wid document through WRC by using any custom data source as input with the help of a plug-in. This is made possible by the CDS Framework extension points.

## Who Should Read this Document?

This document is intended for developers who want to develop, configure, and deploy the CDS Framework plug-in.

Introduction

2

chapter

# Introduction to the Custom Data Source Framework

WebIntelligence Custom Data Source (CDS) Framework is a plug-in framework that facilitates creation of a data provider on custom data sources that could be consumed by Web Intelligence Rich Client. Both the data source information and data provider behavior can be pluggable by implementing the core interfaces of the framework. This will enable creation of Web Intelligence data providers on the sources that is not directly implemented or provided by Web Intelligence Rich Client.

The CDS Framework provides the necessary infrastructure to use information from custom data sources, such as local files, URI locations, and in-memory data structures, and enables the creation of Web Intelligence documents.

The CDS Framework supports both incremental and non-incremental access of data sources. Incremental access of data source refers to the following: a plug-in may require several sets of input from the user to access a custom data source. Each time the plug-in provides only the new/updated information to the CDS Framework, assuming that the Framework has stored the information that was provided earlier. This iteration may continue till the plug-in obtains the required information from the user to access the data source.

For more information on implementing incremental and non-incremental access of data sources, see [Deploying Samples](#) on page 39. The CDS Framework also enables each plug-in to operate in a sandboxed environment of its own.

Before creating a Web Intelligence document by using a custom data source, users must complete the following tasks:

1. Develop a data source specific plug-in.
2. Configure the plug-in and then deploy it on the BusinessObjects installation.

After completing the above-mentioned tasks, users can create a .wid document by using custom data source supported by the newly developed plug-in.

In a typical scenario, users are familiar with the data sources in their environment. However, they may encounter issues while using the Web Intelligence application to create a .wid document. The CDS Framework is

specifically designed to provide an easy-to-implement solution to meet the requirements of OEMs and other partners. The CDS Framework also enables the plug-in implementer to focus on the data source, rather than worry about the complexities involved in using the Web Intelligence application to analyze data.

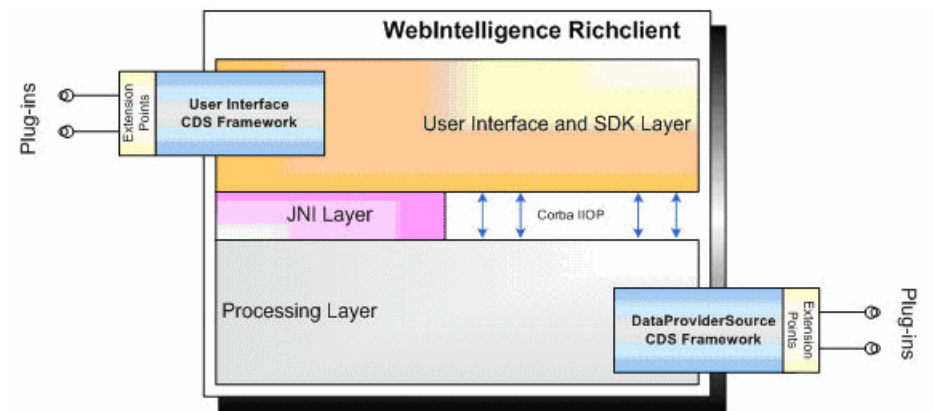
The CDS Framework extension points are located at the following layers of the Web Intelligence Rich Client architecture, as shown in the diagram below:

- User Interface layer
- DataProviderSource processing layer

**Note:**

DataProviderSource processing capabilities may become available in future releases of BusinessObjects Enterprise. Hence, the DataProviderSource Processing layer and Server layer are used interchangeably in this document, Javadocs, and samples.

The CDS Framework consists of the User Interface (client side) CDS Framework and the DataProviderSource (processing/server side) CDS Framework.







# System Workflow



# 3 chapter

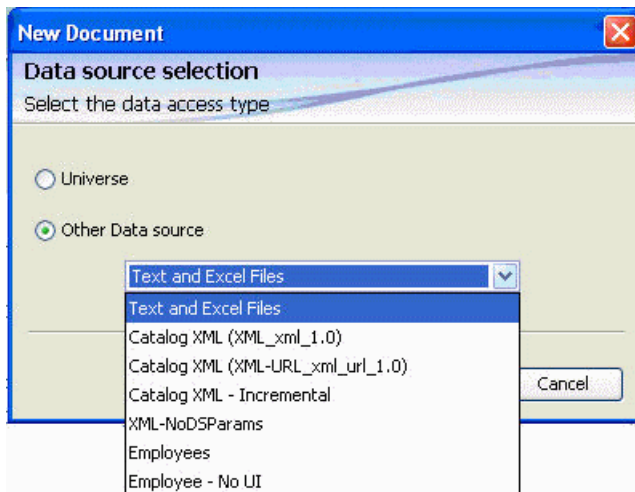
## Workflow of the System

Information about CDS Framework plug-ins is stored in the following configuration file: `webi_customds_extension.xml`.

Users must ensure that this file is available in the following configuration directory: `<BOBJ ENTERPRISE DIR>/<OS_XXX>/config`.

For more information on the configuration file, see [Configuring the Plug-in](#) on page 36.

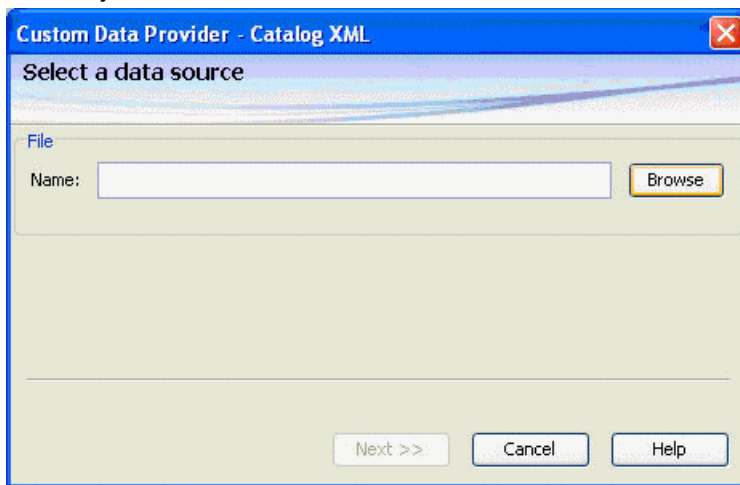
When the Web Intelligence Rich Client application starts, both User Interface (UI) and DataProviderSource (DPS) CDS Frameworks read the required information from the configuration file. These Frameworks read the configuration file separately and verify the validity of information available in the configuration file. If the information provided by the configuration file meets the requirements, the Framework attempts to load and instantiate the User Interface entry point class of the plug-in. If the instantiation is successful, the Framework provides some more useful information to the plug-in and expects the plug-in to prepare itself for use. The Framework then attempts to obtain the display name of the plug-in. When users attempts to create a new .wid document, the list of usable plug-ins is displayed, as shown in the following figure:



**Note:**

The workflow explained in this section may vary depending on the type of extension. For more information on this variance, see [Extension Types](#) on page 33. The workflow and sequence diagram discussed in this section are for extensions with `FILE` as the resource type. These extensions require user input to access the data source and build the data provider.

When a plug-in is selected, a dialog box is displayed so that the user can specify the location of the data source. Users can provide the required input either by using the **Browse** option or by entering the location of the file manually.



After the Framework obtains the user input, the UI Framework sends a request to the processing layer to access the data source and build the data provider. This request contains information about the location of the data source provided by the user, along with some information about the plug-in. The `DataProviderSource` Framework then searches for information about the appropriate plug-in and attempts to load and instantiate the `DataProviderSource` entry point class. Users are presented with appropriate information if an error occurs; otherwise, the Framework provides some useful information along with user input about the data source to the plug-in. At this point, the Framework expects the plug-in to prepare itself for use and validates whether the plug-in is capable of processing the data source provided by the user.

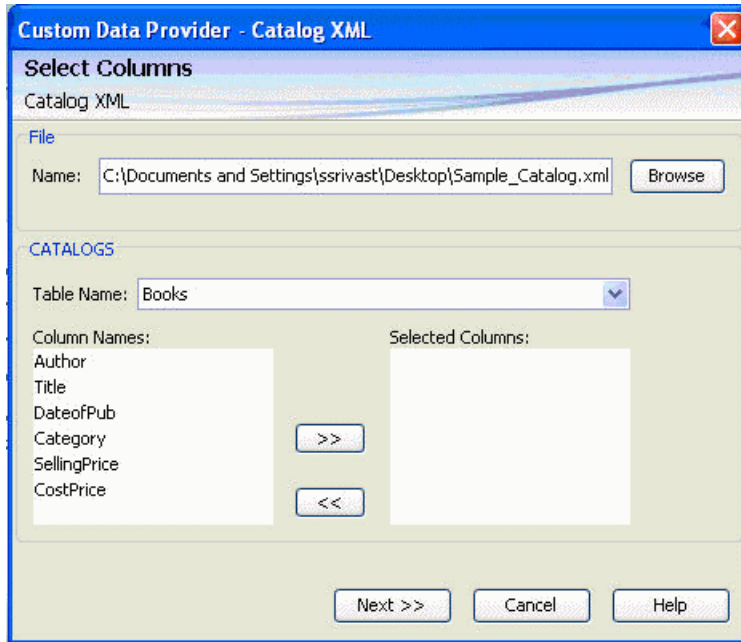
**Note:**

The verification about the correctness or existence of the data source is not performed by the CDS Framework. It is the responsibility of the plug-in to validate and throw appropriate exceptions in case of incorrect source input. If there are any errors, the Framework marks the status of the Data Source as "invalid".

## Data Source Creation Phase

If there are no errors, the data source access/creation phase starts and the DPS Framework checks whether the processing/server-side plug-in implementation is capable of accessing the data source and helps in building the Web Intelligence data provider. For an extension that requires user input other than source information, that is, if the extension has the implementation for the User Interface entry point implementation, the first call returns false. This means that the plug-in is still not capable of providing the required information. If this check by the Framework returns a false value, the Framework marks the status of the Data Source as "incomplete", and attempts to obtain information from the plug-in that will be sent to the UI side plug-in implementation to build its own specific User Interface to obtain the user input. This information is treated as Data Source parameters and is expected to be a name-value pair, where name and value must be non-null `java.lang.String` objects.

After obtaining DS parameters from the processing layer, the UI Framework attempts to obtain the User Interface component that is specific to the plug-in. If an error occurs on the plug-in side while trying to build the user interface, an appropriate error message is displayed. If there are no errors, the plug-in User Interface component is displayed in the dialog box.

**Note:**

- The CDS Framework controls the **Next**, **Cancel**, and **Help** buttons, and also the Source input panel. However, when users click the **Help** button, the plug-in specific **Help** is displayed.
- The dialog box is resized after obtaining the plug-in's UI component. There are no restrictions for the size of the dialog box. Moreover, the dialog box does not resize itself dynamically after obtaining the plug-in's UI component till it performs another operation on the processing layer.

When presented with a new screen, users can provide input and click the **Next** button. When users click the **Next** button, the UI Framework attempts to obtain the updated information from the plug-in.

**Note:**

We recommend that the user interface plug-in must not validate any kind of input. The validation of input must be performed by the Processing layer/Data Provider Source layer of the plug-in implementation.

If there are no errors, the UI Framework again sends a request to the Processing layer to access the Custom data source and build the Data Provider with the new/updated information.

### Note:

The Framework supports both incremental and non-incremental access of data sources. Hence, the plug-in is free to provide either only the new/updated set of Data Source parameters or the complete set. If the plug-in wants to remove any Data Source parameter, it must update the key of this parameter with an empty String value.

After receiving the request to access the Custom Data Source, the CDS Framework checks whether the processing layer/server-side plug-in implementation is now capable of using the custom data source and build the Web Intelligence data provider. Depending on the extension type, the result can be any of the following:

1. The information provided by the user is either incorrect or inadequate.
2. Based on the updated information, plug-in has an additional set of information for which it requires user input.
3. The information provided is adequate and the plug-in is now capable of accessing the data source and help build the Web Intelligence data provider.

In scenario 1 and 2, the Framework again requests the plug-in for the data source parameters. Depending on the scenario, the DPS plug-in is expected to return the new set of information, which is then sent to the UI plug-in. These iterations continue till the DPS plug-in obtains the required information.

### Note:

If the input from the user is invalid, we recommend that the plug-in must alert the user about invalid input.

When the DPS plug-in implementation obtains the required information, it informs the Framework about the availability of this information. The Framework then marks the status of the Data Source as "created".

The DPS Framework now attempts to obtain information from the plug-in for building the Web Intelligence Data Provider. It expects the plug-in to provide details/description of objects /columns that will be included in Web Intelligence Data Provider. Users are alerted if there are any errors. If there are no errors, the DPS Framework attempts to get the instance of DPS plug-in's DataProvider instance. This plug-in's Data Provider instance is used by the Framework to invoke further calls related to retrieval of data. Successful retrieval of this object marks the end of the Data Source phase and also the end of life cycle of the UI component of the UI plug-in implementation.

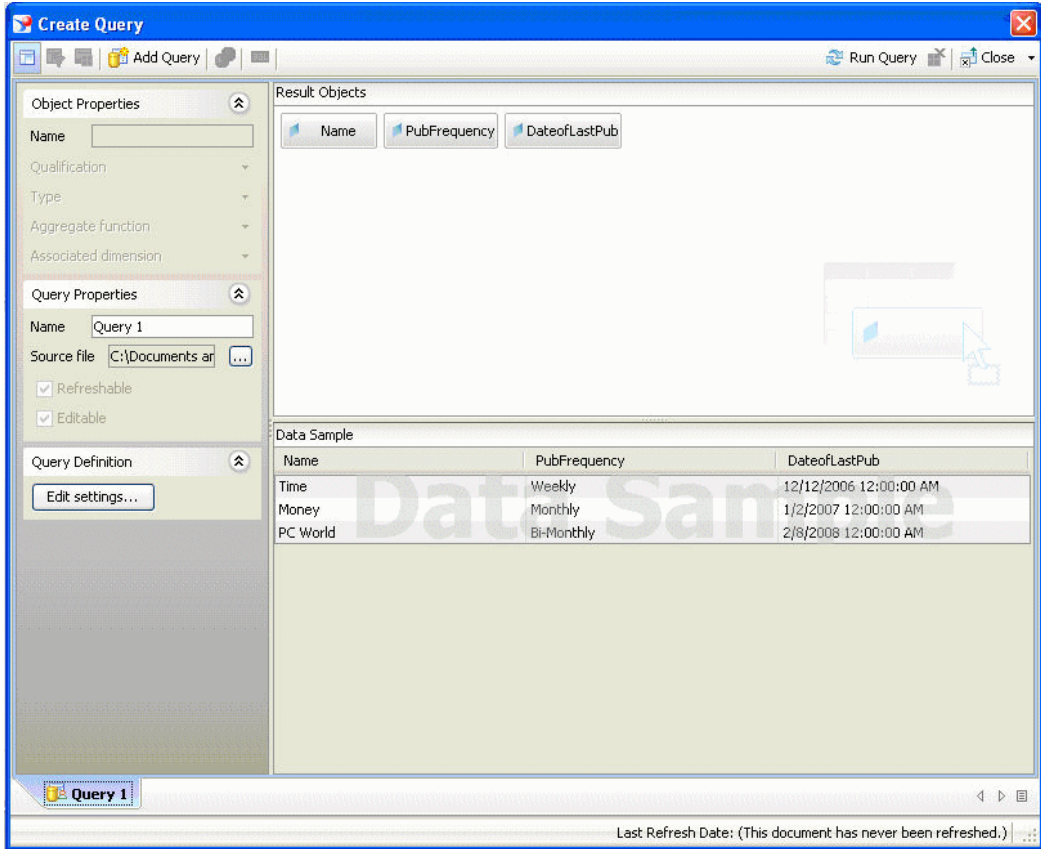
## Data Provider Phase

After the DPS plug-in obtains the `DataProvider` extension point implementation object, the CDS Framework expects the DPS plug-in to take necessary steps to open the data iterator. The CDS Framework also expects that the DPS plug-in has the required information about the number of rows of data corresponding to the data provider objects. If the plug-in has this information, it selects the Framework path by making calls based on Row Count, instead of checking with the plug-in about the availability of the next chunk of data. The plug-in can specify the selection of an alternative path by marking the related property of the Iterator information sent by the DPS plug-in. For more information, see [CustomDataProvider Interface](#) on page 31.

If the DPS plug-in has adequate data for the Web Intelligence data provider objects, the CDS Framework attempts to obtain the data chunk by chunk. If the data required by the CDS Framework is to be used for sampling, the Framework does not request the DPS plug-in for the entire data set. The CDS Framework attempts to retrieve the entire data set only at the time of running or refreshing a query.

The Query panel is displayed while creating the document. Users can perform the following actions by using this panel:

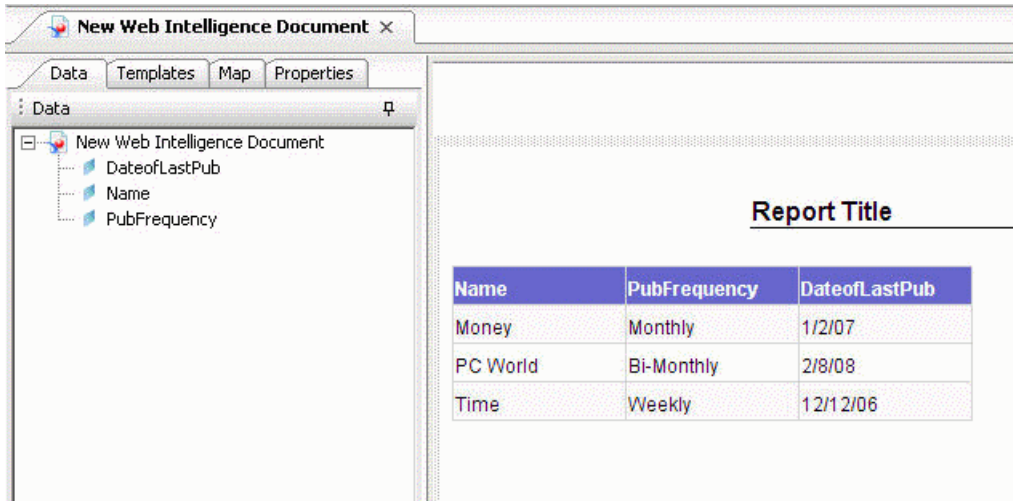
- Change the name, qualification, and type of data provider objects
- Change the Aggregate functions
- Change the DP source
- Edit the settings for the current Data Source



## Note:

- For more information about terms such as qualification, Aggregate functions, options refreshable, and editable, see the Web Intelligence Rich Client documentation.
- The Framework controls the initial assignment of object qualification for data provider objects.
- If the user attempts to change the source on the Query panel, which makes the DPS plug-in incapable of processing the new source/throw error/requires more information to process the new source, the status of the Data Source is marked as "incomplete". In this case, the Framework does not support building the data source information again on the Query panel.

When the user clicks **Run Query** or refreshes the document, the DPS Framework verifies the completeness of information and attempts to obtain the data provider objects again. If there are no errors, the Framework continues with the data provider phase for the entire data set. Upon successful retrieval of the complete data set, user are presented with the Web Intelligence document.







Getting Started



4  
chapter

## Getting Started with Plug-in Development

The CDS Framework conceals the complexities of the Web Intelligence architecture from the CDS Framework plug-in and provides a set of easy-to-implement interfaces that acts as extension points for various functionalities. In the current version, users need to implement only four interfaces to create a Web Intelligence (.wid) document. Users must implement two interfaces for the User Interface entry point and two for the DataProviderSource entry point.

The following interfaces must be implemented for the User Interface CDS Framework:

- `CustomDSExtension` - User Interface entry point of the plug-in
- `CustomDSComponent` - Interface that provides the User Interface component of the plug-in

An abstract implementation of these interfaces is provided. In general, the plug-in developers must extend the respective abstract classes (`AbstractCustomDSExtension` and `AbstractCustomDSComponent`) rather than implement these interfaces.

These interfaces along with their abstract implementation and other common classes of User Interface CDS Framework are bundled in a single binary called `cdsuiframework.jar`, which is available under the `<BOBJ Enterprise Directory>/classes` directory.

The following interfaces must be implemented for the DataProviderSource CDS Framework:

- `CustomDataSource` - DataProviderSource entry point of the plug-in
- `CustomDataProvider` - DataProvider interface of the plug-in

An abstract implementation of these interfaces is provided. In general, the plug-in developers must extend the respective abstract classes (`AbstractCustomDataSource` and `AbstractCustomDataProvider`) rather than implement these interfaces.

These interfaces along with their abstract implementation and other common classes of User Interface CDS Framework are bundled in a single binary

called *cdsframework.jar*, which is available under the <BOBJ Enterprise Directory>/classes directory.

Users can start by creating two new projects, one for UI implementation and the other for DPS implementation, in the IDE, add *cdsuiframework.jar* and *cdsframework.jar* in classpath of respective projects, and then implement the above-mentioned interfaces.

## CustomDSExtension Interface

Following are the `CustomDSExtension` interface methods:

- `init(SessionInfo sessInfo)` - The Framework invokes this method after instantiating the plug-in through the default Constructor. The plug-in is expected to initialize itself for use by the Framework.

The `SessionInfo` method parameter contains the details of the session such as session ID, interface locale at the time when the call is made, and some information about the plug-in. Depending on the start of the life cycle of the plug-in, the session ID at the point of invocation of this method can be an empty String. However, the Framework sends the appropriate value through the `DataHolder` object, which pertains to the session ID, while trying to obtain the instance of `CDSCComponent`.

The following is a sample code snippet of `init(SessionInfo sessInfo)`:

```
public void init(SessionInfo sessInfo) throws CDSExtensionException
{
    interfaceLocale = sessInfo.getInterfaceLocale();
    pluginName = sessInfo.getExtensionName();
    rootHelpPath = sessInfo.getProductHelpPath();
}
Where private Locale interfaceLocale;//used to store the
interface locale
private String pluginName; //used to store the extension
name of plug-in as described in config file.
private String rootHelpPath; // used to store the root help
path of the pug-in.
```

- `setLocale(Locale loc)` - The Framework invokes this method to allow the plug-in to set/reset the locale.

The following is a sample code snippet for `setLocale` :

```
public void setLocale(Locale loc) {  
    interfaceLocale = loc;  
}
```

- `getDisplayName()` - The Framework invokes this method to obtain the localized display name of the plug-in. The CDS Framework uses this information to display the name of the plug-in.

```
public String getDisplayName(){  
    if(pluginName.equalsIgnoreCase(URL_PLUGIN_NAME))  
        return"Catalog XML-URL";  
    else  
        return"Catalog XML";  
}
```

Where `private static String URL_PLUGIN_NAME = "XML-URL";`

- `getFileExtnDescription()` - The Framework invokes this method to obtain the description of plug-in supported file extensions with `FILE` as the resource type. The CDS Framework uses this information to display the **Files of type:** option in the "Open" dialog box.

```
public String getFileExtnDescription() {  
    return"Xml Files";  
}
```

This information can also be provided in the config file. For more information, see [Configuring the Plug-in](#) on page 36.

- `getCDSComponent(DataHolder dataHolder)` - The Framework invokes this method to obtain the `CustomDSComponent` instance. The `CustomDSComponent` instance is used by the Framework to obtain plug-in's User Interface component. The `DataHolder` method parameter contains information that enables the plug-in to build the User Interface.

```
public CustomDSComponent getCDSComponent(DataHolder dataHolder)  
    throws CDSExtensionException  
{  
    return new CatalogUIProvider(dataHolder);  
} // where CatalogUIProvider is a user-defined class that  
implements  
the CustomDSComponent interface. This interface is responsible for the  
look and feel of the plug-in.
```

- `getImageIcon()` - The Framework invokes this method to obtain the plug-in's image icon, if any. The icon is used for purposes such as displaying the Recent Data Sources Menu/Panel.

```
public ImageIcon getImageIcon() throws CDSEExtensionException
{
    return null; // Let the framework get it from FileSystemView
}
```

- `isExtensionHelpAvailable()` - The Framework invokes this method to find out whether it must display plug-in specific help files or product help files.

Plug-in developers can use the following code to avail a help file for the plug-in:

```
public boolean isExtensionHelpAvailable() {
    return true;
}
```

- `getHelpURL()` - The Framework invokes this method to obtain the plug-in's Help URL. This URL can be the absolute path of the help file prefixed with file protocol such as `file:///` or it can be an http-URL. This method is not invoked if the `isExtensionHelpAvailable()` method returns false.

```
public String getHelpURL(){
    StringBuffer helpPath = new StringBuffer();
    helpPath.append("file:///");
    helpPath.append(rootHelpPath);
    helpPath.append(getLanguageFolder());
    helpPath.append("/webintelligence/richclient/html/con
text.htm");
    return
    helpPath.toString();
}
```

- `clean()` - The Framework invokes this method to enable the plug-in perform any clean-up activity.

`AbstractCustomDSEExtension` is the abstract implementation for the `CustomDSEExtension` interface.

## CustomDSComponent Interface

Following are the `CustomDSComponent` interface methods:

- `getUIComponent()` - The Framework invokes this method to obtain the plug-in's User Interface component to be displayed in the user input dialog box.
- `getActionTitle()` - The Framework invokes this method to obtain the localized string of the current action that the user is expected to perform on the User Interface component.
- `getUpdatedDSParams()` - The Framework invokes this method to obtain the updated data source parameters from the plug-in. This method is invoked after the user provides inputs and clicks the **Next** button.
- `processMissingDSParams(Map dsParams)` - The Framework invokes this method to enable the plug-in process invalid, missing, or new data-source parameters. It is expected that the plug-in will alert users about invalid or missing input, or update the User Interface component for accepting another set of input.
- `free()` - The Framework invokes this method to allow the plug-in perform any clean-up activity for the User Interface component.

`AbstractCustomDSComponent` is the abstract implementation for the `CustomDSComponent` interface.

## CustomDataSource Interface

Following are the `CustomDataSource` interface methods:

- `init (CustomDataSourceInfo customDataSourceInfo)` - The Framework invokes this method after the plug-in is instantiated through the default Constructor. It is expected that the plug-in will initialize itself for use by the Framework and verify whether it can process the data source provided by the user.

### Note:

The verification of the validity or existence of the data source is not performed by the CDS Framework. This activity is performed by the plug-in. The plug-in throws exceptions if the source input is incorrect. The `CustomDataSourceInfo()` method parameter contains details such as

source input, session ID, interface locale at the time when the call is made, and some information about the plug-in.

```
public void init(CustomDataSourceInfo dataSourceInfo) throws
DataSourceException{
    CustomDataSourceInfo dsInfo = dataSourceInfo;
    String source = dsInfo.getSource();
    boolean isURLSource = false;
    if(dsInfo.getDataSourceType() != null)
    {
        isURLSource =
        dsInfo.getDataSourceType().equalsIgnoreCase
Case(URL_DSTYPE);//String to
        identify the URL dstype as configured in the configura
tion file
    }
    m_catalogParser = new CatalogXMLFileParser(source,
isURLSource);
    try
    {
        Map catalogDetails = newHashMap();
        catalogDetails.put("Books", "Book");
        catalogDetails.put("Magazines", "Magazine");
        m_catalogParser.parseStructure(catalogDetails);
    }catch (DataSourceException dse) {
        throw dse;
    }
}
catch (Throwable t) {
    /*
    * if the cause is not known better to throw an error with
    "MINOR_GENERIC"
    as Minor code
    * also it is a good practice to put the Major code which can
    closely
    relate to the sequence flow
    */
    throw new DataSourceException (t.getMessage(),ErrorCodes.MA
JOR_GET_INFO_ERROR,
ErrorCodes.MINOR_GENERIC);
}
}
```

- `getDSParameters` (final Map currentDSParams) - The Framework invokes this method to obtain the data source parameters that must be in the form of key-value pairs. The key and value in a key-value pair must be non-null `java.lang.String` objects.

```
public Map getDSParameters(final Map currentDSParams) throws
DataSourceException{
    boolean containCatalogDetails = false;
    boolean containColumnDetails = false;
```

```
if (currentDSParams != null)
{
    /*
    * This check is done to find out if it this call is for
    edit of the source or a new access call.
    * This kind of check can also be done by plug-ins which
    has incremental building of DS.
    * In both cases assumption is that the initial DS params
    are not removed
    */
    Iterator currParamIter = currentDSParams.entrySet().iter
ator();
    while (currParamIter.hasNext ()) {
        Map.Entry currParamEntry = (Map.Entry)
currParamIter.next ();
        String currParamKey = currParamEn
try.getKey().toString();
        if (currParamKey.startsWith("catalog_xml_de
tails."))
            containCatalogDetails = true;
        if(currParamKey.startsWith("catalog_xml_de
tails_"))
            containColumnDetails = true;
        if(containCatalogDetails &&
containColumnDetails)
            break;
    }
}
if (containCatalogDetails && containColumnDetails)
return currentDSParams;
Map newParams = newHashMap();
Map catalogDetails = m_catalogParser.getCatalogDe
tails();
Iterator paramIter = catalogDetails.entrySet().itera
tor();
int index = 1;
while(paramIter.hasNext ()) {
    Map.Entry paramEntry = (Map.Entry)
paramIter.next();
    String paramKey = paramEn
try.getKey().toString();
    newParams.put("catalog_xml_details." + index,
paramKey);
    Vector paramValue = (Vector)paramEntry.getVal
ue();
    /*
    * If the implementation wants to throw an ex
ception with a message that
    is not available in the pre-defined set of
error codes,
    * it is expected to use the minor error code
"MINOR_CUSTOM". However, it
    should be noted that in case of such (Data
```

```

SourceException)
    * exception's with minor code mentioned as
"MINOR_CUSTOM", the onus of passing the localized
string is on the extension point implementa
tion.
    * This localization can be achieved by using
the locale passed by the
framework during init method call.
    * This message is then shown to the user on
the client side.
    */
    if (paramValue == null)
        throw new DataSourceException ("Error While Parsing
the Parameters",ErrorCodes.MAJOR_GET_INFO_ERROR,
ErrorCodes.MINOR_CUSTOM);
        Iterator colIter = paramValue.iterator();
        int colIndex = 1;
        while (colIter.hasNext()) {
            Object colObj = colIter.next();
            if( colObj == null) thrownew DataSourceException
("Unsupported XML",ErrorCodes.MAJOR_GET_INFO_ERROR,
ErrorCodes.MINOR_CUSTOM);
            String colName = colObj.toString();
            newParams.put("catalog_xml_details_"+paramKey+ "."+
colIndex, colName);
            colIndex++;
        }
        if(colIndex == 1)
            throw new DataSourceException ("XML File doesn't
contain structure information",ErrorCodes.MAJOR_GET_INFO_ER
ROR,
ErrorCodes.MINOR_CUSTOM);
            index++;
        }
        return newParams;
    }

```

- `isDataSourceParamsComplete` (final Map currentDSParams) - The Framework invokes this method to obtain information from the plug-in about its ability to access the data source and help build the Data Provider. When this call is made, the result can be any of the following:
  - The plug-in has received correct user input for the first set of data source parameters. However, the plug-in requires more user input for other data source parameter sets.
  - Users attempted to perform the next set of operations (such as obtaining the details of selected objects and sample data) without providing adequate information.
  - Input provided by the user is adequate for the plug-in to access the data source and help build the Data Provider.

In the first two cases, plug-in returns false for the `isDataSourceParamsComplete (final Map currentDSParams)` method. This method returns true for the third case.

```
public boolean isDataSourceParamsComplete (finalMap cur
rentDSParams) throws DataSourceException {
    boolean hasSelectedCols = false;
    Object obj = currentDSParams.get(CDS_XML_SELECTED_COLS);

    if(obj != null) {
        String strSelCols = obj.toString();
        if(!(strSelCols.equals("")))
            hasSelectedCols = true;
    }
    return (currentDSParams.containsKey(CDS_XML_SELECTED_DE
TAIL) &&hasSelectedCols);
}
```

- `getColumnsInfo (final Map currentDSParams)` - The Framework invokes this method to obtain the details of object(s) (column(s)) that are used to build the Web Intelligence data provider.

Following is a sample code snippet:

```
public ColumnsInfo getColumnsInfo (final Map currentDSParams)
throws DataSourceException {
    ColumnsInfo columnsInfo = new ColumnsInfo();
    .....
    String colName = obj.toString();
    Integer colType = ...;
    .....
    columnsInfo.addColumn(new ColumnSpec(colName,colType.int
Value()));
    .....
    return columnsInfo;
}
```

- `getChunkSize ()` - The Framework invokes this method to obtain the size of each chunk provided by the plug-in's `CustomDataProvider` implementation.

```
public int getChunkSize () throws DataSourceException {
    return DEFAULT_CHUNK_SIZE;
}
Where publicstaticfinalint DEFAULT_CHUNK_SIZE = 100;
```

- `getCustomDataProvider (ColumnSpec[] columnSpecs)` - The Framework invokes this method to obtain the implementation for the plug-in's Data Provider.

```
public CustomDataProvider getCustomDataProvider(ColumnSpec[]
columnSpecs) throws DataSourceException {
    return new XMLCatalogDataProvider(m_selCatalogItem,
columnSpecs);
}
```

- `clean ()` - The Framework invokes this method to allow the plug-in to perform any clean-up activity.

`AbstractCustomDataSource` is the abstract implementation for the `Custom DataSource` interface.

## CustomDataProvider Interface

Following are the `CustomDataProvider` Interface methods:

- `openIterator (final IteratorInfo iteratorInfo, ColumnSpec[] columnSpecs)` - The Framework invokes this method to provide the details of Web Intelligence Data Provider objects/columns and the data iterator ID. It is expected that the plug-in will provide the required information for the `IteratorInfo` object. The following points must be noted while implementing this method:

The Framework can take any of the following paths to obtain the data:

- Before each call to the `getNextChunk()` method, the Framework invokes the `hasNextChunk()` method to check whether the next data chunk is available.
- The Framework continues to call the `getNextChunk()` method based on the total number of rows available for the Data Provider, as specified by the plug-in. In this case, the Framework does not invoke the `hasNextChunk()` method at all.

The decision of the Framework is based on the value set by the plug-in for the `isRowCountBased` property. If the value is set to "true", the Framework takes the second path. If not, the Framework takes the first path. The default value is "false".

The Framework provides a unique data iterator ID, which can be retrieved from the `IteratorInfo` object while calling this method. It is expected

that the plug-in will store this information because the rest of the call to the `DataProvider` implementation is based on this `Iterator` ID.

- `setChunkSize (int iIterId, int chunkSize)` - The Framework invokes this method to enable the plug-in to set the chunk size.
- `hasNextChunk (int iIterId)` - The Framework invokes this method to verify whether the plug-in has more chunks to return for the data iterator identified by the ID sent while calling the `openIterator` method.
- `getNextChunk (int iIterId, ColumnSpec[] columnSpecs)` - The Framework invokes this method to obtain the next chunk of data. The value returned must not be null. If the value returned is null, the Framework throws an exception.

The following is a sample code snippet:

```
public Chunk getNextChunk(ColumnSpec int
iIterId, ColumnSpec[] columnSpecs) throws DataSourceException
{
    Chunk chunk = new Chunk (columnSpecs);
    .....
    Iterator rowsIter = vRows.iterator();
    while(rowsIter.hasNext()) {
        Row row = new Row(columnSpecs.length);
        .....
        int colIndex = 0;
        Iterator selColsIter = ...;
        while(selColsIter .hasNext()) {
            Integer colType = ...;
            switch(colType.intValue()) {
                case ObjectTypes.STRING:
                    row.addString(..., colIndex);
                    break;
                case ObjectTypes.NUMBER:
                    Double doubleVal = ...;
                    row.addNumber(doubleVal, colIndex);
                    break;
                case ObjectTypes.DATE:
                    String strColDateVal = ...;
                    Date dateVal = null;
                    if(strColDateVal != null)
                    {
                        SimpleDateFormat sdf = new SimpleDateFormat();
                        sdf.applyPattern("MM/dd/yyyy");
                        sdf.setTimeZone(TimeZone.getTimeZone("GMT"));
                        //set the timezone explicitly otherwise it
                        would take //the default locale and millisecond calcula
                        tion can
                        //be not as aspected
                        dateVal = sdf.parse(strColDateVal);
                    }
            }
        }
    }
}
```

```
        row.addDate(dateVal, colIndex);  
        break;  
    }  
    colIndex++;  
}  
chunk.addRow(row);  
}  
...  
return chunk;  
}
```

- `cancel (int iIterId)` - The Framework invokes this method to allow the plug-in to cancel any operation in progress on the data iterator identified by the ID sent while calling the `openIterator()` method.
- `closeIterator (int iIterId)` - The Framework invokes this method to inform the plug-in about the data iterator identified by the ID sent while calling the `openIterator()` method.

`AbstractCustomDataProvider` is the abstract implementation for the `CustomDataProvider` interface.

## Extension Types

The CDS Framework supports the possibility of different plug-in requirements and, within its limits, attempts to support different use-case scenarios.

The extensions can be broadly categorized as follows:

- Resource-based extensions - Resource-based extensions are extensions that have an exiting location of the data source, either in the form of a physical file location or a URL path (for example, http URL).
- Non-resource-based extensions - Non-resource-based extensions are extensions that do not have an exiting location of the data source. However, these extensions are known to the plug-in. Examples of non-resource-based include in-memory data structures.

Currently, the CDS Framework supports the following extension types:

- Extensions with "FILE" as the resource type - These extensions belong to the category of resource-based extensions.
- Extensions with "URL" as the resource type - These extensions belong to the category of resource-based extensions. If the data source URL uses file protocol, then the resource type of the extension must be "FILE" and not "URL".

- Extensions with "NONE" as the resource type - These extensions belong to the category of non-resource-based extensions.
- Extensions that do not require any user input for accessing the data source and help build the Web Intelligence Data Provider. These can be combined with any of the three extension types mentioned above. If these extensions are combined with a resource-based (File/URL) extension, the claim that they do not need any user-input must be understood as follows: it does not require any input from the user apart from source information.

**Note:**

If an extension chooses to implement this combination, many User Interface feature options will not be available for the extension. For example, the display name of the plug-in cannot be localized, and the plug-in cannot have its own icon or its own help documentation and so on.

For information on configuring and implementing the plug-in, see [Configuring and Deploying the Plug-in](#) on page 35.



# Configuring and Deploying the Plug-in



# 5

chapter



## Overview

After developing the plug-in, users can configure and deploy the plug-in to make it available for Web Intelligence Rich Client users. This section describes how to configure and deploy the plug-in.

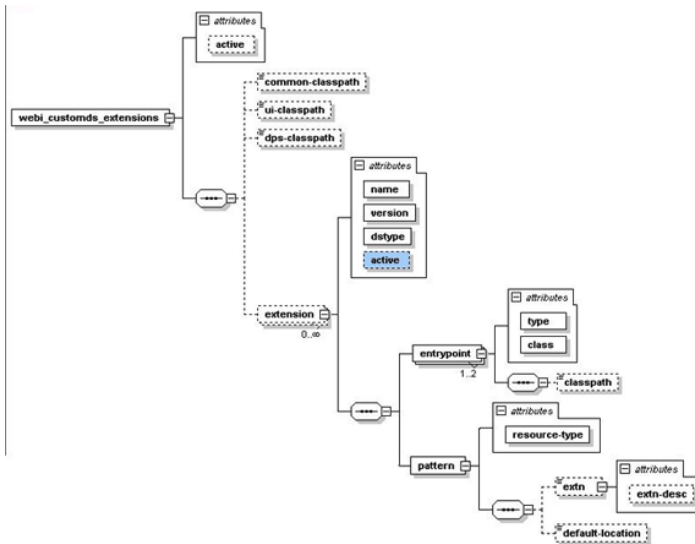
## Configuring the Plug-in

The `webi_customds_extension.xml` (the configuration file) files acts as a repository of plug-ins for the CDS Framework.

To register the plug-in with the CDS Framework, you must specify plug-in specific information in the configuration file and then start or restart the Web Intelligence Rich Client. If the configuration file is not available in the install directory, then you must ensure that it is placed under the BOBJ Configuration Directory (`<BOBJ ENTERPRISE DIR>/<OS_XXX>/config`).

The `.xsd` file is available at the location: `<BOBJ ENTERPRISE DIR>\Samples\customds`

The following diagram illustrates the schema of the configuration file:



Following is the schema definition of the configuration file:

- **active**: This is an attribute of `webi_customds_extensions`. Used to disable all the extensions. If you want to disable all the extensions, value should be set to `false`. Default value is `true`. If set to `false`, all the extensions would be disabled/inactive.
- **common-classpath**: Used to specify classpaths available for all extensions. If the resource is used for more than one extension, the absolute path can be mentioned here. These classpaths would be prefixed separately for each extension's `USERINTERFACE` and `DATAPROVIDERSOURCE` classpath. You can use semi-colon(`;`) separated values if there are more than one resource required by the extension.
- **ui-classpath**: Used to specify classpaths available for all extensions. If the resource is used for more than one extension, the absolute path can be mentioned here. These classpaths would be prefixed separately for each extension's `USERINTERFACE` classpath and suffixed to the `common-classpath`, if any. You can use semi-colon(`;`) separated values if there are more than one resource required by the extension.
- **dps-classpath**: Used to specify classpaths available for all extensions. If the resource is used for more than one extension, the absolute path can be mentioned here. These classpaths would be prefixed separately for each extension's `DATAPROVIDERSOURCE` classpath and suffixed to the `common-classpath`, if any. You can use semi-colon(`;`) separated values if there are more than one resource required by the extension.
- **extension**: Each extension is identified by a unique combination of extension name, version and `dstype`.
- **name**: Used to specify the name of the extension. Should be non-empty, non-localized String.
- **version**: Used to specify the version of the extension. Should be non-empty, non-localized String.
- **dstype**: Used to specify the data source type of the extension. This is internally identified as `ds-sub-type` by the CDS Framework. Should be non-empty, non-localized String.
- **active**: This is an attribute of extension. Used to specify whether the extension is active or not. If you want to disable the extension, value should be set to `false`. Default value is `true`.
- **entrypoint**: Used to specify the implementation class for `UserInterface` and `DataProviderSource`.
- **type**: Used to specify the type of entry point - `USERINTERFACE` or `DATAPROVIDERSOURCE`.

- `class`: Used to specify the qualified class name which implements the entry point interface.
- `classpath`: Used to specify the location of binaries on which extension is dependant. You can use semi-colon(;) separated values if there are more than one resource required by the extension.
- `resource-type`: Used to specify the type of extension - "FILE", "URL" or "NONE".
- `Extn`: Used to specify supported file extensions for an extension of resource-type as "FILE". You can use comma separated values (\*.xml, \*.txt) if there are more than one file-type supported by the extension.
- `default-location`: Used to specify default location for Data Source for resource-based extensions (resource-type as "FILE" or "URL"). You can specify CDATA section for URI having non-xml friendly characters.

Consider the following points while configuring the plug-in:

- Currently, the CDS Framework does not perform a strict validation of the configuration file. Moreover, it does not check whether the XML file is well formed. If the XML file is not well formed, the CDS Framework extensions are not loaded for use. Hence, users must ensure that the configuration file that results after adding the details of the extension is a well-formed XML file.
- Extension must have a unique combination of name, version, and `dstype`. None of these attributes values must be an empty string.
- All characters used in the configuration file must contain byte sequences that comply with UTF-8 encoding requirements.
- All HTML entity reference characters must be escaped by using either numeric character references or strings (for example, "&" and "<" must be replaced with "&amp;" and "&lt;" respectively) unless these characters are part of the CDATA section.
- The CDATA section is allowed only for the `default-location` element.
- Each plug-in operates in a sandbox environment. Hence, you must specify the location and name of the binary on which the plug-in is dependent, except for system (Web Intelligence Rich Client) binaries. This must include the name and location of the binary that contains the class for extension point implementation.
- `$(BOBJ_CLASSES_DIR)` is a keyword that is understood by the Framework while parsing the `classpath` element and refers to the `<BOBJ ENTERPRISE DIR>/classes` directory in the installation. This can be used by extensions if extension-specific binaries are placed in the same directory.

- If "\*" is used for the *extn* element in the configuration file, the value specified for the *extn-desc* attribute in configuration file or the value returned by the plug-in while calling `com.businessobjects.cus tom ds.ui.CustomDSExtension.getFileExtndescription()` is ignored.
- The *classpath* element must contain the absolute path and the name of the binaries on which the plug-in is dependent, including the binary that contains the class for extension point implementation. The entries of the location must be semi-colon ";" delimited.

## Deploying the Plug-in

Check whether the configuration file is present under the `<BOBJ ENTERPRISE DIR>/classes` directory. If it is present, append the required CDS plug-in specific data to the configuration file.

If it is not present, then you must create a configuration file according to the schema, and place it under the `<BOBJ ENTERPRISE DIR>/classes` directory. Specify the CDS Framework plug-in specific information and ensure that the binaries on which the plug-in is dependent are available in the path mentioned in the configuration file.

After you complete the above-mentioned steps, you can create a Web Intelligence (.wid) document from the data source.

## Deploying Samples

To facilitate CDS Framework plug-in development, samples for different extension types are provided. These samples are available under the `<BOBJ ENTERPRISE DIR>/Samples/customds` directory. You can find the sample configuration file and ReadMe in the .zip file provided for each sample. The following samples are provided:

- Catalog XML Sample

This sample demonstrates the implementation of extensions with resource-type as "FILE" and "URL". You can find binary (for User Interface implementation), source code, sample configuration file, and Sample XML that is used as a data source for this extension in compressed file *catalogui.zip* and binary (for Data Provider Source implementation) and source code in the compressed file *ds\_Catalog.zip*. In order to deploy this sample, copy the two binaries available in the compressed files mentioned above into `<BOBJ ENTERPRISE DIR>/classes` directory.

Append information as provided in the sample configuration file in `<BOBJ ENTERPRISE DIR>/<OS_XXX>/config/webi_customds_extension.xml` file. If `<BOBJ ENTERPRISE DIR>/<OS_XXX>/config/webi_customds_extension.xml` file is not available, you can copy the sample configuration file as is at the mentioned location. Now you are ready to use Catalog XML Sample.

- Catalog XML-NoDSParams Sample

This sample demonstrates the implementation of extensions with "FILE" as resource-type. This sample does not require any user input other than source information to access the Data Source and help build the Web Intelligence data provider. You can find the binary (for Data Provider Source implementation), source code, sample configuration file, and the sample XML that is used as a data source for this extension in the compressed file `ds_Catalog_NoDSParams.zip`. To deploy this sample, copy the binary available in the compressed files mentioned above to the `<BOBJ ENTERPRISE DIR>/classes` directory. Append information provided in the sample configuration file to the `<BOBJ ENTERPRISE S_XXX>/config/webi_customds_extension.xml` file. If the `<BOBJ ENTERPRISE DIR>/<OS_XXX>/config/webi_customds_extension.xml` file is not available, you can copy the sample configuration file to the location mentioned above. The XML-NoDSParams sample is now ready for use.

- Catalog XML - Incremental Sample

This sample demonstrates the implementation of extensions with "FILE" as resource type and incremental access of the data source. You can find the binary (for User Interface implementation), source code, sample configuration file, and the sample XML that is used as a data source for this extension in the compressed file `catalogui_incremental.zip`, and the binary (for Data Provider Source implementation) and source code in the compressed file `ds_Catalog_Incremental.zip`. To deploy this sample, copy the two binaries available in the compressed files mentioned above to the `<BOBJ ENTERPRISE DIR>/classes` directory. Append the information provided in the sample configuration file to the `<BOBJ ENTERPRISE DIR>/<OS_XXX>/config/webi_customds_extension.xml` file. If the `<BOBJ ENTERPRISE DIR>/<OS_XXX>/config/webi_customds_extension.xml` file is not available, you can copy the sample configuration file to the location mentioned above. The Catalog XML - Incremental Sample is now ready for use.

- Employees Sample

This sample demonstrates the implementation of extensions with "NONE" as the resource type. You can find the binary (for User Interface implementation), source code, and the sample configuration file for this extension in the compressed file *resource\_none\_ui.zip*, and the binary (for the Data Provider Source implementation) and source code in the compressed file *ds\_Resource\_None.zip*. To deploy this sample, copy the two binaries available in the compressed files mentioned above to the `<BOBJ ENTERPRISE DIR>/classes` directory. Append the information in the sample configuration file to the `<BOBJ ENTERPRISE DIR>/<OS_XXX>/config/webi_customds_extension.xml` file. If the `<BOBJENTERPRISE DIR>/<OS_XXX>/config/webi_customds_extension.xml` file is not available, you can copy the sample configuration file to the location mentioned above. The Employees Sample is now ready for use.

- Employee - No UI Sample

This sample demonstrates the implementation of extensions with "NONE" as the resource type and does not require any user input to access the data source and help build the Web Intelligence data provider. You can find the binary (for Data Provider Source implementation), source code, and the sample configuration file for this extension in the compressed file *ds\_Resource\_None\_NoDSParams.zip*. To deploy this sample, copy the binary available in the compressed file mentioned above to the `<BOBJ ENTERPRISE DIR>/classes` directory. Append the information provided in the sample configuration file to the `<BOBJ ENTERPRISE DIR>/<OS_XXX>/config/webi_customds_extension.xml` file. If the `<BOBJENTERPRISE DIR>/<OS_XXX>/config/webi_customds_extension.xml` file is not available, you can copy the sample configuration file to the location mentioned above. The Employee - No UI Sample is now ready for use.

## 5 | Configuring and Deploying the Plug-in *Deploying Samples*



Points to Consider while  
Developing and Deploying  
the Plug-in

6

chapter



Consider the following points while developing and deploying the plug-in:

- Each plug-in must have a unique combination of plug-in configuration name, version, and dtype. In case of duplicates, both plug-ins are removed from the usable list without displaying any message to the user. However, this information is logged in the log file.
- If there is a display name conflict between the plug-ins (either display name-display name or display name-plug-in name conflict), the Framework attempts to resolve the conflict by displaying `<Plug-in Display Name> (<Plug-in configuration Name>_<Plug-in configuration DStype>_<Plug-in configuration Version>)` to the user when provided with the list of usable plug-ins in the first dialog box of Data Source Selection.
- The information that is traversed between the User Interface layer and the Processing layer for accessing the data source and help in building the Web Intelligence data provider must be in the form of a name-value pair, where name and value must be non-null `java.lang.String` objects.
- The CDS Framework logging information is available in the `jtrace` log file that is generated when logging is enabled for Web Intelligence Rich Client through `BO_trace.ini`. For more information, see the Web Intelligence Rich Client documentation.
- In a scenario where there is more than one column with same name, a suffix such as `_1`, `_2`... is added by the system in the column names while displaying it.
- The Framework is not responsible for any encryption and decryption of Data Source Parameters. The plug-in is responsible for encrypting any of the DS parameters, if necessary.
- If the plug-in has any Data Source parameter for which empty string is a qualified value, it is the plug-in implementer's responsibility to ensure a handshaking between the UI plug-in implementation and the DPS plug-in implementation. The plug-in implementation can include an identifier, which if found is interpreted by the plug-in as an empty string value.
- Whenever a user selects a source, a URL, or a file, and if there are no errors in the Processing layer, the Next button is enabled by default. The Framework does not provide any call-back mechanism to the plug-in developer to disable or enable the **Next** button.
- The Framework calls the `processMissingDSParameters()` method for both incremental and non-incremental scenarios. This means that the Framework invokes the same method on the User Interface plug-in, irrespective of whether the output for request to the processing layer for

accessing the data source and building the Data Provider contains additional set of information or information about invalid or missing inputs.

- The sequence of operations is almost similar for sample and actual data fetch operations.
- The "Change Source" operation by the user in the Web Intelligence Rich Client "Query Panel" is intended to be used for compatible data sources. If the existing data source parameters are not sufficient for the new data source, plug-in cannot re-build the data source parameters set. As a result, this scenario results in an exception.
- Information about Interface Locale may not contain country-specific information.
- In the current version, Web Intelligence prompts/query filters are not available for queries created by using the CDS Framework extension.
- In the current version, features such as refresh and edit query using Java Report Panel/DHTML from InfoView are not supported for queries created by using the CDS Framework extension.
- In the current version, specifying the object hierarchy is not supported for queries created by using the CDS Framework extension.
- In the current version, users cannot open the plug-in supported files directly by using the Web Intelligence Rich Client **Open File** menu item.
- In the current version, the edit operation cannot be performed on extensions with "NONE" as the resource type.
- Currently, the Framework does not provide any extension lookup feature on the client side, which can automatically identify the extension implementation to be used based on a user-selected file (either by using the Open menu item or by means of a drag-and-drop operation inside the WRC application).
- The CDS Framework is not responsible for the look and feel of the User Interface of CDS plug-in.
- The CDS Framework cannot localize error messages provided by the plug-in.
- In the current version, the CDS Framework does not provide any logging mechanism that can be used by the plug-in for logging. The plug-in is expected to have its own logging mechanism, if required.
- If the plug-in has an image icon for Custom Data Source files, then the path of the image icon must be sent to the plug-in. In case the image is not found in the path, the CDS Framework will search for a relevant icon in the system. If the CDS Framework does not find a relevant icon in the system, no icon is displayed for the data source.

To debug the plug-in, add a registry entry that is similar to following:

## 6 | Points to Consider while Developing and Deploying the Plug-in

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Business Objects\Suite 12.0\default\WebIntelligence\RichClient\JVMOptions]
```

The sample registry entry is as follows:

```
"3"="-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=9100"
```

**Note:**

The port number mentioned in the registry must be used while creating remote debug info in the IDE.



Utility Class



7



chapter

Data sources may have complex structures and translating them into name-value pairs that function as input for data source parameters is a very difficult task. To overcome this difficulty, the CDS Framework provides a utility that can take complex structures as input and translate them into name-value pairs as expected by the Framework for DS parameters. The class that can be used for this functionality is called `com.businessobjects.customds.utils.ParamsSerializer`. This utility class can be used for the following purposes:

- Serialization of a `java.util.Map` with the structure specified below to a `java.util.Map` that contains a String (non-null) key-String value pair.
- De-serialization of a serialized `java.util.Map` (generated typically with serialization call) with a String (non-null) key-String value pair back to the map with a structure similar to the original `java.util.Map`.

The current version of the utility has its own limitations.

Following is the structure of the utility:

Original `java.util.Map` can have a key that is a String (non-null) and the `java.util.Map.Entry` value can contain either `java.util.Map` (up to any level, but the elements must conform to the structure of the first level), or `java.util.List` containing only String element(s), or String element(s), or a combination of these three at any level.

## Limitations of the Utility Class

Following are the limitations of the utility class:

- The de-serialized `java.util.Map` contains `java.util.Map(s)`, if any, as an instance of `java.util.HashMap` and List(s), if any, as an instance of `java.util.ArrayList`.
- The serialized `java.util.Map` may contain additional name-value pairs inserted by the Framework that are useful for de-serialization.
- The sequence/order of elements in the de-serialized `java.util.Map` may not be similar to the sequence/order of elements in the original `java.util.Map`.
- The input for the de-serialize method must be the unaltered output `java.util.map` from the serialize method call. In other words, if the serialized map needs to be de-serialized, the serialized map must not be altered.
- Certain keywords are used as identifiers internally by the `ParamsSerializer` class. Currently, if the key used in the map to be serialized contains

any of these keywords, de-serialization may not be executed properly. You may encounter a similar problem if the value of the map entry is empty or null.

The following is an example of the utility of the above-mentioned class:

```
Map servicesMap = new HashMap();
Map portsMap1 = new HashMap();
Map portsMap2 = new HashMap();
Map methodsMap1 = new HashMap();
Map methodsMap2 = new HashMap();
servicesMap.put("QueryService", portsMap1);
servicesMap.put("ReportEngineService", portsMap2);
portsMap1.put("8080", methodsMap1);
portsMap2.put("9090", methodsMap2);
methodsMap1.put("Method1", "InputMethod");
methodsMap1.put("Method1.2", "OutputMethod");
methodsMap2.put("Method2", new HashMap().put(1, "$null$"));
System.out.println("Map is : - "+servicesMap);
ParamsSerializer paramSerialiser = new ParamsSerializer();
Map serialisedMap = ParamsSerializer.serialize(servicesMap);
System.out.println("Serialised Map is : "+serialisedMap);
Map deserialisedMap = paramSerialiser.deserialize(seri
alisedMap);
System.out.println("De Serialized Map is : "+deserialisedMap);
```

The output is as follows:

```
Map is: - {ReportEngineService={9090={Method2=null}},
QueryService={8080={Method1.2=OutputMethod,
Method1=InputMethod}}}
Serialized Map is :
{QueryService#%pm*8080#%pm*Method1.2#%pv*=OutputMethod,
QueryService#%pm*8080#%pm*Method1#%pv*=InputMethod,
ReportEngineService#%pm*9090#%pm*Method2#%pv*=%#%pnl*}
De Serialized Map is :
{ReportEngineService={9090={Method2=null}},
QueryService={8080={Method1.2=OutputMethod,
Method1=InputMethod}}}
```





# Limitations of the CDS Framework



# 8

chapter

The CDS Framework consumes any data source within the following limits:

- Only data provider source extension point implementations that can provide data in a tabular format (Columns and Rows) are supported.
- The CDS Framework does not provide direct/in-built capabilities to include authentication features for data source access. However, authentication feature can be achieved indirectly with data source parameters.
- The CDS Framework does not support wizard-based access of data sources. However, you can simulate a wizard-like workflow (without full-fledged capabilities of a wizard) by implementing incremental access to data sources.
- The CDS Framework does not support automated generation of plug-ins for data sources. However, a few samples are provided to demonstrate implementations of various extensions.



Best Practices



9  
chapter

Following are some best practices that you can consider while developing a plug-in:

- Though there are no technical limitations, allowing for future enhancements, it is recommended that User Interface plug-in classes / interfaces must not refer to any classes/interfaces contained in DPS plug-in binary (jar) or vice versa.
- It is recommended that the DPS plug-in implementation and the User Interface plug-in implementation are packaged in two separate `.jar` files.
- Though the configuration file is not validated against the xsd, it is recommended that the configuration file follows the schema defined in the xsd.



Abbreviations

10



chapter

Following are the abbreviations used in the document:

- DP - Data Provider
- DS - Data Source
- DPS - Data Provider Source
- UI - User Interface
- CDS - Custom Data Source



More Information



---



appendix

Information Resource	Location
SAP BusinessObjects product information	<a href="http://www.sap.com">http://www.sap.com</a>
SAP Help Portal	<p>Select <a href="http://help.sap.com">http://help.sap.com</a> &gt; SAP BusinessObjects.</p> <p>You can access the most up-to-date documentation covering all SAP BusinessObjects products and their deployment at the SAP Help Portal. You can download PDF versions or installable HTML libraries.</p> <p>Certain guides are stored on the SAP Service Marketplace and are not available from the SAP Help Portal. These guides are listed on the Help Portal accompanied by a link to the SAP Service Marketplace. Customers with a maintenance agreement have an authorized user ID to access this site. To obtain an ID, contact your customer support representative.</p>
SAP Service Marketplace	<p><a href="http://service.sap.com/bosap-support">http://service.sap.com/bosap-support</a> &gt; Documentation</p> <ul style="list-style-type: none"> <li>• Installation guides: <a href="https://service.sap.com/bosap-inst-guides">https://service.sap.com/bosap-inst-guides</a></li> <li>• Release notes: <a href="http://service.sap.com/releasenotes">http://service.sap.com/releasenotes</a></li> </ul> <p>The SAP Service Marketplace stores certain installation guides, upgrade and migration guides, deployment guides, release notes and Supported Platforms documents. Customers with a maintenance agreement have an authorized user ID to access this site. Contact your customer support representative to obtain an ID. If you are redirected to the SAP Service Marketplace from the SAP Help Portal, use the menu in the navigation pane on the left to locate the category containing the documentation you want to access.</p>
Developer resources	<p><a href="https://boc.sdn.sap.com/">https://boc.sdn.sap.com/</a></p> <p><a href="https://www.sdn.sap.com/irj/sdn/businessobjects-sdklibrary">https://www.sdn.sap.com/irj/sdn/businessobjects-sdklibrary</a></p>

Information Resource	Location
SAP BusinessObjects articles on the SAP Community Network	<a href="https://www.sdn.sap.com/irj/boc/businessobjects-articles">https://www.sdn.sap.com/irj/boc/businessobjects-articles</a> These articles were formerly known as technical papers.
Notes	<a href="https://service.sap.com/notes">https://service.sap.com/notes</a> These notes were formerly known as Knowledge Base articles.
Forums on the SAP Community Network	<a href="https://www.sdn.sap.com/irj/scn/forums">https://www.sdn.sap.com/irj/scn/forums</a>
Training	<a href="http://www.sap.com/services/education">http://www.sap.com/services/education</a> From traditional classroom learning to targeted e-learning seminars, we can offer a training package to suit your learning needs and preferred learning style.
Online customer support	<a href="http://service.sap.com/bosap-support">http://service.sap.com/bosap-support</a> The SAP Support Portal contains information about Customer Support programs and services. It also has links to a wide range of technical information and downloads. Customers with a maintenance agreement have an authorized user ID to access this site. To obtain an ID, contact your customer support representative.
Consulting	<a href="http://www.sap.com/services/bysubject/businessobjectscounseling">http://www.sap.com/services/bysubject/businessobjectscounseling</a> Consultants can accompany you from the initial analysis stage to the delivery of your deployment project. Expertise is available in topics such as relational and multidimensional databases, connectivity, database design tools, and customized embedding technology.

