



SDK Java Tutorial

Copyright

© 2010 SAP AG. All rights reserved. SAP, R/3, SAP NetWeaver, Duet, PartnerEdge, ByDesign, SAP Business ByDesign, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries. Business Objects and the Business Objects logo, BusinessObjects, Crystal Reports, Crystal Decisions, Web Intelligence, Xcelsius, and other Business Objects products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of Business Objects S.A. in the United States and in other countries. Business Objects is an SAP company. All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary. These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

2010-05-24



Contents

Chapter 1	Introduction	5
	Using the SDK Java tutorial.....	6
	Before you start.....	6
Chapter 2	Navigating through folders	9
	Overview.....	10
	Learning objective.....	10
	Navigating through folders.....	10
	Navigating through folders to list Web Intelligence document in the CMS.....	11
	The folder navigation functionality.....	11
	Listing Web Intelligence documents in a folder.....	13
	Searching for and listing universes in the CMS.....	14
	Listing universes in a folder.....	14
Chapter 3	Viewing a Web Intelligence document	17
	Overview.....	18
	Learning objective.....	18
	Before you start.....	18
	Opening a Web Intelligence document.....	18
	Selecting a Web Intelligence report.....	19
	Setting a pagination mode.....	20
	Setting the image viewing callback.....	21
	Displaying the report in HTML.....	22
	Viewing report parts.....	23

Contents

Chapter 4	Detecting and refreshing prompts	25
	Overview.....	26
	Learning objective.....	26
	Detecting a prompt.....	26
	Retrieving available prompts in the document.....	27
	Retrieving a list of values (LOV) for prompts.....	27
	Setting values to a prompt.....	28
Chapter 5	Saving a Web Intelligence document	31
	Overview.....	32
	Learning objective.....	32
	Saving a document in Personal and Corporate categories.....	32
Chapter 6	Creating a Web Intelligence document	35
	Overview.....	36
	Learning objective.....	36
	Creating a document.....	36
	Saving a new document.....	38
Appendix A	More Information	39
Index		43



Introduction



1
chapter

Using the SDK Java tutorial

The SDK Java Tutorial is a series of code examples that shows you how to do the following:

- Retrieve the list of Web Intelligence documents and universe files available in the repository.
- Refresh Web Intelligence documents.
- Manage prompts.
- Save a document and set the document's properties.
- Create and save a new document.

Before you start

The information in this section applies to all lessons.

The following objects are created at login and used throughout the tutorial:

- `IEnterpriseSession`: This object is created when the user completes a successful login.
- `ReportEngine`: This object is created when the user completes a successful login and is stored in the user's jsp session variable.
- `ISessionMgr`: This object is created when the user completes a successful login and is stored in the user's jsp session variable.
- `IInfoStore`: This object is created when the user completes a successful login and is stored in the user's jsp session variable.

Refer to the file `login.jsp` for information on these objects.

Syntax specific to each tutorial is described before the code examples. For explanations of all other syntax, refer to the *REBean Reference Guide*.

A document can be opened in 2 different ways:

- Using the document ID.
- Using a storage token.

When a document is refreshed, a new storage token is created. This token can be used in another jsp page. Opening a document using a storage token provides better Web Intelligence server performance than using a document ID. Refer to the *Report Engine Java Developer Guide* for more information.

The following table helps you keep track of your position in the tutorials.

Stage	You Learn how to...
<i>Navigating through folders</i>	Navigate through folders to retrieve a list of Web Intelligence documents and universes.
<i>Viewing a Web Intelligence document</i>	View, refresh and navigate through a Web Intelligence document.
<i>Detecting and refreshing prompts</i>	Retrieve a document that contains prompts, refresh and set the document prompts.
<i>Saving a Web Intelligence document</i>	Save a document.
<i>Creating a Web Intelligence document</i>	Create a new Web Intelligence document and save it.

1 | Introduction

Before you start



Navigating through folders



2

chapter



Overview

This chapter describes how to navigate through folders in the Central Management System (CMS), and retrieve a list of documents and universes. Before running this tutorial, you must understand how to navigate through folders and categories to search for Web Intelligence documents and universes using InfoView.

The code used in this tutorial to navigate and view available Web Intelligence documents can be found in `docNav.jsp`. The code to navigate and view available universes can be found in `unvNav.jsp`.

Learning objective

In this chapter you will learn how to program the following:

- Query the CMS to retrieve folder information.
- Navigate through folders
- Retrieve a list of Web Intelligence documents from the CMS.
- Retrieve a list of universes from the CMS.

Navigating through folders

All sub folders that contain Web Intelligence documents are created under the root folder (with id 0). The root folder is created when Web Intelligence is installed. To list folders stored in the CMS, you must first open the root folder, using the root folder ID; it is then possible to list sub folders and create a sub folder navigation functionality.

How To...

[Log out...](#)

Connected user=Administrator

Catalogs

- Folder Navigation

Document lifecycle

- Viewing a WebI doc
- Saving a WebI doc

WebIntelligence document

- Creating a WebI doc
- Drilling in a WebI doc
- Manage WebI doc with Prompts

Document Folder Navigation

Current Folder

Root

Sub-Folders

	ID	Title	Description	ParentID
	3886	Document		0
	248	Feature Samples	Contains examples of new features	0
	247	Report Samples	Contains samples that are shipped with Business Objects Enterprise.	0
	18	User Folders		0

	ID	Title	Description	ParentID
<i>Navigate through folders to see their content.</i>				

In this tutorial, when you open a folder, available sub folders and Web Intelligence documents are listed. The code used to browse sub folders and retrieve the documents and universes is described below.

Navigating through folders to list Web Intelligence document in the CMS

The docNav.jsp script uses the following steps to implement the navigation functionality.

1. Retrieve the variable sID from the HTTP request parameters. The sID parameter is the unique ID for the top folder to be listed. If this parameter is null, the root folder ID (0) is used automatically.
2. Retrieve and list the `IInfoObject` representing the top folder.
3. Retrieve the `IInfoObjects` representing sub folders contained in the top folder. Create links to docNav.jsp for each sub folder, pass the ID of the folder to be navigated in the sID parameter.
4. Retrieve and list the `IInfoObjects` representing Web Intelligence documents contained in the top folder.

The folder navigation functionality

To retrieve a list of folders on a Web Intelligence server, the CMS has to be queried. A query is run using the `IInfoStore` object. In these tutorials the

`IInfoStore` is created when the user logs on to the Web Intelligence server, it is added to the user's jsp session attributes. To retrieve folders or documents in the CMS, `CI_INFOOBJECTS` must be queried. The type of object searched for is controlled using `SI_KIND` in the query and the `CeKind` object. The following code taken from `docNav.jsp` shows how to retrieve the top folder in the document file structure.

Example: Retrieve the top folder in a file structure.

```
IInfoStore iInf = (IInfoStore)session.getAttribute("InfoStore");
...
String sQuery = "SELECT SI_ID, SI_NAME, SI_PARENTID FROM "
    + "CI_INFOOBJECTS WHERE ( SI_ID = "
    + iID + ") AND SI_KIND=\'"
    + CeKind.FOLDER
    + "\' ORDER BY SI_NAME ASC ";
...
IInfoObjectsparentFolders =

(IInfoObjects) iInf.query(sQuery);
```

Note:

In these tutorials, the `ISessionMgr`, `IInfoStore`, `ReportEngine` and `IUserInfo` objects are created when the user logs in (`login.jsp`) and stored in the user's jsp session attributes for later use. These objects are retrieved when necessary using `session.getAttribute(attributeName)`.

By querying on the `SI_ID`, for a specified kind (`CeKind.FOLDER`) only the top or root folder is returned. To query for a list of sub folders, `SI_PARENTID` is used in the query. The result of this query is used to create the navigation mechanism. The following code shows the navigation functionality in `docNav.jsp`.

Example: The navigation functionality in `docNav.jsp`.

```
sQuery = "SELECT SI_ID, SI_NAME, SI_PARENTID FROM "
    + "CI_INFOOBJECTS WHERE ( SI_PARENTID = "
    + iID + ") AND SI_KIND =\'"
    + CeKind.FOLDER
    + "\' ORDER BY SI_NAME ASC ";
IInfoObjects subFolders = (IInfoObjects)
```

```

iInf.query(sQuery);
int iSize = subFolders.size();
...
for (int i=0; i<iSize; i++) {
    IFolder iFld = (IFolder)subFolders.get(i);
    ...
    <% <A HREF="docNav.jsp?sID=<%=iFld.getID() %>"
        <%=iFld.getID() %></A> %>
    ... }

```

Listing Web Intelligence documents in a folder

To retrieve and work with a list of Web Intelligence documents in a folder is essentially the same as retrieving and working with a list of sub folders. What changes is the `CeKind` used in the query. The following code taken from `docNav.jsp` shows how to retrieve a list of Web Intelligence documents.

Example: How to retrieve a list of Web Intelligence documents.

```

sQuery = "SELECT SI_ID, SI_NAME, SI_PARENTID "
+ "FROM CI_INFOOBJECTS WHERE ( SI_PARENTID = "
+ iID + ") AND SI_KIND =\'"
+ CeKind.WEBI
+ "\' ORDER BY SI_NAME ASC";

IInfoObjects webiDocuments =
(IInfoObjects) iInf.query(sQuery);
int wSize = webiDocuments.size();
...
for (int j=0; j<wSize; j++) {
    IInfoObject iObj = (IInfoObject)webiDocuments.get(j);
    ...
}

```

Note:

Web Intelligence documents are represented by an `IInfoObject` in the CMS, their is no specialized object such as `IFolder` used for folders.

Searching for and listing universes in the CMS

Searching for and listing universes is done in the `unvNav.jsp` file. The method used to navigate through folders and list universes is the same as that used to navigate through folders and list Web Intelligence documents. The differences are:

- You select objects from `CI_APOBJECTS` to navigate to and list universes. To navigate and list documents, `IInfoObjects` are retrieved from `CI_INFOOBJECTS`.
- The Root folder ID for Universes is 95

The following code is taken from `unvNav.jsp` shows how to retrieve the top folder of the universe file structure in the CMS. Note that although the query searches in `CI_APOBJECTS`, it is still an object of `SI_KIND`, `CeKind.FOLDER` that is being searched for.

Example: Retrieve the top folder of the universe file structure in the CMS.

```
String sQuery = "SELECT SI_ID, SI_NAME, SI_PARENTID FROM  
CI_APOBJECTS WHERE ( SI_ID='\''  
+ iID + '\'' ) AND SI_KIND =\''  
+ CeKind.FOLDER  
+ '\'' ORDER BY SI_NAME ASC ";
```

To query for a list of sub folders, `SI_PARENTID` is used in the query in the place of `SI_ID`.

Listing universes in a folder

To retrieve and work with a list of universes in a folder is essentially the same as retrieving and working with a list of sub folders. Note that `CeKind.UNIVERSE` is used in the query. The following code taken from `unvNav.jsp` shows how to retrieve a list of universes in a folder.

Example: How to retrieve a list of universes

```
sQuery = "SELECT SI_ID, SI_NAME, SI_PARENTID FROM
CI_APOBJECTS WHERE "
+ "SI_PARENTID=\'" + iID
+ "\' AND SI_KIND=\'" + CeKind.UNIVERSE
+ "\' ORDER BY SI_NAME ASC";

IInfoObjects webiUniverses =
(IInfoObjects) iInf.query(sQuery);
int wSize = webiUniverses.size();
...
for (int j=0; j<wSize; j++) {
    IInfoObject iObj = (IInfoObject)webiUniverses.get(j);
    ...
}
```

2 | Navigating through folders *Searching for and listing universes in the CMS*



Viewing a Web Intelligence
document

3

chapter



Overview

This chapter describes how to view a Web Intelligence document.

Learning objective

In this chapter you will learn how to program the following:

- Open a Web Intelligence document.
- Select a report to view from the list of available reports.
- Set a pagination mode to display a specific page in full page mode.
- Set the callback script for image viewing
- Display the results in HTML

Before you start

Before you start this lesson, you need to know the Document ID or the storage token for the document that you will open. The document ID can be retrieved from the list of documents displayed when running the [Navigating through folders](#) tutorial.

Opening a Web Intelligence document

The first time a document is opened in a user session, it must be opened using the Web Intelligence document ID; at this point a storage token is created. The storage token tracks the document state as it is edited by the user, and can be used later in the user session to open and reconstitute a document in a certain stage of editing.

Note:

Storage tokens are linked to an individual `ReportEngine` instance. It is not possible to open a document using a storage token created by a different instance of a `ReportEngine`.

The JSP file `retrieveReDoc.jsp` retrieves a document ID or storage token passed to it in the request parameter "docIdentifier". This Id is passed to a `ReportEngine` instance which opens the document.

The following code is taken from `retrieveReDoc.jsp`.

Example: Opening a Web Intelligence document.

```
if (! strEntry.equals(""))
{
    //Retrieve the document using its storage token.
    cdzDocument =
        cdzReportEngine.getDocumentFromStorageToken(
            strEntry);
    ...
}
//If a document id has been given.
else if ((! strDocId.equals(""))
{
    //Open the document using its identifier.
    cdzDocument=
        cdzReportEngine.openDocument(
            Integer.parseInt(strDocId));
}
}
```

Selecting a Web Intelligence report

In the previous section a Web Intelligence document was opened. Web Intelligence documents may include more than one report, it is important to retrieve the report the user wishes to view. In `retrieveReReport.jsp` an `int` value pertaining to the report number the user wishes to see is retrieved from the HTTP parameters, if no value is set, the first report (index 0), is used by default. The report requested is retrieved from the list of reports available in the document.

Example: Retrieving a single report from a Web Intelligence document.

```
String strReportIndex =
    getNonNullValue(request.getParameter("report"), "0");
int iReport = Integer.parseInt(strReportIndex);
```

```
Reports cdzReports = cdzDocument.getReports();  
Report cdzReport = cdzReports.getItem(iReport);
```

Setting a pagination mode

Before displaying a report, the pagination mode is set to navigate through the report pages. The following code, taken from `view_doc_HTML.jsp`, shows how to set the pagination mode to page-by-page mode.

Example: Setting the pagination mode

```
String strPage = getNonNullValue(request.getParameter("page"), "");  
//Set a report to be viewed in page-by-page  
//PaginationMode.  
cdzReport.setPaginationMode(PaginationMode.Page);
```

The code below shows how a report is opened to a specific page requested by the current user. Once the page has been set, the document has changed state (see [Opening a Web Intelligence document](#)); a new storage token is retrieved so the user can navigate through report states, that is to say, undo or redo changes that have been made to the document.

Example: Setting the report page and status navigation

```
String strPage = getNonNullValue(request.getParameter("page"), "");  
...  
PageNavigation cdzPageNavigation =  
    cdzReport.getPageNavigation();  
if (strPage.equals(C_STR_PAGE_FIRST)) {  
    cdzPageNavigation.first();  
    //Set strEntry to "" so a new storage token is created  
  
    strEntry = "";  
}  
...  
else if (strPage.equals(C_STR_PAGE_PREVIOUS)) {  
    cdzPageNavigation.previous();  
    // Remember that a new token is required  
    strEntry = "";  
}
```

```
...  
if (strEntry.equals(""))  
//Get storage token  
strEntry = cdzDocument.getStorageToken();
```

Note:

- The document `cdzDocument` is retrieved in `retrieveReDoc.jsp`, see [Selecting a Web Intelligence report](#) for more information.
- The variables `C_STR_PAGE_FIRST`, `C_STR_PAGE_PREVIOUS` etc. are declared in `wistartpage.jsp`.

Setting the image viewing callback

The code below shows how to display a Web Intelligence report in HTML format. If a report contains charts they must be displayed as binary images. It is not possible to send binary data to client web browser using the text HTML stream. To display binary images it is necessary to create a callback script, this script sends image data to the client browser in a binary stream, this script is called from the HTML generated to view the report. The code in the following example is taken from `view_image.jsp`.

Example: A callback script used to display images

```
response.setContentType("image/gif");  
String strEntry = request.getParameter("entry");  
String strImageName = request.getParameter("image");  
...  
ReportEngine cdzReportEngine =  
    (ReportEngine) session.getAttribute("ReportEngine");  
DocumentInstance cdzDocument =  
    cdzReportEngine.getDocumentFromStorageToken(strEntry);  
Image objImage = cdzDocument.getImage(strImageName);  
byte[] abyBinaryContent = objImage.getBinaryContent();  
ServletOutputStream objServletOutputStream =  
    response.getOutputStream();  
response.setContentLength(abyBinaryContent.length);  
objServletOutputStream.write(abyBinaryContent);  
objServletOutputStream.flush();  
objServletOutputStream.close();
```

The following code example from `view_doc_HTML.jsp` shows how to set an image callback script so charts can be displayed correctly when a report is viewed in HTML format.

Example: Setting an image callback script

```
ImageOption cdzImageOption = cdzDocument.getImageOption();
cdzImageOption.setImageCallback("view_image.jsp");
cdzImageOption.setImageNameHolder("image");
cdzImageOption.setStorageTokenHolder("entry");
```

Displaying the report in HTML

Before displaying a report, the document is set to open a report in the requested format at a specific page. After an image callback has been set, a HTML view is now retrieved and sent to the client browser.

Example: Retrieve and display a report page in HTML format.

```
HTMLView cdzHtmlView =
    (HTMLView) cdzReport.getView(OutputFormatType.HTML);
...
<%
// In order to increase performance, call getContent
and parse the returned HTML.
String strContent = cdzHtmlView.getContent();
// The following code is written on the assumption that
the head and body tags are not written in the compact
form (<HEAD/> and <BODY/>).

// Search for the HEAD tag attributes.
int iPositionHeadAttributesBegin =
    strContent.indexOf("<head") + "<head".length();
int iPositionHeadAttributesEnd = strContent.indexOf(
    ">", iPositionHeadAttributesBegin);

int iPositionHeadContentBegin =
    iPositionHeadAttributesEnd + 1;
int iPositionHeadContentEnd = strContent.indexOf(
    "</head>", iPositionHeadContentBegin);

String strHeadAttributes = strContent.substring(
    iPositionHeadAttributesBegin,
```

```
iPositionHeadAttributesEnd);  
String strHeadContent = strContent.substring(  
    iPositionHeadContentBegin, iPositionHeadContentEnd);  
  
// Search for the BODY tag attributes.  
...  
%>
```

Viewing report parts

The report part can be a block, section, image or cell within a report. Each report part is identified by the reference.

There are two types of report part references:

- **Unique Reference:** It is a constant reference that identifies a single element in the report structure. Its occurrence in the report output is associated with the data in the cube.
- **Temporary Reference:** It is a temporary reference that identifies a single element in the report structure. Its occurrence in the report output is index-based.

You can find the report part reference of the particular report element by getting the report output in the XML view.

Enter Reference ID:
'| to be used to separate multiple reference ID's

Select Report Part View Type :

```
- <cell ref="Fs.Fv" bid="1017" x="198" y="12" h="42" w="402" pad="1" bt="0,0,1,0" sid="3">
  <ct>Report Title</ct>
</cell>
- <graph ref="Fs.Fw" bid="1018" x="32" y="92" h="602" w="573" bt="4" sid="6">
  <blob key="dxXMLDraft_1*1*3" type="image/gif" pixel_width="573" pixel_height="602" />
</graph>
- <graph ref="Fs.GD" bid="1037" x="650" y="77" h="230" w="288" sid="8">
  <blob key="dxXMLDraft_1*1*4" type="image/gif" pixel_width="288" pixel_height="230" />
</graph>
- <graph ref="Fs.Gi" bid="1068" x="86" y="721" h="230" w="288" sid="8">
  <blob key="dxXMLDraft_1*1*5" type="image/gif" pixel_width="288" pixel_height="230" />
</graph>
```

You can view the report parts in the following format types:

- PDF
- DHTML
- XML
- XLS

Example: Viewing the report part in PDF format

```
String[] reportPartReference = new String[1];
reportPartReference[0] = "Fs.Fw";
ReportParts reportParts = document.getReportParts(reportPartReference, OutputFormatType.PDF);
BinaryView reportPartView = (BinaryView) reportParts.getView();

OutputStream outputStream = response.getOutputStream();
response.setContentType("application/pdf");
response.setHeader("Content - Type", "application/pdf");
response.setDateHeader("expires", 0);
reportPartView.getContent(outputStream);
outputStream.close();
```



Detecting and refreshing prompts



4

chapter



Overview

This chapter describes how to detect prompts in a Web Intelligence document, and to retrieve and set the prompts contained within it.

Learning objective

In this chapter you will learn how to program the following:

- Detect prompts in a document.
- Retrieve a list of prompts to be filled.
- View and refresh the list of values (LOV).
- Set the prompts in a document.

Detecting a prompt

In this tutorial, prompt detection, filling and setting is achieved by repeat passes through `refresh.jsp` and associated files. On the first pass, a document ID or storage token is passed to the jsp script, which opens the corresponding document using the `ReportEngine`. If the current user has the appropriate rights, prompts are detected by refreshing a document. Prompts are detected by calling `DocumentInstance.getMustFillPrompts()`. If prompts are detected the method `getPromptsHtml` in `processTC_Prompts_methods.jsp` is called.

The method `getPromptsHtml` generates a form containing the list of prompts with their current values in HTML text boxes, and buttons permitting the user to view List Of Values (LOV) values assigned to a prompt. When prompt values are filled, the user is redirected to `refresh.jsp`, prompt values are passed via the parameter string.

If the document contains nested prompts, `getPromptsHtml` is called recursively until all prompts are filled.

Retrieving available prompts in the document

Prompts are retrieved by calling `cdzDocument.getPrompts()`, nested prompts are contained in a List of Values (LOV) associated to the document. The following code shows how to retrieve both simple and nested prompts.

Example: The following code is taken from `viewTCDoc_promptsForm.jsp` and `processTC_Prompts_methods.jsp`.

```
Prompts cdzPrompts = null;
//If strPrefix is PV, no prompts have been filled.
if (strPrefix.equals("PV"))
{
    strLovId = "";
    // Get the document's root prompts.
    cdzPrompts = cdzDocument.getPrompts();
}
else
{
    // Get the lov id whose nested prompts need to be filled

    strLovId =
        strLovIdList.substring(
            strLovIdList.lastIndexOf('-') + 1);
    cdzPrompts = cdzDocument.getLOV(strLovId,
        LovType.LOV_OBJECT).getNestedPrompts();
}
...
//iterate through prompts
for (int iPromptIndex = 0; iPromptIndex < iPromptCount;
    iPromptIndex++)
{
    Prompt cdzPrompt = cdzPrompts.getItem(iPromptIndex);
    PromptType cdzPromptType = cdzPrompt.getType();
```

Retrieving a list of values (LOV) for prompts

In the previous part of the tutorial, you retrieved the prompt. Now you will:

- Retrieve the List Of Values (LOV) for the prompt
- Parse all values in the LOV

The user now has a list of values they can select from for the prompt. The following code is taken from `processTC_Prompts_methods.jsp`.

Example: Retrieving LOV from a document

```
//Retrieve the List Of Values (LOV) for the prompt.
Values cdzLovValues = cdzLov.getAllValues();
// Parse all values in the LOV.
for (int iValueIndex = 0;
     iValueIndex < cdzLovValues.getCount(); iValueIndex++)
{
    String strLovValue, strCompleteLovValue;
    // For a multi column LOV.
    if (cdzLovValues.isMultiColumns())
    {
        RowValue objRowValue =
            cdzLovValues.getRowValue(iValueIndex);
        strLovValue = objRowValue.getItem(0);
        objStringBufferCompleteLovValue.setLength(0);

        // Display each value/row as the concatenation of the
        //value of each column of the row.
        for (int iRowValueIndex = 0; iRowValueIndex <
            objRowValue.getCount(); iRowValueIndex++)
            ...
    }
    else
    {
        strLovValue = cdzLovValues.getValue(iValueIndex);
        strCompleteLovValue = strLovValue;
    }
}
```

Setting values to a prompt

Once a user has the list of values, one or more values for the prompt is selected using: `cdzPrompt.enterValues(astrPromptSelectedValues)`;

Once the user has entered values, call the `DocumentInstance.setPrompts` method to validate the user choice. You can now retrieve the HTML document for display.

Example: The following code is taken from `processTC_Prompts_methods.jsp`.

```
// Set the values of the given prompts
public void setPrompts(ServletRequest objServletRequest,
    Prompts cdzPrompts, int iPromptCount, String strPrefix)
{
    for (int iPromptIndex = 0; iPromptIndex < iPromptCount;
        iPromptIndex++)
    {
        Prompt cdzPrompt = cdzPrompts.getItem(iPromptIndex);
        String strPromptParamName = strPrefix + "."
            + String.valueOf(iPromptIndex);

        String[] astrPromptSelectedValues =
            objServletRequest.getParameterValues(
                strPromptParamName);

        // Split its value list into an array if needed
        if (astrPromptSelectedValues != null)
            if (astrPromptSelectedValues.length == 1)
                astrPromptSelectedValues =
                    splitValues(astrPromptSelectedValues[0]);

        if (astrPromptSelectedValues != null)
            // Enter the selected values to the prompt.
            cdzPrompt.enterValues(astrPromptSelectedValues);
    }
}

//The method setPrompts (above) is used for a LOV as
follows

Lov cdzLov = cdzDocument.getLOV(strLovId,
    LovType.LOV_OBJECT);

// Set each nested prompt
Prompts cdzPromptsNested = cdzLov.getNestedPrompts();
int iPromptNestedCount = cdzPromptsNested.getCount();
setPrompts(request, cdzPromptsNested,
    iPromptNestedCount, strPrefix);
// The nested prompts have been filled
cdzLov.setNestedPrompts();
```

4 | Detecting and refreshing prompts *Setting values to a prompt*



Saving a Web Intelligence document



5

chapter



Overview

This chapter describes how to save a Web Intelligence document.

Learning objective

In this chapter you will learn how to program the following:

- Save a Web Intelligence document.
- Reopen the document and set properties.

Saving a document in Personal and Corporate categories

In the Central Management System, a document is stored in a folder and may be attached to a category or categories. In the previous tutorial [Navigating through folders](#), you were shown how to navigate through folders. The purpose of this tutorial is to:

- Save the document in a corporate folder with a new name.
- Attach the new document to the user's personal root category.
- Attach the new document to corporate categories.
- Modify the documents comments and keywords.

Note:

This tutorial will not overwrite a document with the same name in the same folder.

The following code is taken from `save.jsp`.

Example: Save a document copy in a folder and attach specified categories.

```
IInfoObjects personalCategories = null;  
...  
// CeSecurityID.Folder.PERSONAL_CATEGORY is the parent
```

```
// folder for root personal category objects
String sQuery = "SELECT SI_ID FROM CI_INFOOBJECTS WHERE
"
    + "(SI_NAME like '" + userName
    + "') AND (SI_PARENTID = "
    + CeSecurityID.Folder.PERSONAL_CATEGORIES
    + " ) AND SI_KIND = '\" + CeKind.PERSONALCAT
    + "\"";
personalCategories = (IInfoObjects) iInf.query(sQuery);
...
//Retrieve Personal Category
Vector objCatList = new Vector();
ICategory iPersonalCat =
    (ICategory)personalCategories.get(0);
Integer iCatID = new Integer(iPersonalCat.getID());
objCatList.add(iCatID);
...
//Retrieve Corporate category
Vector objCatCorpList = new Vector();
...
ICategory iCorpCat = (ICategory)categories.get(0);
Integer iCatCorpID = new Integer( iCorpCat.getID());
objCatCorpList.add(iCatCorpID);
...
//Save the document
cdzDocument.saveAs(strName,iFolderID,
    (List)objCatList, (List)objCatCorpList);
...
//Open the document previously saved
cdzDocument = cdzReportEngine.openDocument(strName,
    String.valueOf(iDocument.getID()), "corporate", "wid");

//Set the properties defined by the user
Properties strDocProps = cdzDocument.getProperties();
strDocProps.setProperty(
    PropertiesType.KEYWORDS, strKeywords);
strDocProps.setProperty(PropertiesType.DESCRPTION ,
    strComments);
cdzDocument.setProperties(strDocProps);
```

5 | Saving a Web Intelligence document

Saving a document in Personal and Corporate categories



Creating a Web Intelligence
document



6

chapter



Overview

This chapter describes how to create and save a new Web Intelligence document.

Learning objective

In this chapter you will learn how to program the following:

- Create a Web Intelligence document.
- Define parameters for the document.
- Save the newly created document in the CMS.

Creating a document

The following code example shows how to run the document viewer java applet in an HTML page. It contains a list of parameters that apply to the Web Intelligence server. In this tutorial the java applet is used to create a new document.

The following parameters are set to run the applet that are specific to document creation:

- **CdzSession:** a `ReportEngine` server instance is created by calling `ReportEngine.createServerInstance`.
- **DocumentID:** a string containing the unique ID for the target document. To create a document an empty string is passed.
- **UniverseID:** A string containing the CUID for the universe to be used to create a document. A CUID is a unique identifier for an object in the central management system (CMS). The UniverseID string is set in the following format "UnivCUID="CUID value for universe".
- **Save As:** A link to the JSP page which generates a form used to gather information used to save the document.

To create a new document, pass an empty string as the docID parameter. Before you save a document, the document does not exist in the CMS, thus, the document has no ID.

When you save a new document in this tutorial, a save page appears. Enter the document title and description information. This information is sent back to the applet which in turns tells the Web Intelligence server to save the document. The server now creates an `IInfoObject` containing the document with the information that has been passed.

The following code is taken from `CreateWebiDoc.jsp`.

Example: Running the Web Intelligence applet

```
document.writeln(
  '<APPLET name="webiApplet"' + embed_size_attr +
  'codebase="<%= request.getContextPath() %>/creatingWebiDoc/webiApplet/'
  + 'archive="ThinCadenza.jar" '
  + 'code="com.businessobjects.wp.tc.TCMain"> '
  + '<param name="Isapi"
  value="<%= currPath %>cdzServlet"></param>'
  + '<param name="Server"
  value="<%= request.getServerName() %>"></param>'
  + '<param name="Protocol" value="http"></param>'
  + '<param name="Port"
  value="<%= request.getServerPort() %>"></param>'
  + '<param name="Type" value="signed"></param>'
  + '<param name="WebiSession"
  value="<%= strWISession %>"></param>'
  + '<param name="CdzSession"
  value="<%= instanceID %>"></param>'
  + '<param name="DocumentID" value="<%= "" %>"></param>'

  + '<param name="UniverseID"
  value="UnivCUID=<%=webiUniverse.getCUID() %>">
  </param>'
  + '<param name="bRobot" value="false"></param>'
  + '<param name="bTraceInLogFile" val
ue="false"></param>'
  + '<param name="HelpRoot"
  value="<%= request.getContextPath().substring(1) %>">

  </param>'
  + '<param name="SaveAs"
  value="<%= request.getContextPath() %>
  /creatingWebiDoc/save.jsp?folderID=
```

```
<%=iFolderID%>&unvId=<%=sID%>"></param> '  
+ '<param name="Lang"  
  value="<%= request.getAttribute( "lang" ) %>">  
</param> '  
+ '</APPLET>' );
```

Saving a new document

The `save.jsp` page included in this tutorial displays the save document user interface. Information retrieved from the interface is passed to `createWebiDoc.jsp` using the following javascript function:

```
window.opener.saveDocumentCall(value[0],value[1],value[2],value[3],value[4],value[5],value[6]);
```

In `createWebiDoc.jsp`, the javascript function called in `save.jsp` is declared as follows:

```
function saveDocumentCall( title, desc, keyword,  
folderID, corpCat, persCat, refreshOnOpen ) {  
  
  document.applets[0].saveDocument( title, desc, keyword,  
  
    folderID, corpCat, persCat, refreshOnOpen );}
```

You can use the "liveConnect" technique to call an applet using javascript. See <http://java.sun.com/products/plugin/1.3/docs/jsobject.html> and

<http://www.sislands.com/javascript/week9/java/example.htm> for more information.



More Information



A

appendix

Information Resource	Location
SAP BusinessObjects product information	http://www.sap.com
SAP Help Portal	<p>Select http://help.sap.com > 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>http://service.sap.com/bosap-support > Documentation</p> <ul style="list-style-type: none"> • Installation guides: https://service.sap.com/bosap-inst-guides • Release notes: http://service.sap.com/releasenotes <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>https://boc.sdn.sap.com/</p> <p>https://www.sdn.sap.com/irj/sdn/businessobjects-sdklibrary</p>

Information Resource	Location
SAP BusinessObjects articles on the SAP Community Network	https://www.sdn.sap.com/irj/boc/businessobjects-articles These articles were formerly known as technical papers.
Notes	https://service.sap.com/notes These notes were formerly known as Knowledge Base articles.
Forums on the SAP Community Network	https://www.sdn.sap.com/irj/scn/forums
Training	http://www.sap.com/services/education 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	http://service.sap.com/bosap-support 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	http://www.sap.com/services/bysubject/businessobjectscounseling 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.



Index

C

- categories 32
 - corporate 32
 - personal 32
- categories example 32
- CESDK
 - navigating 10
 - searching 11
- CMS
 - query objects 11
 - searching 11

D

- document
 - example opening 18
 - opening 18
 - saving 32
 - searching 11
- documents
 - listing 13

F

- folders
 - navigation 11
 - retrieve current 11
 - retrieve subfolders 11

I

- images 21
 - example callback 21
 - viewing 21

L

- LOV
 - prompt 27

N

- navigating 10

P

- pagination
 - example 20
 - mode 20
- prompt
 - detecting 26
 - LOV 27
 - retrieving 27
 - retrieving example 27
 - retrieving 27
 - retrieving example 27
 - setting 28
 - setting example 28
 - values 28

R

- report 19
 - display html 22
 - displaying 22
 - example 19
 - HTML 22
 - opening 19

Index

S

saving 32
searching 11

U

universe
 searching 14

W

WebIntelligence
 opening 18
 searching 11