



# SDK Java Tutorial

- Copyright** Copyright © 2004 Business Objects. All rights reserved.  
If you find any problems with this documentation, please report them to Business Objects in writing at [documentation@businessobjects.com](mailto:documentation@businessobjects.com).
- Trademarks** Business Objects, the Business Objects logo, Crystal Reports, and Crystal Enterprise are trademarks or registered trademarks of Business Objects SA or its affiliated companies in the United States and other countries. All other names mentioned herein may be trademarks of their respective owners.  
Contains IBM Runtime Environment for AIX(R), Java(TM) 2 Technology Edition Runtime Modules (c) Copyright IBM Corporation 1999, 2000. All Rights Reserved.  
This product includes code licensed from RSA Security, Inc. Some portions licensed from IBM are available at <http://oss.software.ibm.com/icu4j>.
- Use restrictions** This software and documentation is commercial computer software under Federal Acquisition regulations, and is provided only under the Restricted Rights of the Federal Acquisition Regulations applicable to commercial computer software provided at private expense. The use, duplication, or disclosure by the U.S. Government is subject to restrictions set forth in subdivision (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at 252.227-7013.
- Patents** Business Objects owns the following U.S. patents, which may cover products that are offered and sold by Business Objects: 5,555,403, 6,247,008 B1, 6,578,027 B2, 6,490,593 and 6,289,352.
- Part Number** 307-10-610-01
- Third-party contributors** Business Objects products in this release may contain redistributions of software licensed from third-party contributors. Some of these individual components may also be available under alternative licenses. A partial listing of third-party contributors that have requested or permitted acknowledgments, as well as required notices, can be found at:  
<http://www.businessobjects.com/thirdparty>

# Contents

<b>Chapter 1</b>	<b>Introduction</b>	<b>3</b>
	Using the SDK Java tutorial .....	4
	Before you start .....	4
<b>Chapter 2</b>	<b>Navigating through folders</b>	<b>5</b>
	Overview .....	6
	Navigating through folders .....	7
	Navigating through folders to list Web Intelligence document in the CMS ..	7
	Searching for and listing universes in the CMS .....	9
<b>Chapter 3</b>	<b>Viewing a Web Intelligence document</b>	<b>11</b>
	Overview .....	12
	Opening a Web Intelligence document .....	13
	Selecting a Web Intelligence report .....	14
	Setting a pagination mode .....	15
	Setting the image viewing callback .....	16
	Displaying the report in HTML .....	17
<b>Chapter 4</b>	<b>Detecting and refreshing prompts</b>	<b>19</b>
	Overview .....	20
	Detecting a prompt .....	20
	Retrieving available prompts in the document .....	21
	Retrieving a list of values (LOV) for prompts .....	21
	Setting values to a prompt .....	22
<b>Chapter 5</b>	<b>Saving a Web Intelligence document</b>	<b>25</b>
	Overview .....	26
	Saving a document in Personal and Corporate categories .....	26

## Contents

<b>Chapter 6</b>	<b>Creating a Web Intelligence document</b>	<b>29</b>
	Overview .....	30
	Creating a document .....	30
	Saving a new document .....	32
	<b>Index</b>	<b>33</b>

Introduction

# 1

chapter

## Using the SDK Java tutorial

The SDK Java Tutorial is a series of code examples that show 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.



# 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**.

The following table helps you keep track of your position in the tutorials. The current stage is highlighted.

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

## 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.

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

*Navigate through folders to see their content.*

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 ";
...
IInfoObjects parentFolders =
    (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_APPOBJECTS` 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_APPOBJECTS`, 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_APPOBJECTS 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_APPOBJECTS 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);  
    ...  
}
```

---



# Viewing a Web Intelligence document



# 3

chapter



## Overview

This chapter describes how to view a Web Intelligence document.

The following table helps you keep track of where you are in the tutorial. The current stage is highlighted.

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

## 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 [Chapter 2: 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 [Chapter 3: 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 [Chapter 3: Selecting a Web Intelligence report](#) for more information.

**Note:** 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.
...
%>
```

---

### 3 | Viewing a Web Intelligence document *Displaying the report in HTML*



# 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.

The following table helps you keep track of where you are in the tutorial. The current stage is highlighted.

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

The following table helps you keep track of where you are in the tutorial. The current stage is highlighted.

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

## 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 [Chapter 2: 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.

The following table helps you keep track of where you are in the tutorial. The current stage is highlighted.

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

## 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" value="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>'<br>+ '<param name="Lang"<br>    value="<%= request.getAttribute( "lang" ) %>"><br>    </param>'<br>+ '</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.

# Index

## C

- categories 26
  - corporate 26
  - personal 26
- categories example 27
- CESDK
  - navigating 7
  - searching 7
- CMS
  - query objects 8
  - searching 7

## D

- document
  - example opening 13
  - opening 13
  - saving 26
  - searching 7
- documents
  - listing 9

## F

- folders
  - navigation 7, 8
  - retrieve current 8
  - retrieve subfolders 8

## I

- images 16
  - example callback 16
  - viewing 16

## L

- LOV
  - prompt 21

## N

- navigating 7

## P

- pagination
  - example 15
  - mode 15
- prompt
  - detecting 20
  - LOV 21
    - retrieving 21
    - retrieving example 22
  - retrieving 21
  - retrieving example 21
  - setting 22
  - setting example 22
  - values 22

## R

- report 14
  - display html 17
  - displaying 17
  - example 14
  - HTML 17
  - opening 14, 14

## S

- saving 27
- searching 7

## U

- universe
  - searching 9

## W

- WebIntelligence

## Index

opening 13  
searching 7