# Oracle® Fusion Middleware

Developer's Guide for Oracle Business Intelligence Publisher

11*g* Release 1 (11.1.1)

**E22259-04**

December 2013

Explains how to incorporate Oracle Business Intelligence Publisher functionality into custom applications using the Java and Web services application programming interfaces.

Oracle Fusion Middleware Developer's Guide for Oracle Business Intelligence Publisher, 11*g* Release 1 (11.1.1)

E22259-04

Primary Author: Leslie Grumbach Studdard

Contributing Author: Suzanne Gill

Contributors: Oracle Business Intelligence Publisher development, product management, and quality assurance teams

# Contents

# Preface

Oracle Business Intelligence Publisher (BI Publisher) is a comprehensive set of enterprise reporting tools and infrastructure, including a scalable and efficient query and data generation engine, enterprise reporting document generation, interactive report consumption, and scheduled report execution and delivery. Oracle BI Publisher is designed to author, generate, and deliver all the operational documents you need to run your organization and provide greater insight to a wide variety of users.

Oracle BI Publisher provides a common service-oriented architecture (SOA) and Java APIs for data access and generation, document generation and delivery, a security model and user preferences, and Web-based administration. Oracle BI Publisher provides scalability and performance with a multi-tier architecture that separates data generation from report generation and rendering. Oracle BI Publisher also provides sophisticated data and document caching services, and can be clustered for high availability or scaled out to support high volume requirements.

This guide contains information about developing custom applications through the Web services and Java APIs that Oracle BI Publisher provides for data access, document generation, and delivery.

## Audience

This guide is intended for developers who want to use the Oracle BI Publisher Web services and Java APIs to develop custom applications.

## Documentation Accessibility

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

### Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit
`http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit
`http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

## Related Documentation and Other Resources

See the Oracle Business Intelligence documentation library for a list of related Oracle Business Intelligence documents.

In addition:

- Go to the Oracle Learning Library for Oracle Business Intelligence-related online training resources.
- Go to the Product Information Center support note (Article ID 1338762.1) on My Oracle Support at `https://support.oracle.com`.

## Related Documentation and Other Resources

See the Oracle Business Intelligence documentation library for a list of related Oracle Business Intelligence documents.

In addition, go to the Oracle Learning Library for Oracle Business Intelligence-related online training resources.

### System Requirements and Certification

Refer to the system requirements and certification documentation for information about hardware and software requirements, platforms, databases, and other information. Both of these documents are available on Oracle Technology Network (OTN).

The system requirements document covers information such as hardware and software requirements, minimum disk space and memory requirements, and required system libraries, packages, or patches:

`http://www.oracle.com/technetwork/middleware/ias/downloads/fusion-requirements-100147.html`

The certification document covers supported installation types, platforms, operating systems, databases, JDKs, and third-party products:

`http://www.oracle.com/technetwork/middleware/ias/downloads/fusion-certification-100350.html`

## Conventions

The following text conventions are used in this document:

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

# Part I

# Oracle BI Publisher Web Services

This part explains how to use the Oracle BI Publisher Web services. It includes the following chapters:

1

# Introduction to the BI Publisher Web Services

This chapter provides an introduction to the Oracle BI Publisher Web services.

It includes the following sections:

- Section 1.1, "About BI Publisher Web Services"
- Section 1.2, "Accessing the WSDLs"
- Section 1.3, "About In-Session Methods"
- Section 1.4, "Debugging Web Service Applications"

## 1.1 About BI Publisher Web Services

Oracle BI Publisher Web services provide data types and services. Data types include base data types and complex data types. Oracle BI Publisher Web services also support XML-to-Java data type mappings. For more information on supported data types and mappings, see Chapter 2, "Data Types in Oracle BI Publisher Web Services."

Oracle BI Publisher provides the following public Web services:

- ScheduleService: Provides methods for executing scheduler tasks, such as to schedule report jobs, retrieve report outputs, and manage report history. See Chapter 3, "ScheduleService."
- ReportService: Provides methods to interact with BI Publisher Report object, such as to run reports, get information about reports, define and modify reports, and upload report templates. See Chapter 4, "ReportService."
- SecurityService: Provides methods for security operations, such as authentication, impersonation, login, logout, and account management. See Chapter 5, "SecurityService."
- CatalogService: Provides methods to manage (create, copy, upload, download and delete) report catalog objects, such as folders, reports, data models, style templates, and sub-templates. See Chapter 6, "CatalogService."

Many of the methods in these services are provided in pairs. The first provides a stateless operation that requires login credentials. The second provides an "in session" operation that uses an existing user's session ID, which is obtained through the login() or impersonate() method of the SecurityService. For more information, see Section 1.3, "About In-Session Methods."

## 1.2 Accessing the WSDLs

After you have installed or deployed Oracle BI Publisher, there is a unique URL associated with BI Publisher's web services. Enter the URL for the BI Publisher server and append "/services" as follows:

http://<host>:<port>/xmlpserver/services/

This page provides access to the WSDLs for each of the report services. Use the following WSDLs for the 11g web services:

- ScheduleService - v2/ScheduleService

  WSDL: http://<host>:<port>/xmlpserver/services/v2/ScheduleService?wsdl
- ReportService- v2/ReportService

  WSDL: http://<host>:<port>/xmlpserver/services/v2/ReportService?wsdl
- SecurityService - v2/SecurityService

  WSDL: http://<host>:<port>/xmlpserver/services/v2/SecurityService?wsdl
- CatalogService - v2/CatalogService

  WSDL: http://<host>:<port>/xmlpserver/services/v2/CatalogService?wsdl

## 1.3 About In-Session Methods

Oracle BI Publisher Web services provide many "in session" methods, such as the deliveryServiceInSession() Method, createReportInSession() Method, and copyObjectInSession() Method. In-session methods enable your applications to perform a variety of operations for active user sessions. For this, these methods rely on the bipSessionToken string, which acts as a proprietary token and is generated at user login.

To leverage in-session methods, the user must log in through the SecurityService login() Method or impersonate() Method. Upon successful user authentication from SecurityService, BI Publisher server generates a bipSessionToken string. This bipSessionToken string can be used to perform all in-session operations in this guide.

## 1.4 Debugging Web Service Applications

As a Web services developer, you may need to see the SOAP request messages being used to invoke Web services along with the SOAP responses to those request messages. To do this, you can use the Apache Axis TCP Monitor utility. With this utility, you can monitor the SOAP message flow without requiring you to perform any special configuration, restarting the server, or gaining access to the computer where BI Publisher is running.

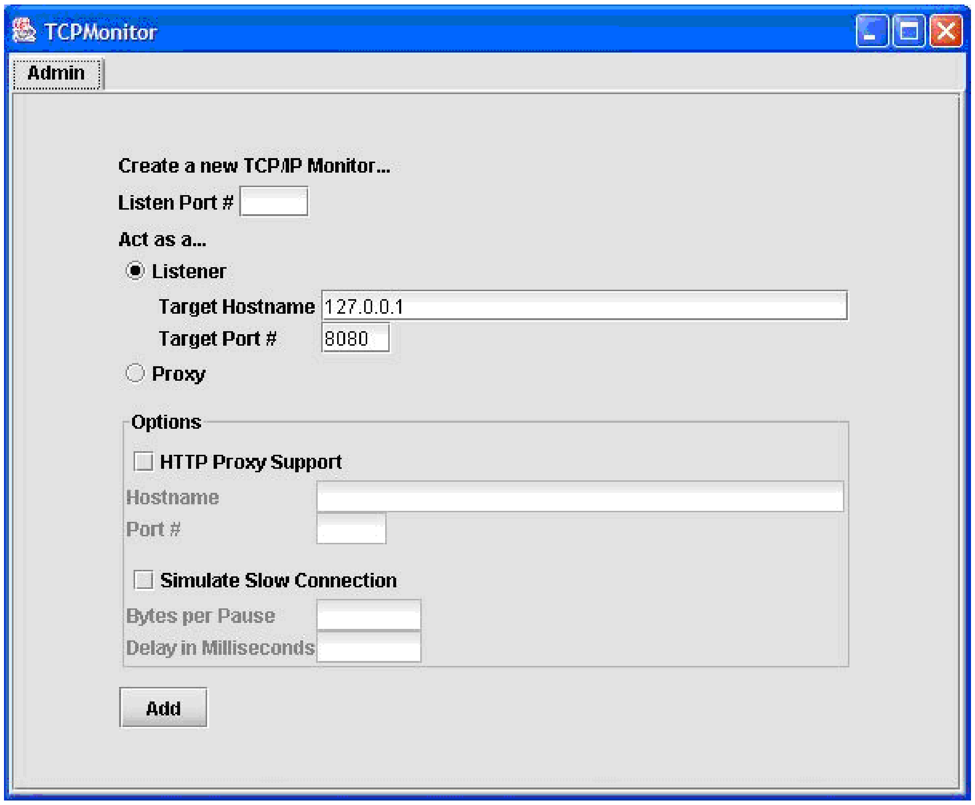
To install TCP Monitor, go to the Apache website (apache.org), and download axis.jar to your computer.

To start TCP Monitor, open a command window and cd to the directory where you downloaded axis.jar. Then, from the command line enter the following:

```
% java -classpath axis.jar org.apache.axis.utils.tcpmon
```

You should see the following screen:

*Figure 1–1 Sample Axis TCP Monitor Window*



Create a new TCP/IP Monitor...

Listen Port #

Act as a...

(●) Listener

Target Hostname 127.0.0.1

Target Port # 8080

( ) Proxy

Options

☐ HTTP Proxy Support

Hostname

Port #

☐ Simulate Slow Connection

Bytes per Pause

Delay in Milliseconds

To configure TCP Monitor:

1. In the Listen Port # field, enter an unused local port on your computer that TCP Monitor will use to listen for messages. For example, `7777`.
2. In the Target Hostname field, enter the host name of the server that is running BI Publisher. For example, `mypublisher.foobar.com.example`.
3. In the Target Port # field, enter the port that is used for the BI Publisher server. For example, `9704`.
4. Click the **Add** button. You will see a new tab with the new monitor listening on the listen port number of the local computer.

Now you can start a browser, SOAP utility, or your application, and run commands against your local computer using the listen port on the local computer (for example, `localmachine:7777/xmlpserver`). TCP Monitor will route those requests to the target host and port. From there, you will see the SOAP request and response messages, which facilitates debugging.

# 2 Data Types in Oracle BI Publisher Web Services

This chapter provides details on the data types that Oracle BI Publisher Web services use or define.

It contains the following sections:

- Section 2.1, "Base Data Types"
- Section 2.2, "XML-to-Java Data Type Mappings"
- Section 2.3, "Complex Data Types"

## 2.1 Base Data Types

Oracle BI Publisher Web services use the following base data types:

***Table 2–1 Base Data Types***

| Base Type | Description | Example |
|---|---|---|
| xsd:boolean | Boolean | true, false |
| xsd:dateTime | Date and Time | 2007-10-26T21:32:52 |
| xsd:int | Integer | 23 |
| xsd:string | String | Home/Shared/HR Reports/Salary Report |
| xsd:base64Binary | 64-bit binary | A document, such as a PDF or HTML report |

## 2.2 XML-to-Java Data Type Mappings

BI Publisher Web Services use document/literal formats. The mapping between Web service XML schema data types and Java data types depends on the SOAP development environment. The following table shows mappings for the Oracle JDeveloper environment:

***Table 2–2 XML-to-Java Data Type Mappings***

| Base Type | Oracle JDeveloper |
|---|---|
| xsd:boolean | java.lang.Boolean |
| xsd:dateTime | java.util.Date |
| xsd:int | java.lang.Integer |
| xsd:string | java.lang.String |

*Table 2–2 (Cont.) XML-to-Java Data Type Mappings*

| Base Type | Oracle JDeveloper |
|---|---|
| xsd:base64Binary | java.lang.Byte |

## 2.3 Complex Data Types

Oracle BI Publisher Web services define and use the following complex data types:

- ArrayOf_xsd_string
- ArrayOfEMailDeliveryOption
- ArrayOfFaxDeliveryOption
- ArrayOfFTPDeliveryOption
- ArrayOfItemData
- ArrayOfJobInfo
- ArrayOfJobOutput
- ArrayOfJobOutputDelivery
- ArrayOfLocalDeliveryOption
- ArrayOfMetaData
- ArrayOfParamNameValue
- ArrayOfPrintDeliveryOption
- ArrayOfString
- ArrayOfTemplateFormatLabelValue
- ArrayOfTemplateFormatLabelValues
- ArrayOfWebDAVDeliveryOption
- BIPDataSource
- CatalogContents
- CatalogObjectInfo
- DeliveryChannels
- DeliveryRequest
- DeliveryServiceDefinition
- EMailDeliveryOption
- FaxDeliveryOption
- FileDataSource
- FTPDeliveryOption
- ItemData
- JDBCDataSource
- JobDetail
- JobFilterProperties
- JobInfo

- JobInfoList
- JobOutput
- JobOutputDelivery
- JobOutputDeliverysList
- JobOutputsList
- JobStatus
- LocalDeliveryOption
- MetaData
- MetaDataList
- ParamNameValue
- ParamNameValues
- PrintDeliveryOption
- ReportDataChunk
- ReportDefinition
- ReportRequest
- ReportResponse
- ScheduleRequest
- TemplateFormatLabelValue
- TemplateFormatsLabelValues
- WebDAVDeliveryOption

### 2.3.1 ArrayOf_xsd_string

Use this data type to hold an array of strings, such as for objects contained in the catalog.

***Table 2–3 Fields Provided by ArrayOf_xsd_string***

| Field | Description |
|---|---|
| String[] item | An array of strings. |

### 2.3.2 ArrayOfEMailDeliveryOption

Use this data type to hold an array of EMailDeliveryOption objects.

***Table 2–4 Fields Provided by ArrayOfEMailDeliveryOption***

| Field | Description |
|---|---|
| EMailDeliveryOption[] item | See Section 2.3.23, "EMailDeliveryOption." |

### 2.3.3 ArrayOfFaxDeliveryOption

Use this data type to hold an array of FaxDeliveryOption objects.

*Table 2–5 Fields Provided by ArrayOfFaxDeliveryOption*

| Field | Description |
|---|---|
| FaxDeliveryOption[] item | See Section 2.3.24, "FaxDeliveryOption." |

### 2.3.4 ArrayOfFTPDeliveryOption

Use this data type to hold an array of FTPDeliveryOption objects.

*Table 2–6 Fields Provided by ArrayOfFTPDeliveryOption*

| Field | Description |
|---|---|
| FTPDeliveryOption[] item | See Section 2.3.26, "FTPDeliveryOption." |

### 2.3.5 ArrayOfItemData

Use this data type to hold an array of objects contained in the catalog.

*Table 2–7 Fields Provided by ArrayOfItemData*

| Field | Description |
|---|---|
| ItemData[] item | See Section 2.3.27, "ItemData." |

### 2.3.6 ArrayOfJobInfo

Use this data type to hold an array of JobInfo objects.

*Table 2–8 Fields Provided by ArrayOfJobInfo*

| Field | Description |
|---|---|
| JobInfo[] item | See Section 2.3.31, "JobInfo." |

### 2.3.7 ArrayOfJobOutput

Use this data type to hold an array of JobOutput objects.

*Table 2–9 Fields Provided by ArrayOfJobOutput*

| Field | Description |
|---|---|
| JobOutput[] item | See Section 2.3.33, "JobOutput." |

### 2.3.8 ArrayOfJobOutputDelivery

Use this data type to hold an array of JobOutputDelivery objects.

*Table 2–10 Fields Provided by ArrayOfJobOutputDelivery*

| Field | Description |
|---|---|
| JobOutputDelivery[] item | See Section 2.3.34, "JobOutputDelivery." |

### 2.3.9 ArrayOfLocalDeliveryOption

Use this data type to hold an array of LocalDeliveryOption objects.

*Table 2–11 Fields Provided by ArrayOfLocalDeliveryOption*

| Field | Description |
|---|---|
| LocalDeliveryOption[] item | See Section 2.3.38, "LocalDeliveryOption." |

### 2.3.10 ArrayOfMetaData

Use this data type to hold an array of MetaData objects.

*Table 2–12 Fields Provided by ArrayOfMetaData*

| Field | Description |
|---|---|
| MetaData[] item | See Section 2.3.39, "MetaData." |

### 2.3.11 ArrayOfParamNameValue

Use this data type to hold an array of ParamNameValue objects (field name-value pairs).

*Table 2–13 Fields Provided by ArrayOfParamNameValue*

| Field | Description |
|---|---|
| ParamNameValue[] item | See Section 2.3.41, "ParamNameValue." |

### 2.3.12 ArrayOfPrintDeliveryOption

Use this data type to hold an array of PrintDeliveryOption objects.

*Table 2–14 Fields Provided by ArrayOfPrintDeliveryOption*

| Field | Description |
|---|---|
| PrintDeliveryOption[] item | See Section 2.3.43, "PrintDeliveryOption." |

### 2.3.13 ArrayOfString

Use this data type to hold an array of strings.

*Table 2–15 Fields Provided by ArrayOfString*

| Field | Description |
|---|---|
| String[] item | An array of strings. |

### 2.3.14 ArrayOfTemplateFormatLabelValue

Use this data type to hold an array of TemplateFormatLabelValue objects (template label-value pairs).

*Table 2–16 Fields Provided by ArrayOfTemplateFormatLabelValue*

| Field | Description |
|---|---|
| TemplateFormatLabelValue[] item | See Section 2.3.49, "TemplateFormatLabelValue." |

### 2.3.15 ArrayOfTemplateFormatLabelValues

Use this data type to hold an array of TemplateFormatsLabelValues objects. ArrayOfTemplateFormatsLabelValues is included in the ReportDefinition complex data type to contain the specific fields to describe the available template formats.

***Table 2–17 Fields Provided by ArrayOfTemplateFormatLabelValue***

| Field | Description |
|---|---|
| TemplateFormatsLabelValues[] item | See Section 2.3.50, "TemplateFormatsLabelValues." |

### 2.3.16 ArrayOfWebDAVDeliveryOption

Use this data type to hold an array of WebDAVDeliveryOption objects.

***Table 2–18 Fields Provided by ArrayOfWebDAVDeliveryOption***

| Field | Description |
|---|---|
| WebDAVDeliveryOption[] item | See Section 2.3.51, "WebDAVDeliveryOption." |

### 2.3.17 BIPDataSource

Use this data type to dynamically specify a data source when using the runReport() method. See Section 4.19, "runReport() Method."

BIPDataSource is used by the ReportRequest complex data type.

The following table lists the fields:

***Table 2–19 Fields Provided by BIPDataSource***

| Field | Description |
|---|---|
| JDBCDataSource JDBCDataSource | Contains the elements to specify a JDBC data source. See Section 2.3.28, "JDBCDataSource." |
| FileDataSource fileDataSource | Contains the elements to specify a file data source. See Section 2.3.25, "FileDataSource." |

### 2.3.18 CatalogContents

Use this data type to hold objects contained in the catalog.

***Table 2–20 Fields Provided by CatalogContents***

| Field | Description |
|---|---|
| ArrayOfItemData catalogContents | See Section 2.3.5, "ArrayOfItemData." |

### 2.3.19 CatalogObjectInfo

Use this data type to return information about an object in the catalog. This data type is returned by the following methods:

- getObjectInfo() Method
- getObjectInfoInSession() Method

***Table 2–21 Fields Provided by CatalogObjectInfo***

| Field | Description |
|---|---|
| String accessPermissions | The permissions that are required to access the catalog object. |
| String[] availableLocales | The array of locales available to the catalog object. |
| long creationDate | The creation date of the catalog object. |
| String description | The description of the catalog object. |
| String displayName | The display name for the catalog object. |
| long lastModifiedDate | The date the catalog object was last modified. |
| String objectAbsolutePath | The absolute path to the catalog object. |
| String objectName | The name of the catalog object. |
| String objectSubType | The subtype of the catalog object.<br>For Folder object, object type and subtype are Folder.<br>For Report object, object type is ReportItem and subtype is xdo.<br>For Data Model object, object Type is ReportItem and subtype is xdm.<br>For Style Template object, object type is StyleTemplate and subtype is RTF or XSL.<br>For Sub Template object, object Type is SubTemplate and subtype is RTF or XSL. |
| String objectType | The type of catalog object. Valid values are:<br>`xdm` (data model)<br>`xdo` (report)<br>`xsb` (sub-template)<br>`xss` (style template) |
| String owner | The owner of the catalog object. |

### 2.3.20 DeliveryChannels

Use this data type to define the specifications to deliver a report to multiple destinations.

This data type is used by the DeliveryRequest and ScheduleRequest complex data types.

The following table lists the fields:

***Table 2–22 Fields Provided by DeliveryChannels***

| Field | Description |
|---|---|
| ArrayOfEMailDeliveryOption emailOptions | See Section 2.3.2, "ArrayOfEMailDeliveryOption." |
| ArrayOfFaxDeliveryOption faxOptions | See Section 2.3.3, "ArrayOfFaxDeliveryOption." |
| ArrayOfFTPDeliveryOption ftpOptions | See Section 2.3.4, "ArrayOfFTPDeliveryOption." |
| ArrayOfLocalDeliveryOption localOptions | See Section 2.3.9, "ArrayOfLocalDeliveryOption." |

*Table 2–22 (Cont.) Fields Provided by DeliveryChannels*

| Field | Description |
|---|---|
| ArrayOfPrintDeliveryOpti on printOptions | See Section 2.3.12, "ArrayOfPrintDeliveryOption." |
| ArrayOfWebDAVDelivery Option webDAVOption | See Section 2.3.16, "ArrayOfWebDAVDeliveryOption." |

### 2.3.21 DeliveryRequest

Use this data type to define the specifications to deliver a report to multiple destinations.

The following table lists the fields:

*Table 2–23 Fields Provided by DeliveryRequest*

| Field | Description |
|---|---|
| String contentType | The content type of the generated document. Possible values are: "text/html;charset=UTF-8" "text/plain;charset=UTF-8" "application/pdf" "application/vnd.ms-powerpoint" "application/vnd.ms-powerpoint" "application/vnd.ms-excel" "application/msword" "application/x-shockwave-flash" "text/xml" "message/rfc822" |
| DeliveryChannels deliveryChannels | See Section 2.3.20, "DeliveryChannels." |
| byte[] documentData | The output document. |
| BIPDataSource dynamicDataSource | See Section 2.3.17, "BIPDataSource." |

### 2.3.22 DeliveryServiceDefinition

Use this data type to return data in the response for the getDeliveryServiceDefinition() Method. Use this method to obtain information about the delivery servers set up for BI Publisher.

The following table lists the fields:

*Table 2–24 Fields Provided by DeliveryServiceDefinition*

| Field | Description |
|---|---|
| ArrayOf_xsd_String EMailServerNames | The list of e-mail server names returned in the ArrayOf_xsd_string data type. See Section 2.3.1, "ArrayOf_xsd_string." |
| ArrayOf_xsd_String FTPServerNames | The list of FTP server names returned in the ArrayOf_xsd_stringdata type. See Section 2.3.1, "ArrayOf_xsd_string." |
| ArrayOf_xsd_String HTTPServerNames | The list of HTTP server names returned in the ArrayOf_xsd_string data type. See Section 2.3.1, "ArrayOf_xsd_string." |
| ArrayOf_xsd_String SFTPServerNames | The list of SFTP server names returned in the ArrayOf_xsd_string data type. See Section 2.3.1, "ArrayOf_xsd_string." |
| ArrayOf_xsd_String defaultServerNames | The list of the default server names for each defined type, returned in the ArrayOf_xsd_string data type. See Section 2.3.1, "ArrayOf_xsd_string." |
| ArrayOf_xsd_String faxServerNames | The list of fax server names returned in the ArrayOf_xsd_string data type. See Section 2.3.1, "ArrayOf_xsd_string." |

*Table 2–24 (Cont.) Fields Provided by DeliveryServiceDefinition*

| Field | Description |
|---|---|
| ArrayOf_xsd_String printerNames | The list of printer names returned in the ArrayOf_xsd_ stringdata type. See Section 2.3.1, "ArrayOf_xsd_string." |
| ArrayOf_xsd_Stringg webDAVServerNames | The list of WebDAV server names returned in the ArrayOf_ xsd_string data type. See Section 2.3.1, "ArrayOf_xsd_string." |

### 2.3.23 EMailDeliveryOption

Use this data type to define the specifications to deliver a report through e-mail.

This data type is used by the ArrayOfEMailDeliveryOption complex data type.

The following table lists the fields:

*Table 2–25 Fields Provided by EMailDeliveryOption*

| Field | Description |
|---|---|
| String emailBCC | The e-mail addresses to receive blind copies of the e-mail. |
| String emailBody | A text string that will appear as the body of the e-mail. |
| String emailCC | The e-mail addresses to receive copies of the e-mail. |
| String emailFrom | Required. The e-mail address that will appear as the From address. If this field is empty, a SOAP fault is thrown with the following message: `Sender (From) email address is not specified.` |
| String emailReplyTo | The e-mail address to appear in the Reply-to field. |
| String emailServerName | The e-mail server name, for example: "Oracle Mail". |
| String emailSubject | The subject line of the e-mail. |
| String emailTo | Required. The addresses to which to send the e-mail. If this field is empty, a SOAP fault is thrown with the following message: `Recipient (TO) email address is not specified.` |

### 2.3.24 FaxDeliveryOption

Use this data type to define the options to set for facsimile (fax) delivery of a report.

This type is used in the ArrayOfFaxDeliveryOption complex data type.

*Table 2–26 Fields Provided by FaxDeliveryOption*

| Field | Description |
|---|---|
| String faxNumber | Required. The number to which to send the fax (for example, `916505069560`). If this field is empty, a SOAP fault is thrown with the following message: `Fax Number is not specified.` |
| String faxServer | Required. The fax server (defined on the BI Publisher Administration page) to which to send the fax (for example, `ipp://mycupsserver:631/printers/fax2`). If this field is empty, a SOAP fault is thrown with the following message: `Fax server is not specified.` |

### 2.3.25 FileDataSource

Use this data type to dynamically create a connection to a file data source when you run a report. You can specify a direct path to a location on your server, or indicate that the file is in the temporary directory

This data type is used in the BIPDataSource complex data type.

The following table lists the fields:

***Table 2–27 Fields Provided by FileDataSource***

| Field | Description |
|---|---|
| String dynamicDataSourcePath | To specify a path to a data source that resides on an available server, specify the full path to the data source and set temporaryDataSource to "false". For example: "D:\BI\OracleBI\xmlp\XMLP\DemoFiles\Balance.xml")"<br><br>If the file is located in the system temporary directory, set temporaryDataSource to true, and specify the file name here. For example: "Balance.xml". |
| boolean temporaryDataSource | Set to "true" when the file data source is in the system temporary directory. Set to "false" when dynamicDataSourcePath specifies the full path. |

### 2.3.26 FTPDeliveryOption

Use this data type to define the options to set for FTP delivery of a report. The FTP server must be set up in the BI Publisher Administration pages first. To set up an FTP server see "Setting Up Delivery Destinations" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Business Intelligence Publisher*.

This type is used in the ArrayOfFTPDeliveryOption complex data type.

***Table 2–28 Fields Provided by FTPDeliveryOption***

| Field | Description |
|---|---|
| String ftpServerName | Required. The FTP Server Name as defined in the BI Publisher Administration page. If this field is empty, a SOAP fault is thrown with the following message:<br><br>`FTP server is not specified.` |
| String ftpUserName | A user name for the FTP server; required only for SFTP. |
| String ftpUserPassword | The password for the user entered; required only for SFTP. |
| String remoteFile | The name to assign the file on the server. For example: report.pdf. |
| boolean sftpOption | Set to "true" to use secure FTP (SFTP). Set to "false" to use FTP. |

### 2.3.27 ItemData

Use this data type to return object metadata of an object stored in the catalog.

***Table 2–29 Fields Provided by ItemData***

| Field | Description |
|---|---|
| String absolutePath | The path to the report object. For example: /HR Manager/ HR Reports/Employee Listing.xdo |
| dateTime creationDate | The creation date of the report object. |

*Table 2–29 (Cont.) Fields Provided by ItemData*

| Field | Description |
|---|---|
| String displayName | The display name for the report object. For example: Employee Listing |
| String fileName | The file name for the report object (for example, Employee Listing.xdo). |
| dateTime lastModified | The last modified date for the report object. |
| String lastModifier | The user name of the last person to modify the report. |
| String owner | The user name of the owner of the report. |
| String parentAbsolutePath | The absolute path of the parent folder. For example, "/HR Manager/HR Reports" is the parentAbsolutePath for the report having the absolute path "/HR Manager/HR Reports/Employee Listing.xdo". |
| String type | The item type. Possible values are: "report" or "folder". |

### 2.3.28 JDBCDataSource

Use this data type to dynamically create a connection to a JDBC data source when you run a report.

This data type is used by the BIPDataSource complex data type.

The following table lists the fields:

*Table 2–30 Fields Provided by JDBCDataSource*

| Field | Description |
|---|---|
| String JDBCDriverClass | The JDBC driver class for the data source (for example, oracle.jdbc.OracleDriver). |
| String JDBCDriverType | The driver type as defined in the BI Publisher data source definition page (for example, Oracle 9i/10g/11g). |
| String JDBCPassword | The password for the data source as defined in the BI Publisher data source definition page. |
| String JDBCURL | The connection string for the data source (for example, jdbc:oracle:thin:@mydatabase.foobar.com.example:1521: orcl). |
| String JDBCUserName | The user name for the data source as defined in the BI Publisher data source definition page. |
| String dataSourceName | The Data Source Name assigned to the data source in the BI Publisher data source definition page (for example, Oracle). |

### 2.3.29 JobDetail

The collection of information about a job request.

*Table 2–31 Fields Provided by JobDetail*

| Field | Description |
|---|---|
| boolean bursting | The value `true` indicates the bursting option is enabled for the job. |
| String burstingParameters | The parameters for the bursting engine. |
| dateTime created | The date the job was created. |

***Table 2–31 (Cont.) Fields Provided by JobDetail***

| Field | Description |
|---|---|
| String dataLocator | When the storageType is DB, dataLocator is the primary key for retrieving the data from database. |
| boolean deleted | The value `true` indicates the job was deleted. |
| String deliveryDescription | The description for the job delivery. |
| String deliveryParameters | The parameters for the delivery channels. |
| dateTime endDate | The date the job is scheduled to end. |
| int instanceId | The numeric identification for the job instance. |
| String issuer | The issuer of the job. |
| String jobGroup | The group to which the job belongs. |
| int jobId | The numeric identification for the job instance. |
| int jobSetId | Inactive. Do not use. |
| String jobType | The type of job. |
| dateTime lastUpdated | The date and time the job was last updated. |
| String locale | The locale to which the job belongs. |
| String notificationParameters | The notification parameters for the job. |
| String owner | The name of the job owner. |
| int parentJobId | The numeric identifier for the parent of the job. |
| boolean public | Whether the job is viewable by other users (true) or not (false). |
| String reportParameters | The parameters for the report. |
| String reportUrl | The URL to the job output. |
| String runType | The type of job (either Single or Recurring). |
| String scheduleContext | The context for the job schedule when external applications submit the report. |
| String scheduleDescription | The description of the job schedule. |
| String scheduleParameters | The parameters of the job schedule. |
| String scheduleSource | The source of the job schedule, which is used with the scheduleContext to provide information on external applications. |
| dateTime startDate | The date the job is scheduled to start or started. |
| String status | The status of the job. |
| String statusDetail | The details of the job status. |
| String storageType | The storage type for the job. Supported value is `DB`. |
| String userDescription | A user-assigned description for the job. |
| String userJobName | The user-assigned named for the job. |
| boolean xmlDataAvailable | Whether XML data for the job is available (true) or not (false). |
| boolean xmlDataCompressed | A value of true indicates the XML data for the job is or will be compressed. |
| String xmlDataContentType | The content type of the XML data. |

*Table 2–31 (Cont.) Fields Provided by JobDetail*

| Field | Description |
|---|---|
| String xschurl | The URL of the report. |

### 2.3.30 JobFilterProperties

This data type is used by the following methods to define the filter criteria for a specific report job:

- getAllScheduledReport() Method
- getAllScheduledReportHistory() Method
- getAllScheduledReportHistoryInSession() Method
- getAllScheduledReportInSession() Method

The fields in this data type identify the specific report job or jobs about which you want information returned.

*Table 2–32 Fields Provided by JobFilterProperties*

| Field | Description |
|---|---|
| String endTime | The time the job is scheduled to end or ended. |
| String endTimeOperator | The operator for endTime. Valid values are "Equals or Earlier Than", "Equals" or "Earlier Than". |
| long jobID | The numeric identification assigned by BI Publisher for the job request. |
| String jobName | The user-assigned job name. Valid values are "Contains" or "Equals". |
| String jobNameOperator | The operator of the jobName. |
| String owner | The name of the job's owner. |
| String ownerOperator | The operator for owner. Valid values are "Contains" or "Equals". |
| String reportName | The user-assigned named for the job. |
| String reportNameOperator | The operator for reportName. Valid values are "Contains" or "Equals". |
| String scope | The scope of the job. Valid values are "All", "Private" or "Public". |
| String startTime | The time the job is scheduled to start or started. |
| String startTimeOperator | The operator for startTime. Valid values are "Equals or Later Than", "Equals" or "Later Than". |
| String status | The current status for the job. |

### 2.3.31 JobInfo

The collection of information about a job request.

*Table 2–33 Fields Provided by JobInfo*

| Field | Description |
|---|---|
| boolean burstingJob | A value of `true` indicates the bursting option is enabled. |
| dateTime created | The date the job was created. |
| boolean deleted | The value of `true` indicates the job was deleted. |

***Table 2–33 (Cont.) Fields Provided by JobInfo***

| Field | Description |
|---|---|
| dateTime endDate | The date the job is scheduled to end or ended. |
| long instanceId | The numeric identification for the scheduled job request. |
| long jobID | The numeric identification assigned by BI Publisher to the job request. |
| String jobType | The type of job. |
| dateTime lastUpdated | The date and time the job was last updated. |
| String owner | The owner of the job. |
| long parentJobId | The numeric identification for the parent of the scheduled job request. |
| boolean public | True indicates the report is a member of a report set. In the current implementation this will always return false. |
| String reportUrl | The report absolute path URL, for example: /HR Manager/Employee Reports/Employee Salary Report.xdo. |
| dateTime startDate | The date the job is scheduled to start or started. |
| String status | The status of the scheduled job request. Valid values are: Canceled", "Done", "Scheduled", "Suspended", or "Unknown". |
| String statusDetail | Additional details for the status. |
| String userJobName | The username of the user submitting the job request. |

## 2.3.32 JobInfoList

Use this data type to return an array of JobInfo objects.

***Table 2–34 Fields Provided by JobInfoList***

| Field | Description |
|---|---|
| ArrayOfJobInfo jobInfoList | An array of JobInfo objects. See Section 2.3.6, "ArrayOfJobInfo.". |

## 2.3.33 JobOutput

Use this data type to return a description of job outputs.

***Table 2–35 Fields Provided by JobOutput***

| Field | Description |
|---|---|
| String burstKey | The key used to split the data for each bursted job. |
| dateTime created | The date the report job was created. |
| boolean deleted | The value "true" indicates the job output was deleted. |
| boolean documentDataAvailable | True indicates that the user selected the "Save Output" option when the report was scheduled. |
| boolean documentDataCompressed | True indicates that the document data is compressed. |

*Table 2–35 (Cont.) Fields Provided by JobOutput*

| Field | Description |
|---|---|
| String documentDataConentType | The content type of the generated document. Possible values are: "text/html;charset=UTF-8" "text/plain;charset=UTF-8" "application/pdf" "application/vnd.ms-powerpoint" "application/vnd.ms-excel" "application/msword" "application/x-shockwave-flash" "text/xml" "message/rfc822" |
| long jobID | The identification number assigned to the job by BI Publisher. |
| String jobName | The user-assigned job name. |
| dateTime lastUpdated | The date and time the job was last updated. |
| long outputID | The identification of the report in history. One scheduled JobID can be associated with multiple outputIDs or historyIDs. This is because one scheduled report can be executed or republished multiple times. |
| String outputName | The name assigned to the output. |
| long parentOutputId | The output ID of the parent request. |
| String status | Valid values are: "Completed", "Error", "Running", "Scheduled", "Suspended" and "Unknown" |
| String statusDetail | Detailed status information from the BI Publisher server. |

## 2.3.34 JobOutputDelivery

Use this data type to return a description of job output.

*Table 2–36 Fields Provided by JobOutputDelivery*

| Field | Description |
|---|---|
| dateTime created | The date the report job was created. |
| long deliveryID | The primary key to identify the job delivery. |
| byte[] deliveryParameters | The parameters for the deliverychannels. |
| dateTime lastUpdated | The date and time the job was last updated. |
| long outputID | The identification of the report in history. One scheduled JobID can be associated with multiple outputIDs or historyIDs. This is because one scheduled report can be executed or republished multiple times. |
| long parentDeliveryID | The delivery ID of the parent request. |
| String status | Valid values are: "Completed", "Error", "Running", "Scheduled", "Suspended" and "Unknown" |
| String statusDetail | Detailed status information from the BI Publisher server. |

### 2.3.35 JobOutputDeliverysList

This data type is a wrapper class implemented to return an array of JobOutputDelivery objects.

***Table 2–37 Fields Provided by JobOutputDeliverysList***

| Field | Description |
|---|---|
| ArrayOfJobOutputDelivery jobOutputDeliveryList | See Section 2.3.8, "ArrayOfJobOutputDelivery.". |

### 2.3.36 JobOutputsList

Use this data type to return an array of JobOutput objects.

***Table 2–38 Fields Provided by JobOutputsList***

| Field | Description |
|---|---|
| ArrayOfJobOutput jobOutputList | See Section 2.3.7, "ArrayOfJobOutput." |

### 2.3.37 JobStatus

Use this data type to return the status of a job request.

***Table 2–39 Fields Provided by JobStatus***

| Field | Description |
|---|---|
| String jobID | The numeric identification assigned by BI Publisher to the job request. |
| String jobStatus | The current status of the job. |
| String message | Details on the job status. |

### 2.3.38 LocalDeliveryOption

The options to set for delivery of a report to the BI Publisher repository.

This type is used in the ArrayOfLocalDeliveryOption complex data type.

***Table 2–40 Fields Provided by LocalDeliveryOption***

| Field | Description |
|---|---|
| String destination | Required. The file path to the BI Publisher repository on the local server. This field supports concatenation of the directory path and the file path. If empty, a SOAP fault is thrown with the following message: `Local destination is not specified.` |

### 2.3.39 MetaData

Use the data type to set the name-value pair for a MetaData object.

***Table 2–41 Fields Provided by MetaData***

| Field | Description |
|---|---|
| String metaDataName | The name of the metadata. |
| String metaDataValue | The metadata content. |

### 2.3.40 MetaDataList

Use this data type to return a list of MetaData objects.

***Table 2–42 Fields Provided by MetaDataList***

| Field | Description |
|---|---|
| ArrayOfMetaData metaDataList | See Section 2.3.10, "ArrayOfMetaData." |

### 2.3.41 ParamNameValue

This data type describes parameters defined for a reports and templates in BI Publisher.

The ParamNameValue data type is used in the ArrayOfParamNameValue, which is included in the ReportRequest, ReportDefinition, and JobInfo data types. ParamNameValue is also returned by the getTemplateParameters() Method and getReportParameters() Method.

***Table 2–43 Fields Provided by ParamNameValue***

| Field | Description |
|---|---|
| String UIType | The type of parameter as defined in the BI Publisher data model user interface.<br>Valid values include:<br>■ Date<br>■ Hidden<br>■ Menu<br>■ Search<br>■ Text |
| String dataType | Valid values include:<br>■ Boolean<br>■ Date<br>■ Float<br>■ Integer<br>■ String |
| String dateFormatString | If UIType is "Date", this specifies the Date Format String. The date format string must be a Java date format (for example, MM-DD-YYYY). |
| String dateFrom | If UIType is "Date", this specifies the begin value of the date. |
| String dateTo | If UIType is "Date", this specifies the end value of the date. |
| String defaultValue | Specifies the default value of the parameter. |
| String fieldSize | For parameter types "Text" and "Date", specifies the text field size for the parameter. |
| String label | For all parameter types except "Hidden", specifies the display label for the parameter. |
| ArrayOfString lovLabels | If the parameter type is "Menu", specifies the values displayed in the list of values to the user. |
| boolean multiValuesAllowed | `True` indicates that a parameter may contain multiple values. |

*Table 2–43 (Cont.) Fields Provided by ParamNameValue*

| Field | Description |
|---|---|
| String name | The parameter name. |
| boolean refreshParamOnChange | For parameter types "Text" and "Menu", a value of `true` for this parameter indicates that other defined parameters should be refreshed when a selection is made for this parameter. |
| boolean selectAll | For parameter type "Menu", a value of `true` indicates that all values can be selected for the LOV. |
| boolean templateParam | A value of `true` indicates the parameter is defined in the RTF template. |
| boolean useNullForAll | For parameter type "Menu", a value of `true` indicates that a null will be passed if all values are selected for the parameter. |
| ArrayOfString values | See Section 2.3.13, "ArrayOfString." |

### 2.3.42 ParamNameValues

This data type is a wrapper class that returns an array of ParamNameValue objects.

*Table 2–44 Fields Provided by ParamNameValues*

| Field | Description |
|---|---|
| ArrayOfParamNameValue listOfParamNameValues | See Section 2.3.11, "ArrayOfParamNameValue." |

### 2.3.43 PrintDeliveryOption

Use this data type to set the options for printer delivery of a report.

This type is used in the ArrayOfPrintDeliveryOption complex data type.

*Table 2–45 Fields Provided by PrintDeliveryOption*

| Field | Description |
|---|---|
| String printNumberOfCopy | The number of copies to print. |
| String printOrientation | Valid values are "portrait" or "landscape". |
| String printRange | The range of pages to print. For example, "3" will print page 3; "2-5" will print pages 2-5; "1,3-5" will print pages 1 and pages 3-5. |
| String printSide | Valid values are:<br>■ "d_single_sided" for single-sided<br>■ "d_double_sided_l" for duplex/long edge<br>■ "d_double_sided_s" for tumble/short edge<br>If the parameter is not specified, single-sided is used. |
| String printTray | Valid values are:<br>■ "t1" for "Tray 1"<br>■ "t2" for "Tray 2"<br>■ "t3" for "Tray 3"<br>If not specified, the printer default is used. |

*Table 2–45 (Cont.) Fields Provided by PrintDeliveryOption*

| Field | Description |
|---|---|
| String printerName | Required. The printer Server Name (as set up in the BI Publisher Administration page) to which to send the report. If empty, a SOAP fault is thrown with the following message:<br>`Print server is not specified.` |

## 2.3.44 ReportDataChunk

Use this data type to handle large report data sets, or to upload and download report data in smaller data chunks.

*Table 2–46 Fields Provided by ReportDataChunk*

| Field | Description |
|---|---|
| byte[] reportDataChunk | Byte[] array representing binary report data transported between the BI Publisher client and server. |
| String reportDataFileID | The identifier for the data file of the report on the BI Publisher server. |
| long reportDataOffset | The offset value for the location of the previously downloaded report data file. |

## 2.3.45 ReportDefinition

Use this data type to define a report object. This is the object returned by the getReportDefinition() Method.

*Table 2–47 Fields Provided by ReportDefinition*

| Field | Description |
|---|---|
| boolean autoRun | True indicates that the report property Auto Run is turned on. |
| boolean cacheDocument | True indicates that the report property Enable document cache is turned on. |
| boolean controledByExtApp | Whether the report definition is controlled by an external application (true) or not (false). |
| String dataModelURL | The .xdm location from where to get the Data Model definition. |
| String defaultOutputFormat | The default output format.<br>Valid values include:<br>■ `csv` (CSV)<br>■ `data` (Data)<br>■ `eText` (eText template)<br>■ `excel` (Microsoft Excel)<br>■ `excel2000` (Microsoft Excel 2000)<br>■ `flash` (Adobe Flash)<br>■ `html` (HTML)<br>■ `mhtml` (MIME HTML)<br>■ `pdf` (Adobe PDF)<br>■ `ppt` (Microsoft PowerPoint)<br>■ `rtf` (Rich Text Format) |
| String defaultTemplateId | The default template identified for the report. |

*Table 2–47 (Cont.) Fields Provided by ReportDefinition*

| Field | Description |
|---|---|
| boolean diagnostics | True indicates that diagnostics have been turned on for the report. |
| String ESSJobName | The ESS job name. |
| String ESSPackageName | The ESS package name. Used in conjunction with ESSJobName. |
| ArrayOfTemplateFormatsLabelValues listOfTemplateFormatsLabelValues | Passes the list of template format labels through the ArrayOfTemplateFormatsLabelValues data type. See Section 2.3.15, "ArrayOfTemplateFormatLabelValues." |
| boolean onLine | True indicates the property "Run report online" is turned on for the report. |
| boolean openLinkInNewWindow | True indicates the property "Open Links in New Window" is turned on for the report. |
| integer parameterColumns | The value of the report property "Parameters per line." |
| ArrayOfString parameterNames | Passes the parameter names defined for the report through the ArrayOfString data type. See Section 2.3.13, "ArrayOfString." |
| String reportDefnTitle | Inactive. Do not use. |
| String reportDescription | The user-assigned description of the report. |
| String reportName | The user-assigned name for the report. |
| ArrayOfParamNameValue reportParameterNameValues | Passes the report name-value pairs through the ArrayOfParamNameValue data type. See Section 2.3.11, "ArrayOfParamNameValue." |
| String reportType | Inactive. Do not use. |
| boolean showControls | True indicates the report property "Show controls" has been turned on. |
| boolean showReportLinks | True indicates the report property "Show report links" has been turned on. |
| ArrayOfString templateIds | Passes the layout names of the report templates through the ArrayOfString data type. See Section 2.3.13, "ArrayOfString." |

### 2.3.46 ReportRequest

Use this data type to define the settings needed to run a report. Note that allowable values for `attributeFormat` will vary according to the type of template used (for example, PDF templates can only generate PDF output.)

*Table 2–48 Fields Provided by ReportRequest*

| Field | Description |
|---|---|
| String attributeCalendar | The formatting calendar to use for the report request. Valid values are: "Gregorian", "Arabic Hijrah", "English Hijrah", "Japanese Imperial", "Thai Buddha", and "ROC Official". |
| String attributeFormat | The output format of the requested report. See Section 2.3.46.1, "Values for attributeFormat." |
| String attributeLocale | The locale selection for the report. Example: fr-FR |
| String attributeTemplate | The template to apply to the report. For example: Employeelisting.rtf. |

*Table 2–48 (Cont.) Fields Provided by ReportRequest*

| Field | Description |
|---|---|
| String attributeTimeZone | Specifies the time zone to use for the request, using a supported Java time zone ID. For example: "America/Los_Angeles". |
| boolean byPassCache | True indicates to bypass document cache. |
| BIPDataSource dynamicDataSource | If the data source for the report is not defined, you can dynamically define it. See Section 2.3.17, "BIPDataSource." |
| boolean flattenXML | True indicates that the XML is to be flattened. This flag is used for the Analyzer for Microsoft Excel because Excel requires XML data type to be flattened. |
| ParamNameValues parameterNameValues | The parameter name-value pairs to be used in the submission of this report request, passed through the ParamNameValues data type. See Section 2.3.42, "ParamNameValues." |
| String reportAbsolutePath | The absolute path to the report in the BI Publisher repository. For example: /HR Manager/HR Reports/Employee Listing.xdo. |
| byte[] reportData | If you are providing the data directly for the report use this element to pass the data. |
| String reportOutputPath | Specifies the output path for the generated report. |
| String reportRawData | If raw XML data is used for the report, this element contains the XML data. |
| integer sizeOfDataChunkDownload | If you set flattenXML to true, or if you do not want to chunk the data, set this parameter to -1 to return all data back to the client. |
| MetaDataList XDOPropertyList | See Section 2.3.40, "MetaDataList." |

### 2.3.46.1 Values for attributeFormat

The following table lists the valid values for attributeFormat and the template types that support each output format type.

*Table 2–49 Valid Values for attributeFormat*

| Output Format | Value to Enter for attributeFormat Field | Template Types That Can Generate This Output Format |
|---|---|---|
| Interactive | Interactive | BI Publisher |
| HTML | html | BI Publisher, RTF, XSL Stylesheet (FO) |
| PDF | pdf | BI Publisher, RTF, PDF, Flash, XSL Stylesheet (FO) |
| RTF | rtf | BI Publisher, RTF, XSL Stylesheet (FO) |
| Excel (mhtml) | excel | BI Publisher, RTF, Excel, XSL Stylesheet (FO) |
| Excel (html) | excel2000 | BI Publisher, RTF, Excel, XSL Stylesheet (FO) |
| Excel (*.xslx) | xslx | BI Publisher, RTF, XSL Stylesheet (FO) |
| PowerPoint (mhtml) | ppt | BI Publisher, RTF, XSL Stylesheet (FO) |
| PowerPoint (.*pptx) | pptx | BI Publisher, RTF, XSL Stylesheet (FO) |
| MHTML | mhtml | BI Publisher, RTF, Flash, XSL Stylesheet (FO) |
| PDF/A | pdfa | BI Publisher, RTF, XSL Stylesheet (FO) |
| PDF/X | pdfx | BI Publisher, RTF, XSL Stylesheet (FO) |
| Zipped PDFs | pdfz | BI Publisher, RTF, PDF, XSL Stylesheet (FO) |

*Table 2–49 (Cont.) Valid Values for attributeFormat*

| Output Format | Value to Enter for attributeFormat Field | Template Types That Can Generate This Output Format |
|---|---|---|
| FO Formatted XML | xslfo | BI Publisher, RTF, XSL Stylesheet (FO) |
| Data (XML) | xml | BI Publisher, RTF, PDF, Excel, Flash, XSL Stylesheet (FO), Etext, XSL Stylesheet (HTML XML/Text) |
| Data (CSV) | csv | BI Publisher, RTF, PDF, Excel, Flash, XSL Stylesheet (FO), XSL Stylesheet (HTML XML/Text), Etext |
| XML | xml | XSL Stylesheet (HTML XML/Text) |
| Text | text | XSL Stylesheet (HTML XML/Text), Etext |
| Flash | flash | Flash |

## 2.3.47 ReportResponse

ReportResponse is the object returned from runReport.

*Table 2–50 Fields Provided by ReportResponse*

| Field | Description |
|---|---|
| MetaDataList metaDataList | See Section 2.3.40, "MetaDataList." |
| byte[] reportBytes | The report binary data output. |
| String reportContentType | The report content type. Possible values include: "text/html;charset=UTF-8" "text/plain;charset=UTF-8" "application/pdf" "application/vnd.ms-powerpoint" "application/vnd.ms-excel" "application/msword" "application/x-shockwave-flash" "text/xml" "message/rfc822" |
| String reportFileID | The numeric identification for the report file. |
| String reportLocale | The locale selected for the report (for example, `fr_FR`). |

## 2.3.48 ScheduleRequest

The options to schedule a report.

*Table 2–51 Fields Provided by ScheduleRequest*

| Field | Description |
|---|---|
| boolean bookBindingOutputOption | Whether the book binding output is enabled (true) or not (false). |
| String dataModelUrl | The location of the .xdm file from which to obtain the Data Model definition. |
| DeliveryChannels deliveryChannels | See Section 2.3.20, "DeliveryChannels." |
| String endDate | The end date of the schedule. |
| String jobLocale | The locale to use for the scheduled requests. Example: fr-FR |

*Table 2–51 (Cont.) Fields Provided by ScheduleRequest*

| Field | Description |
|---|---|
| String jobTZ | The time zone to use for the scheduled requests. |
| boolean mergeOutputOption | Whether the merge output option is enabled (true) or not (false). |
| String notificationPassword | The HTTP notification server password when scheduling notification through an HTTP server. |
| String notificationServer | The name of the HTTP server used for notification. |
| String notificationTo | E-mail addresses to which to send notifications. |
| String notificationUserName | The user name for the HTTP server used for notification. |
| boolean notifyHttpWhenFailed | True indicates to send a notification when the job request fails. |
| boolean notifyHttpWhenSuccess | True indicates to send a notification when the job request succeeds. |
| boolean notifyHttpWhenWarning | True indicates to send a notification when the job completes with a warning. |
| boolean notifyWhenFailed | True indicates to send a notification when the job request fails. |
| boolean notifyWhenSuccess | True indicates to send a notification when the job request succeeds. |
| boolean notifyWhenWarning | True indicates to send a notification when the job completes with a warning. |
| String recurrenceExpression | The expression that defines a recurring schedule. |
| String recurrenceExpressionType | The type of expression defined for a recurring schedule. Valid value is `cron`. |
| integer repeatCount | The number of times to repeat the schedule. For the recursive scheduling of a report, startDate must not be null, and repeatCount repeatInterval should be greater than 0 for any meaningful schedule. The endDate can be null. |
| integer repeatInterval | The interval between two scheduled jobs in seconds. |
| ReportRequest reportRequest | Information about the request included through the ReportRequest data type. See Section 2.3.46, "ReportRequest." |
| boolean saveDataOption | True indicates that the report data from the scheduled request run will be saved. |
| boolean scheduleBurstingOption | True indicates that the scheduled requests will be burst. |
| boolean schedulePublicOption | True indicates that the scheduled requests are to be made public. |
| String startDate | The date on which the schedule starts. |
| boolean useUTF8Option | True indicates that the Use UTF8 option is enabled. |
| String userJobDesc | The user-entered description for the scheduled job. |
| String userJobName | The user-entered name for the scheduled job. |

## 2.3.49 TemplateFormatLabelValue

To specify the template format labels and values for a report. TemplateFormatLabelValue is included in the ArrayOfTemplateFormatLabelValue complex data type. The elements that comprise TemplateFormatLabelValue are as follows:

***Table 2–52 Fields Provided by TemplateFormatLabelValue***

| Field | Description |
|---|---|
| String templateFormatLabel | The label that displays for a template format. Examples include: "HTML" "PDF" "Excel" |
| String templateFormatValue | The template format value that corresponds to the label. Examples include: "html" "pdf" "excel" |

### 2.3.50 TemplateFormatsLabelValues

Provides detailed information about template formats stored in the BI Publisher repository. TemplateFormatsLabelValues is included in the ArrayOfTemplateFormatLabelValues complex data type.

***Table 2–53 Fields Provided by TemplateFormatsLabelValues***

| Field | Description |
|---|---|
| boolean active | Whether the template is active (true) or not (false). |
| boolean applyStyleTemplate | Whether to apply the style template (true) or not (false). |
| boolean default | Whether the template is the default template (true) or not (false). |
| ArrayOfTemplateFormatLabelValue listOfTemplateFormatLabelValue | Contains the TemplateFormatLabelValue label-value pairs. See Section 2.3.14, "ArrayOfTemplateFormatLabelValue." |
| ArrayOfString templateAvailableLocales | The available locale options defined for a template passed in the ArrayOfString data type. See Section 2.3.13, "ArrayOfString." |
| String templateBaseLocale | The base locale options defined for a template. |
| String templateDefaultLocale | The default locale options defined for a template. |
| String templateID | The name assigned to the template in BI Publisher, for example: "Employee Listing". |
| String templateType | The type of template, for example: "rtf" or "pdf". |
| String templateURL | The template file name in the BI Publisher repository, for example: "Employee Listing.rtf". |
| boolean viewOnline | Whether the template can be viewed online (true) or not (false). |

### 2.3.51 WebDAVDeliveryOption

The options to set for Web-based Distributed Authoring and Versioning (WebDAV) delivery of a report.

This type is used in the ArrayOfWebDAVDeliveryOption complex data type.

***Table 2–54 Fields Provided by WebDAVDeliveryOption***

| Field | Description |
|---|---|
| String deliveryAuthType | Authentication type. Valid values are: None, Basic, Digest |

*Table 2–54 (Cont.) Fields Provided by WebDAVDeliveryOption*

| Field | Description |
|---|---|
| String password | If a proxy server has been set up, the password required to access the proxy server. |
| String remoteFilePath | The path to directory on the remote server to which to deliver the report file. |
| String server | Required. The WebDAV server name (for example, `myserver`). If empty, a SOAP fault is thrown with the following message: `WebDAV server is not specified.` |
| String userName | If a proxy server has been set up, the user name required to access the proxy server. |

# 3 ScheduleService

This chapter provides details on the ScheduleService methods that you can use to interact with the BI Publisher scheduler. This includes methods for scheduling report jobs, retrieving report outputs, and managing report histories.

This chapter contains the following sections:

- Section 3.26, "getScheduledReportOutputInfoInSession() Method"
- Section 3.27, "getXMLData() Method"
- Section 3.28, "getXMLDataInSession() Method"
- Section 3.29, "purgeJobHistory() Method"
- Section 3.30, "purgeJobHistoryInSession() Method"
- Section 3.31, "resendScheduledReport() Method"
- Section 3.32, "resendScheduledReportInSession() Method"
- Section 3.33, "resumeSchedule() Method"
- Section 3.34, "resumeScheduleInSession() Method"
- Section 3.35, "scheduleReport() Method"
- Section 3.36, "scheduleReportInSession() Method"
- Section 3.37, "suspendSchedule() Method"
- Section 3.38, "suspendScheduleInSession() Method"

**Note:** For information on debugging applications built with BI Publisher Web services, see Section 1.4, "Debugging Web Service Applications."

## 3.1 cancelSchedule() Method

Use the cancelSchedule() method to cancel a currently running scheduled job.

### Signature

boolean cancelSchedule(String jobInstanceID, String userID, String password);

***Table 3–1 Parameters for cancelSchedule() Method***

| Parameter | Description |
|---|---|
| String jobInstanceID | The ID assigned to the instance of the job to be canceled. The jobInstanceID is a string of integers. |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 3.2 cancelScheduleInSession() Method

Cancels the schedule associated with the bipSessionToken string for a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

String cancelScheduleInSession(String jobInstanceID, String bipSessionToken);

*Table 3–2 Parameters for cancelScheduleInSession() Method*

| Parameter | Description |
|---|---|
| String jobInstanceID | The ID assigned to the instance of the job that generated the output. The jobInstanceID is a string of integers. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 3.3 deleteJobHistory() Method

Use the deleteJobHistory() method to perform a "soft" delete the historical information about a report job, as opposed to the purgeJobHistory() method, which performs a "hard" (permanent) deletion. The deleteJobHistory() method must precede the purgeJobHistory() method.

boolean deleteJobHistory(String instanceJobID, String userID, String password);

### Signature

*Table 3–3 Parameters for deleteJobHistory() Method*

| Parameter | Description |
|---|---|
| String instanceJobID | The ID assigned to the instance of the job that generated the output. The instanceJobID is a string of integers. |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 3.4 deleteJobHistoryInSession() Method

Deletes the job history associated with the bipSessionToken string for a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

boolean deleteJobHistoryInSession(String jobInstanceID, String bipSessionToken);

*Table 3–4 Parameters for deleteJobHistoryInSession() Method*

| Parameter | Description |
|---|---|
| String jobInstanceID | The ID assigned to the instance of the job that generated the output. The jobInstanceID is a string of integers. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 3.5 deleteSchedule() Method

Use the deleteSchedule() method to delete a scheduled job from the scheduler queue.

### Signature
boolean deleteSchedule(String jobInstanceID, String userID, String password);

*Table 3–5 Parameters for deleteSchedule() Method*

| Parameter | Description |
|---|---|
| String jobInstanceID | The ID assigned to the instance of the job to be deleted. The jobInstanceID is a string of integers. |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 3.6 deleteScheduleInSession() Method

Deletes the schedule associated with the bipSessionToken string for a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature
boolean deleteScheduleInSession(String jobInstanceID, String bipSessionToken);

*Table 3–6 Parameters for deleteScheduleInSession() Method*

| Parameter | Description |
|---|---|
| String jobInstanceID | The ID assigned to the instance of the job that generated the output. The jobInstanceID is a string of integers. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 3.7 deliveryService() Method

Use the deliveryService() method to deliver a document from source to destination through the specified delivery channel.

### Signature
String deliveryService(DeliveryRequest deliveryRequest, String userID, String password);

*Table 3–7 Parameters for deliveryService() Method*

| Parameter | Description |
|---|---|
| DeliveryRequest deliveryRequest | The DeliveryRequest object. See Section 2.3.21, "DeliveryRequest." |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 3.8 deliveryServiceInSession() Method

Delivers a document associated with the deliveryRequest and bipSessionToken string for a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

String deliveryServiceInSession(String jobInstanceID, String bipSessionToken);

***Table 3–8 Parameters for deliveryServiceInSession() Method***

| Parameter | Description |
|---|---|
| DeliveryRequest deliveryRequest | The DeliveryRequest object. See Section 2.3.21, "DeliveryRequest." |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 3.9 downloadDocumentData() Method

Saves a report document into the local temporary directory of the BI Publisher server, and returns the fileID of the user for later downloads. This implementation is for performance concern in case that report data size is significant.

### Signature

String downloadDocumentData(String JobOutputID, String userID, String password);

***Table 3–9 Parameters for downloadDocumentData() Method***

| Parameter | Description |
|---|---|
| String jobOutputID | The ID assigned to the output. The jobOutputID is a string of integers. |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 3.10 downloadDocumentDataInSession() Method

Downloads the document report associated with the jobInstanceID and bipSessionToken string for a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

String downloadDocumentDataInSession(String jobInstanceID, String bipSessionToken);

***Table 3–10 Parameters for downloadDocumentDataInSession() Method***

| Parameter | Description |
|---|---|
| String jobInstanceID | The ID assigned to the instance of the job that generated the output. The jobInstanceID is a string of integers. |

***Table 3–10 (Cont.) Parameters for downloadDocumentDataInSession() Method***

| Parameter | Description |
|---|---|
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 3.11 downloadXMLData() Method

Returns XML data used to generate a report document. It returns raw data in XML format.

### Signature

downloadXMLData(String jobInstanceID, String userID, String password);

***Table 3–11 Parameters for downloadXMLData() Method***

| Parameter | Description |
|---|---|
| String jobInstanceID | The ID assigned to the XML data. The jobInstanceID is a string of integers. |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 3.12 downloadXMLDataInSession() Method

Downloads the XML data for a document report associated with the bipSessionToken string for a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

String downloadXMLDataInSession(String jobInstanceID, String bipSessionToken);

***Table 3–12 Parameters for downloadXMLDataInSession() Method***

| Parameter | Description |
|---|---|
| String jobInstanceID | The ID assigned to the instance of the job that generated the output. The jobInstanceID is a string of integers. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 3.13 getAllScheduledReport() Method

Use the getAllScheduledReportInfo() method to return information about all scheduled report jobs that match filter criteria passed through the JobFilterProperties object.

### Signature

JobInfosList getAllScheduledReportInfo(JobFilterProperties filter, int beginIdx, String userID, String password);

*Table 3–13 Parameters for getAllScheduledReport() Method*

| Parameter | Description |
|---|---|
| JobFilterProperties filter | The JobFilterProperties object specifies the specific criteria for the report jobs you want to return information about. See Section 2.3.30, "JobFilterProperties." |
| int beginIdx | The starting point of the index (default is `1`). |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 3.14 getAllScheduledReportHistory() Method

Use the getAllScheduledReportHistory() method to return information about all scheduled report histories that match filter criteria passed through the JobFilterProperties object.

### Signature

JobInfosList getAllScheduledReportHistory(JobFilterProperties filter, int beginIdx, String userID, String password);

*Table 3–14 Parameters for getAllScheduledReportHistory() Method*

| Parameter | Description |
|---|---|
| JobFilterProperties filter | The JobFilterProperties object specifies the specific criteria for the report jobs you want to return information about. See Section 2.3.30, "JobFilterProperties." |
| int beginIdx | The starting point of the index (default is `1`). |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 3.15 getAllScheduledReportHistoryInSession() Method

Use the getAllScheduledReportHistoryInSession() method to return information about all scheduled report histories that match filter criteria passed through the JobFilterProperties object and that are based on the bipSessionToken string for a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

JobInfosList getAllScheduledReportHistoryInSession(JobFilterProperties filter, int beginIdx, String bipSessionToken);

***Table 3–15 Parameters for getAllScheduledReportHistoryInSession() Method***

| Parameter | Description |
|---|---|
| JobFilterProperties filter | The JobFilterProperties object specifies the specific criteria for the report jobs you want to return information about. See Section 2.3.30, "JobFilterProperties." |
| int beginIdx | The starting point of the index (default is `1`). |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 3.16 getAllScheduledReportInSession() Method

Use the getAllScheduledReportInSession() method to return information about all scheduled reports that match filter criteria passed through the JobFilterProperties object and that are based on the bipSessionToken string for a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

JobInfosList getAllScheduledReportInSession(JobFilterProperties filter, int beginIdx, String bipSessionToken);

***Table 3–16 Parameters for getAllScheduledReportInSession() Method***

| Parameter | Description |
|---|---|
| JobFilterProperties filter | The JobFilterProperties object specifies the specific criteria for the report jobs you want to return information about. See Section 2.3.30, "JobFilterProperties." |
| int beginIdx | The starting point of the index (default is `1`). |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 3.17 getDeliveryServiceDefinition() Method

Use the getDeliveryServiceDefinition() method to get the delivery service definition for a given userID and password. See deliveryService() Method.

### Signature

DeliveryServiceDefinition getDeliveryServiceDefinition(String userID, String password);

***Table 3–17 Parameters for getDeliveryServiceDefinition() Method***

| Parameter | Description |
|---|---|
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 3.18 getDeliveryServiceDefinitionInSession() Method

Use the getDeliveryServiceDefinitionInSession() method to get the delivery service definition based on the bipSessionToken of a given user. See deliveryService() Method.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

DeliveryServiceDefinition getDeliveryServiceDefinitionInSession(String bipSessionToken);

*Table 3–18 Parameters for getDeliveryServiceDefinitionInSession() Method*

| Parameter | Description |
|---|---|
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 3.19 getDocumentData() Method

Use the getDocumentData() method to return a document generated by a BI Publisher scheduled job. The getDocumentData() method takes the jobOutputID as a parameter to return the appropriate document.

To get the jobOutputID: The scheduleReport() method returns jobID. Call getAllScheduledReportHistory() using this parent jobID to get a list of its children (schedule jobs). Use the appropriate child jobID to call getScheduledOutputInfo() to get a list of the outputIDs for this particular job. Use the outputID of the desired job as input (JobOutputID parameter) to the getDocumentData() method to retrieve the document data.

Note that the getDocumentData() method returns the byte[] of a report document, while the downloadDocumentData() Method saves the report document onto BI Publisher server as a local file. The latter method returns the file ID, enabling the user to download the report document later through the Delivery Service. This is for performance concerns in cases where a report document size is large.

### Signature

byte[] getDocumentData(String jobOutputID, String userID, String password);

*Table 3–19 Parameters for getDocumentData() Method*

| Parameter | Description |
|---|---|
| String jobOutputID | Job output assigned to the output. The output ID is a string of integers. |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 3.20 getDocumentDataInSession() Method

Returns the byte[] of a report document based on the JobOutputID and bipSessionToken of a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

byte[] getDocumentDataInSession(String jobOutputID, String bipSessionToken);

***Table 3–20 Parameters for getDocumentDataInSession() Method***

| Parameter | Description |
|---|---|
| String jobOutputID | The ID assigned to the output. The jobOutputID is a string of integers. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 3.21 getScheduledJobInfo() Method

Use the getScheduledJobInfo() method to return a JobDetail object that provides the details about a submitted job, including report parameters and other properties. This method retrieves all information stored in the database for a given jobInstanceID, userID, and password.

### Signature

JobInfo getScheduledJobInfo(int jobInstanceID, String userID, String password);

***Table 3–21 Parameters for getScheduledJobInfo() Method***

| Parameter | Description |
|---|---|
| int jobInstanceID | The ID of the job for which to return job information. |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 3.22 getScheduledJobInfoInSession() Method

Returns a JobDetail object that provides the details for the job that's associated with a given jobInstanceID and the bipSessionToken of a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

JobDetail getScheduledJobInfoInSession(String jobInstanceID, String bipSessionToken);

***Table 3–22 Parameters for getScheduledJobInfoInSession() Method***

| Parameter | Description |
|---|---|
| String jobInstanceID | The ID assigned to the job instance. The jobInstanceID is a string of integers. |

*Table 3–22 (Cont.) Parameters for getScheduledJobInfoInSession() Method*

| Parameter | Description |
|---|---|
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 3.23 getScheduledReportDeliveryInfo() Method

Use the getScheduledReportdeliveryInfo() method to retrieve information about the delivery of a scheduled job output. For each scheduled Job, it could have multiple outputIDs. For each outputID, there could be multiple delivery info. See Section 2.3.35, "JobOutputDeliverysList."

### Signature

JobOutputDeliverysList getScheduledReportDeliveryInfo(String jobOutputID, String userID, String password);

*Table 3–23 Parameters for getScheduledReportDeliveryInfo() Method*

| Parameter | Description |
|---|---|
| String jobOutputID | The ID assigned to the output of the job for which you want information. The jobOutputID is a string of integers. |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 3.24 getScheduledReportDeliveryInfoInSession() Method

Returns a JobOutputDeliverysList object that provides the details for a given jobOutputID and the bipSessionToken of a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

JobOutputDeliverysList getScheduledReportDeliveryInfoInSession(String jobOutputID, String bipSessionToken);

*Table 3–24 Parameters for getScheduledReportDeliveryInfoInSession() Method*

| Parameter | Description |
|---|---|
| String jobOutputID | The ID assigned to the job output. The jobOutputID is a string of integers. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 3.25 getScheduledReportOutputInfo() Method

Use the getScheduledReportOutputInfo() method to return information about a specific scheduled report output.

### Signature

JobOutputsList getScheduledReportOutputInfo(String jobInstanceID, String userID, String password);

*Table 3–25 Parameters for getScheduledReportOutputInfo() Method*

| Parameter | Description |
|---|---|
| String jobInstanceID | The ID of the job for which to return job information. |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 3.26 getScheduledReportOutputInfoInSession() Method

Use the getScheduledReportOutputInfo() method to return information about a specific scheduled report output based on its jobInstanceID and the bipSessionToken of a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

JobOutputsList getScheduledReportOutputInfoInSession(String jobInstanceID, String bipTokenSession);

*Table 3–26 Parameters for getScheduledReportOutputInfoInSession() Method*

| Parameter | Description |
|---|---|
| String jobInstanceID | The ID of the job for which to return job information. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 3.27 getXMLData() Method

Use the getXMLData() method to return, in XML format, the data document generated by a BI Publisher scheduled job. You can use the JobOutputID returned from the scheduleReport() method to retrieve the generated XML document.

Note, the getXMLData() method returns the byte[] of a report document, while the downloadXMLData() Method saves the XML data on the BI Publisher server as a local file. The latter method returns the file ID, enabling the user to download the XML-based document later through the Delivery Service. This is for performance concerns in cases where a report document size is quite large.

### Signature

byte[] getXMLData(String JobInstanceID, String userID, String password);

*Table 3–27 Parameters for getXMLData() Method*

| Parameter | Description |
|---|---|
| String JobInstanceID | The ID assigned to the instance of the job that generated the output. The JobInstanceID is a string of integers. |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 3.28 getXMLDataInSession() Method

Returns the byte[] of XML data based on the jobInstanceID and bipSessionToken of a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

byte[] getXMLDataInSession(String jobInstanceID, String bipSessionToken);

*Table 3–28 Parameters for getXMLDataInSession() Method*

| Parameter | Description |
|---|---|
| String jobInstanceID | The ID assigned to the job instance. The jobInstanceID is a string of integers. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 3.29 purgeJobHistory() Method

Use the purgeHistory() method to perform a "hard" delete of the historical information about a report job. That is, to permanently purge the information from the database.

You must precede the purgeJobHistory() method with the deleteJobHistory() method, otherwise the following SOAP fault is thrown:

```
purgeJobHistory failed due to job is not deleted. You have to
delete JobHistory first prior to purge.
```

### Signature

boolean purgeJobHistory(String instanceJobID, String userID, String password);

*Table 3–29 Parameters for purgeJobHistory() Method*

| Parameter | Description |
|---|---|
| String instanceJobID | The ID assigned to the instance of the job that generated the output. The instanceJobID is a string of integers. |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 3.30 purgeJobHistoryInSession() Method

Permanently purges the job history from the database for the given instanceJobID and bipSessionToken of the given user. This action must be preceded by a deleteJobHistoryInSession.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

boolean purgeJobHistoryInSession(String instanceJobID, String bipSessionToken);

***Table 3–30 Parameters for purgeJobHistoryInSession() Method***

| Parameter | Description |
|---|---|
| String instanceJobID | The ID assigned to the job instance. The instanceJobID is a string of integers. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 3.31 resendScheduledReport() Method

Use the resendScheduledReport() method to resend a previously-defined scheduled report. The resend action is respective to the outputJobID. There's no need to define any delivery channels options, as the previously-defined delivery parameters are used to perform the resend action.

### Signature

boolean resendScheduledReport(String outputJobID, String userID, String password);

***Table 3–31 Parameters for resendScheduledReport() Method***

| Parameter | Description |
|---|---|
| String outputJobID | The ID of the scheduled job to resend. |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 3.32 resendScheduledReportInSession() Method

Resends a previously-defined scheduled report based on its associated outputJobID and the bipSessionToken of a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

boolean resendScheduledReportInSession(String outputJobID, String bipSessionToken);

*Table 3–32 Parameters for resendScheduledReportInSession() Method*

| Parameter | Description |
|---|---|
| String outputJobID | The ID of the scheduled job to resend. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 3.33 resumeSchedule() Method

Use the resumeSchedule() method to resume a schedule job that has been suspended.

### Signature

boolean resumeSchedule(String jobInstanceID, String userID, String password);

*Table 3–33 Parameters for resumeSchedule() Method*

| Parameter | Description |
|---|---|
| String jobInstanceID | The ID assigned to the instance of the job to be resumed. The jobInstanceID is a string of integers. |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 3.34 resumeScheduleInSession() Method

Resumes a scheduled job that was previously suspended based on its jobInstanceID and the bipSessionToken of a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

boolean resumeScheduleInSession(String outputJobID, String bipSessionToken);

*Table 3–34 Parameters for resumeScheduleInSession() Method*

| Parameter | Description |
|---|---|
| String jobInstanceID | The ID assigned to the instance of the job to be resumed. The jobInstanceID is a string of integers. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 3.35 scheduleReport() Method

Use the scheduleReport() method to schedule the reports that are in the BI Publisher catalog. You can submit and run the reports immediately or create a job to schedule the reports to run. When you schedule reports you can also deliver reports to any of the delivery destinations that are set up in your BI Publisher Enterprise Server instance.

The method returns a JobID of the scheduled job.

### Signature

String scheduleReport(ScheduleRequest scheduleRequest, String userID, String password);

***Table 3–35 Parameters for scheduleReport() Method***

| Parameter | Description |
|---|---|
| ScheduleRequest scheduleRequest | Specifies a ScheduleRequest object for the report that you want to run. See Section 2.3.48, "ScheduleRequest." |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 3.36 scheduleReportInSession() Method

Schedules a report based on the schedule request, delivery channel, and bipSessionToken of a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

String scheduleReportInSession(ScheduleRequest scheduleRequest, DeliveryChannel deliveryChannel, String bipSessionToken);

***Table 3–36 Parameters for scheduleReportInSession() Method***

| Parameter | Description |
|---|---|
| ScheduleRequest scheduleRequest | Specifies a ScheduleRequest object for the report that you want to run. See Section 2.3.48, "ScheduleRequest.". |
| DeliveryChannel deliveryChannel | Specifies the delivery channels through which the report will be delivered. See Section 2.3.20, "DeliveryChannels." |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 3.37 suspendSchedule() Method

Use the suspendSchedule() method to suspend a schedule job.

### Signature

boolean suspendschedule(String jobInstanceID, String userID, String password);

***Table 3–37 Parameters for suspendSchedule() Method***

| Parameter | Description |
|---|---|
| String jobInstanceID | The ID assigned to the instance of the job to be suspended. The jobInstanceID is a string of integers. |

*Table 3–37 (Cont.) Parameters for suspendSchedule() Method*

| Parameter | Description |
|---|---|
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 3.38 suspendScheduleInSession() Method

Suspends a scheduled report based on its associated jobInstanceID and the bipSessionToken of a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

boolean suspendScheduleInSession(String jobInstanceID, String bipSessionToken);

*Table 3–38 Parameters for suspendScheduleInSession() Method*

| Parameter | Description |
|---|---|
| String jobInstanceID | The ID assigned to the instance of the job to be suspended. The jobInstanceID is a string of integers. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

# 4 ReportService

This chapter provides details on the ReportService methods that you can use to interact with the BI Publisher Report object. This includes methods for designing and defining reports, report templates, run-time operations, and parameters.

This chapter includes the following sections:

- Section 4.26, "updateXLIFFForReportInSession() Method"
- Section 4.27, "uploadReportDataChunk() Method"
- Section 4.28, "uploadReportDataChunkInSession() Method"
- Section 4.29, "uploadTemplateForReport() Method"
- Section 4.30, "uploadTemplateForReportInSession() Method"
- Section 4.31, "uploadXLIFFForReport() Method"
- Section 4.32, "uploadXLIFFForReportInSession() Method"

**Note:** For information on debugging applications built with BI Publisher Web services, see Section 1.4, "Debugging Web Service Applications."

## 4.1 createReport() Method

Use the createReport() method to create a report in the BI Publisher catalog. The method enables you to set the path to the data model and supply template files and translation (XLIFF) files to the report definition.

### Signature

String createReport(String reportName, String folderAbsolutePathURL, String dataModelURL, String templateFileName, byte[] templateData, String XLIFFFileName, byte[] XLIFFData, boolean updateFlag, String userID, String password);

*Table 4–1 Parameters for createReport() Method*

| Parameter | Description |
|---|---|
| String reportName | The report name to create with the suffix ".xdo". For example, "myreport.xdo". |
| String folderAbsolutePathURL | The path to the folder in which to place the created report. For example: `xmlp/Reports/financials` |
| String dataModelURL | The path to the data model that will be used as the data source for this report. For example: `xmlp/Reports/financials/Data Models/my data model.xdm` |
| String templateFileName | The file name of the template to add the report definition. |
| byte[] templateData | The template file. |
| String XLIFFFileName | The file name of the XLIFF file. You must append the locale to the XLIFF file name as follows:<br>`template_<language code>_<country code>.xlf`<br>where<br>`<language_code>` is the two-letter ISO 639 language code<br>`<country_code>` is the two-letter ISO 639 language code<br>For example: template_en_us.xlf |
| byte[] XLIFFData | The XLIFF file. |
| boolean updateFlag | If `true`, overwrites existing report. If `false`, throws error if the report exists. |

***Table 4–1 (Cont.) Parameters for createReport() Method***

| **Parameter** | **Description** |
|---|---|
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 4.2 createReportInSession() Method

Use the createReport() method to create a report in the BI Publisher catalog based on the bipSessionToken of a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

String createReport(String reportName, String folderAbsolutePathURL, String dataModelURL, String templateFileName, byte[] templateData, String XLIFFFileName, byte[] XLIFFData, boolean updateFlag, String bipSessionToken);

***Table 4–2 Parameters for createReportInSession() Method***

| **Parameter** | **Description** |
|---|---|
| String reportName | The report name to create with the suffix ".xdo". For example, "myreport.xdo". |
| String folderAbsolutePathURL | The path to the folder in which to place the created report. For example: xmlp/Reports/financials |
| String dataModelURL | The path to the data model that will be used as the data source for this report. For example: xmlp/Reports/financials/Data Models/my data model.xdm |
| String templateFileName | The file name of the template to add the report definition. |
| byte[] templateData, | The template file. |
| String XLIFFFileName | The file name of the XLIFF file. You must append the locale to the XLIFF file name as follows: `template_<language code>_<country code>.xlf` where `<language_code>` is the two-letter ISO 639 language code `<country_code>` is the two-letter ISO 639 language code For example: template_en_us.xlf |
| byte[] XLIFFData | The XLIFF file. |
| boolean updateFlag | If `true`, overwrites existing report. If `false`, throws error if the report exists. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 4.3 downloadReportDataChunk() Method

Use downloadReportDataChunk() method to download very large documents, so that the caller calls this method multiple times until all document content is downloaded. Each call to this method downloads one chunk of the document, where the beginIdx parameter refers to the file download starting point.

See Section 2.3.44, "ReportDataChunk."

**Note:** When using uploadReportDataChunk() or downloadReportDataChunk() in a clustered environment, you must set the System Temporary Directory to be a shared directory accessible to all servers within the cluster. To set the System Temporary Directory:

1. Sign in to BI Publisher with Administrator privileges.
2. Click the **Administration** link.
3. Under **System Maintenance**, click **Server Configuration**.
4. Under **General Properties** in the **System Temporary Directory** property, enter the absolute path to a directory accessible to all servers in the cluster.

   For example, the directory can exist under `${xdo.server.config.dir}/temp` but you must enter the absolute path, such as `/net/subfoldera/scratch/subfolderb/11gcat/temp`

Repeat this procedure for all servers in the cluster, entering the same value for **System Temporary Directory**.

### Signature

ReportDataChunk downloadReportDataChunk(String fileID, int beginIdx, int size);

***Table 4–3 Parameters for downloadReportDataChunk() Method***

| Parameter | Description |
|---|---|
| String fileID | fileID is returned inside ReportResponse, which is returned when calling runReport() Method. |
| int beginIdx | The starting point of the index (default is `1`). |
| int size | The size of the file to download (in kilobytes). |

## 4.4 downloadReportDataChunkInSession() Method

Use downloadReportDataChunk() method to download very large documents using the bipSessionToken of a given user. The caller calls this method multiple times until all document content is downloaded. Each call to this method downloads one chunk of the document, where the beginIdx parameter refers to the file download starting point.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

ReportDataChunk downloadReportDataChunkInSession(String fileID, int beginIdx, int size, String bipSessionToken);

***Table 4–4 Parameters for downloadReportDataChunkInSession() Method***

| Parameter | Description |
|---|---|
| String fileID | fileID is returned inside ReportRequest, which is returned when calling runReport() Method. |
| int beginIdx | The starting point of the index (default is `1`). |
| int size | The size of the file to download (in kilobytes). |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 4.5 getReportDefinition() Method

Use the getReportDefinition() method to get information about a report, such as the default template, output type, and a list of template IDs. With the list of template IDs, you can generate a report with a template other than the default.

See Section 2.3.45, "ReportDefinition."

### Signature

ReportDefinition getReportDefinition(String reportAbsolutePath, String userID, String password);

***Table 4–5 Parameters for getReportDefinition() Method***

| Parameter | Description |
|---|---|
| String reportAbsolutePath | The path to the report for which to retrieve the report definition. For example: /HR Manager/Employee Reports/Employee Listing.xdo |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 4.6 getReportDefinitionInSession() Method

Use the getReportDefinitionInSession() method to get information about a report using the bipSessionToken of a given user. This method returns report details such as the default template, output type, and a list of template IDs. With the list of template IDs, you can generate a report with a template other than the default.

See Section 2.3.45, "ReportDefinition."

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

ReportDefinition getReportDefinitionInSession(String reportAbsolutePath, String bipSessionToken);

***Table 4–6 Parameters for getReportDefinitionInSession() Method***

| **Parameter** | **Description** |
|---|---|
| String reportAbsolutePath | The path to the report for which to retrieve the report definition. For example: /HR Manager/Employee Reports/Employee Listing.xdo |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 4.7 getReportParameters() Method

Use the getReportParameters() method to get an array of report parameters and their default values. With the list of parameters, you can set parameter values before running or scheduling a report.

See Section 2.3.42, "ParamNameValues."

### Signature

ParamNameValues getReportParameters(ReportRequest reportRequest, String userID, String password);

***Table 4–7 Parameters for getReportParameters() Method***

| **Parameter** | **Description** |
|---|---|
| ReportRequest reportRequest | See Section 2.3.46, "ReportRequest." |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 4.8 getReportParametersInSession() Method

Use the getReportParameters() method to get an array of report parameters and their default values based on the bipSessionToken of a given user. With the list of parameters, you can set parameter values before running or scheduling a report.

See Section 2.3.42, "ParamNameValues."

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

ParamNameValues getReportParametersInSession(ReportRequest reportRequest, String bipSessionToken);

***Table 4–8 Parameters for getReportParametersInSession() Method***

| **Parameter** | **Description** |
|---|---|
| ReportRequest reportRequest | See Section 2.3.46, "ReportRequest." |

***Table 4–8 (Cont.) Parameters for getReportParametersInSession() Method***

| Parameter | Description |
|---|---|
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 4.9 getReportSampleData() Method

Use the getReportSampleData() method to retrieve the sample data file stored with the report data model.

### Signature

byte[] getReportSampleData(String reportAbsolutePath, String userID, String password);

***Table 4–9 Parameters for getReportSampleData() Method***

| Parameter | Description |
|---|---|
| String reportAbsolutePath | The path to the report for which to retrieve the report data model sample data. For example: /HR Manager/Employee Reports/Employee Listing.xdo |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 4.10 getReportSampleDataInSession() Method

Use the getReportSampleDataInSession() method to retrieve the sample data file stored with the report data model based on the bipSessionToken of a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

byte[] getReportSampleData(String reportAbsolutePath, String bipSessionToken);

***Table 4–10 Parameters for getReportSampleDataInSession() Method***

| Parameter | Description |
|---|---|
| String reportAbsolutePath | The path to the report for which to retrieve the report data model sample data. For example: /HR Manager/Employee Reports/Employee Listing.xdo |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 4.11 getTemplate() Method

Use getTemplate() method to retrieve a template from a report definition in the BI Publisher catalog.

#### Signature

byte[] getTemplate(String reportAbsolutePath, String templateID, String locale, String userID, String password);

***Table 4–11 Parameters for getTemplate() Method***

| Parameter | Description |
|---|---|
| String reportAbsolutePath | The path to the report to which the template is associated. For example: /HR Manager/Employee Reports/Employee Listing.xdo |
| String templateID | The ID of the template (for example, `Chart Layout`). |
| String locale | The locale of the template to retrieve (for example, `en_US`). |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 4.12 getTemplateInSession() Method

Use getTemplateInSession() method to retrieve a template from a report definition in the BI Publisher catalog based on the bipTokenSession of a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

#### Signature

byte[] getTemplateInSession(String reportAbsolutePath, String templateID, String locale, String bipSessionToken);

***Table 4–12 Parameters for getTemplateInSession() Method***

| Parameter | Description |
|---|---|
| String reportAbsolutePath | The path to the report to which the template is associated. For example: /HR Manager/Employee Reports/Employee Listing.xdo |
| String templateID | The ID of the template (for example, `Chart Layout`). |
| String locale | The locale of the template to retrieve (for example, `en_US`). |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 4.13 getTemplateParameters() Method

Use the getReportParameters() method to get the parameters for a template.

See Section 2.3.41, "ParamNameValue."

#### Signature

ParamNameValue[] getTemplateParameters(String reportAbsolutePath, String templateID, String userID, String password);

***Table 4–13 Parameters for getTemplateParameters() Method***

| Parameter | Description |
|---|---|
| String reportAbsolutePath | The path to the report for which to retrieve the report definition. For example: /HR Manager/Employee Reports/Employee Listing.xdo |
| String templateID | The ID assigned to the template, for example: "Chart Layout". |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 4.14 getTemplateParameterInSession() Method

Use the getReportParametersInSession() method to get the parameters for a template.

See Section 2.3.41, "ParamNameValue."

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

ParamNameValue[] getTemplateParameters(String reportAbsolutePath, String templateID, String bipSessionToken);

***Table 4–14 Parameters for getTemplateParameterInSession() Method***

| Parameter | Description |
|---|---|
| String reportAbsolutePath | The path to the report for which to retrieve the report definition. For example: /HR Manager/Employee Reports/Employee Listing.xdo |
| String templateID | The ID assigned to the template, for example: "Chart Layout". |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 4.15 getXDOSchema() Method

Use getXDOSchema() method to retrieve the XDO schema for a report definition in the BI Publisher catalog.

### Signature

byte[] getXDOSchema(String reportAbsolutePath, String locale, String userID, String password);

***Table 4–15 Parameters for getXDOSchema() Method***

| Parameter | Description |
|---|---|
| String reportAbsolutePath | The path to the report from which to retrieve the XDO schema. For example: /HR Manager/Employee Reports/Employee Listing.xdo |
| String locale | The locale of the template to retrieve (for example, `en_US`). |

*Table 4–15 (Cont.) Parameters for getXDOSchema() Method*

| Parameter | Description |
|---|---|
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 4.16 getXDOSchemaInSession() Method

Use getXDOSchemaInSession() method to retrieve the XDO schema for a report definition in the BI Publisher catalog based on a bipSessionToken of a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

byte[] getXDOSchemaInSession(String reportAbsolutePath, String locale, String bipSessionToken);

*Table 4–16 Parameters for getXDOSchemaInSession() Method*

| Parameter | Description |
|---|---|
| String reportAbsolutePath | The path to the report from which to retrieve the XDO schema. For example: /HR Manager/Employee Reports/Employee Listing.xdo |
| String locale | The locale of the template to retrieve (for example, `en_US`). |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 4.17 removeTemplateForReport() Method

Use removeTemplateForReport() method to remove a template from a report definition in the BI Publisher catalog.

### Signature

boolean removeTemplateForReport(String reportAbsolutePath, String templateFileName, String userID, String password);

*Table 4–17 Parameters for removeTemplateForReport() Method*

| Parameter | Description |
|---|---|
| String reportAbsolutePath | The path to the report from which to remove the template. For example: /HR Manager/Employee Reports/Employee Listing.xdo |
| String templateFileName | The file name of the template to remove. |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 4.18 removeTemplateForReportInSession() Method

Use removeTemplateForReportInSession() method to remove a template from a report definition in the BI Publisher catalog based on the bipSessionToken of a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

boolean removeTemplateForReportInSession(String reportAbsolutePath, String templateFileName, String bipSessionToken);

***Table 4–18 Parameters for removeTemplateForReportInSession() Method***

| Parameter | Description |
|---|---|
| String reportAbsolutePath | The path to the report from which to remove the template. For example: /HR Manager/Employee Reports/Employee Listing.xdo |
| String templateFileName | The file name of the template to remove. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 4.19 runReport() Method

Use the runReport() method to send a request to the BI Publisher server to run a specific report.

See Section 2.3.46, "ReportRequest" and Section 2.3.47, "ReportResponse."

### Signature

ReportResponse runReport(ReportRequest reportRequest, String userID, String password);

***Table 4–19 Parameters for runReport() Method***

| Parameter | Description |
|---|---|
| ReportRequest reportRequest | See Section 2.3.46, "ReportRequest." |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 4.20 runReportInSession() Method

Use the runReportInSession() method to send a request to the BI Publisher server to run a specific report based on the bipSessionToken of a given user.

See Section 2.3.46, "ReportRequest" and Section 2.3.47, "ReportResponse."

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

ReportResponse runReportInSession(ReportRequest reportRequest, String bipSessionToken);

*Table 4–20 Parameters for runReportInSession() Method*

| Parameter | Description |
|---|---|
| ReportRequest reportRequest | See Section 2.3.46, "ReportRequest." |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 4.21 updateReportDefinition() Method

Use the updateReportDefinition() to update attributes of the report definition file (.xdo) and write the file back to the BI Publisher catalog.

### Signature

boolean updateReportDefinition(String reportAbsPath, ReportDefinition newReportDefn, String userID, String password);

*Table 4–21 Parameters for updateReportDefinition() Method*

| Parameter | Description |
|---|---|
| String reportAbsPath | The path to the report for which to update the report definition. For example: /HR Manager/Employee Reports/Employee Listing.xdo |
| ReportDefinition newReportDefn | See Section 2.3.45, "ReportDefinition." |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 4.22 updateReportDefinitionInSession() Method

Use the updateReportDefinitionInSession() to update attributes of the report definition file (.xdo) based on the bipTokenSession of a given user, and then to write the file back to the BI Publisher catalog.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

boolean updateReportDefinitionInSession(String reportAbsPath, ReportDefinition newReportDefn, String bipSessionToken);

*Table 4–22 Parameters for updateReportDefinitionInSession() Method*

| Parameter | Description |
|---|---|
| String reportAbsPath | The path to the report for which to update the report definition. For example: /HR Manager/Employee Reports/Employee Listing.xdo |
| ReportDefinition newReportDefn | See Section 2.3.45, "ReportDefinition." |

*Table 4–22 (Cont.) Parameters for updateReportDefinitionInSession() Method*

| Parameter | Description |
|---|---|
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 4.23 updateTemplateForReport() Method

Use updateTemplateForReport() method to update a template for a specific report in the BI Publisher catalog.

### Signature

boolean updateTemplateForReport(String reportAbsolutePath, String templateName, String locale, byte[] templateData, String userID, String password);

*Table 4–23 Parameters for updateTemplateForReport() Method*

| Parameter | Description |
|---|---|
| String reportAbsolutePath | The path to the report that contains the template to update. For example: /HR Manager/Employee Reports/Employee Listing.xdo |
| String templateName | The name of the template to update (for example, `Chart Layout`). |
| String locale | The locale of the template to update (for example, `en_US`). |
| byte[] templateData | The template file. |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 4.24 updateTemplateForReportInSession() Method

Use updateTemplateForReportInSession() method to update a template for a specific report in the BI Publisher catalog based on the bipSessionToken of a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

boolean updateTemplateForReportInSession(String reportAbsolutePath, String templateName, String locale, byte[] templateData, String bipSessionToken);

*Table 4–24 Parameters for updateTemplateForReportInSession() Method*

| Parameter | Description |
|---|---|
| String reportAbsolutePath | The path to the report that contains the template to update. For example: /HR Manager/Employee Reports/Employee Listing.xdo |
| String templateName | The name of the template to update (for example, `Chart Layout`). |
| String locale | The locale of the template to update (for example, `en_US`). |

*Table 4–24 (Cont.) Parameters for updateTemplateForReportInSession() Method*

| Parameter | Description |
|---|---|
| byte[] templateData | The template file. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 4.25 updateXLIFFForReport() Method

Use updateXLIFFForReport() method to update a translation file (XLIFF) associated with a layout definition in the BI Publisher catalog.

### Signature

boolean updateXLIFFForReport(String reportAbsolutePath, byte[] xliffData, String layoutFileName, String locale, String userID, String password);

*Table 4–25 Parameters for updateXLIFFForReport() Method*

| Parameter | Description |
|---|---|
| String reportAbsolutePath | The path to the report to that contains the XLIFF file to update. For example: /HR Manager/Employee Reports/Employee Listing.xdo |
| byte[] xliffData | The XLIFF fie to upload. |
| String layoutFileName | The file name of the layout for which the XLIFF file is to be updated. For example: employee_listing.rtf. |
| String locale | The locale to assign to the XLIFF (for example, `en_US`). |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 4.26 updateXLIFFForReportInSession() Method

Use updateXLIFFForReportInSession() method to update a translation file (XLIFF) associated with a layout definition in the BI Publisher catalog based on the bipSessionToken of a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

boolean updateXLIFFForReportInSession(String reportAbsolutePath, byte[] xliffData, String layoutFileName, String locale, String bipSessionToken);

*Table 4–26 Parameters for updateXLIFFForReportInSession() Method*

| Parameter | Description |
|---|---|
| String reportAbsolutePath | The path to the report to that contains the XLIFF file to update. For example: /HR Manager/Employee Reports/Employee Listing.xdo |
| byte[] xliffData | The XLIFF fie to upload. |

*Table 4–26 (Cont.) Parameters for updateXLIFFForReportInSession() Method*

| Parameter | Description |
|---|---|
| String layoutFileName | The file name of the layout for which the XLIFF file is to be updated. For example: employee_listing.rtf. |
| String locale | The locale to assign to the XLIFF (for example, `en_US`). |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 4.27 uploadReportDataChunk() Method

Use uploadReportDataChunk() method to upload a report data chunk.

**Note:** When using uploadReportDataChunk() or downloadReportDataChunk() in a clustered environment, you must set the System Temporary Directory to be a shared directory accessible to all servers within the cluster. To set the System Temporary Directory:

1. Sign in to BI Publisher with Administrator privileges.
2. Click the **Administration** link.
3. Under **System Maintenance**, click **Server Configuration**.
4. Under **General Properties** in the **System Temporary Directory** property, enter the absolute path to a directory accessible to all servers in the cluster.

   For example, the directory can exist under `${xdo.server.config.dir}/temp` but you must enter the absolute path, such as `/net/subfoldera/scratch/subfolderb/11gcat/temp`

Repeat this procedure for all servers in the cluster, entering the same value for **System Temporary Directory**.

### Signature

uploadReportDataChunk(String fileID, byte[] reportDataChunk, String reportRawDataChunk, String userID, String password);

*Table 4–27 Parameters for uploadReportDataChunk() Method*

| Parameter | Description |
|---|---|
| String fileID | In the first call, you do not need to provide the fileID, after the successful uploading of the first chunk of XML data, it will return a fileID, for example: filename. On your subsequent calls, you can supply the same fileID to append the subsequent data chunks to the same file. |
| byte[] reportDataChunk | The XML data to upload. |
| String reportRawDataChunk | String representation of XML data, presenting as reportRawDataChunk. This is an alternative to reportDataChunk byte[]. |
| String userID | Specifies the BI Publisher user name. |

*Table 4–27 (Cont.) Parameters for uploadReportDataChunk() Method*

| Parameter | Description |
|---|---|
| String password | Specifies the password for the user name. |

## 4.28 uploadReportDataChunkInSession() Method

Use uploadReportDataChunkInSession() method to upload a report data chunk based on the bipSessionToken of a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

uploadReportDataChunkInSession(String fileID, byte[] reportDataChunk, String reportRawDataChunk, String bipSessionToken);

*Table 4–28 Parameters for uploadReportDataChunkInSession() Method*

| Parameter | Description |
|---|---|
| byte[] reportDataChunk | The XML data to upload. |
| String reportRawDataChunk | String representation of XML data, presenting as reportRawDataChunk. This is an alternative to reportDataChunk byte[]. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 4.29 uploadTemplateForReport() Method

Use uploadTemplateForReport() method to upload a template to a report definition in the BI Publisher catalog.

### Signature

boolean uploadTemplateForReport(String reportAbsolutePath, String templateName, String templateType, String locale, byte[] templateData, String userID, String password);

*Table 4–29 Parameters for uploadTemplateForReport() Method*

| Parameter | Description |
|---|---|
| String reportAbsolutePath | The path to the report to which to upload the template. For example: /HR Manager/Employee Reports/Employee Listing.xdo |
| String templateName | The file name of the template to upload. |

*Table 4–29 (Cont.) Parameters for uploadTemplateForReport() Method*

| Parameter | Description |
|---|---|
| String templateType | The template type. Valid values are: <br>■ `csv` (CSV)<br>■ `eText` (eText template)<br>■ `excel` (Microsoft Excel)<br>■ `excel2000` (Microsoft Excel 2000)<br>■ `flash` (Adobe Flash)<br>■ `html` (HTML)<br>■ `mhtml` (MIME HTML)<br>■ `pdf` (Adobe PDF)<br>■ `pdfz` (eBook)<br>■ `ppt` (Microsoft PowerPoint)<br>■ `rtf` (Rich Text Format)<br>■ `text` (Text)<br>■ `txml` (Transformed XML)<br>■ `xml` (XML)<br>■ `xpa` (Analyzer template)<br>■ `xpt` (BI Publisher template)<br>■ `xslfo` (XSL-FO style sheet) |
| String locale | The locale to assign to the template (for example, `en_US`). |
| byte[] templateData | The contents of the template file to upload. |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 4.30 uploadTemplateForReportInSession() Method

Use uploadTemplateForReportInSession() method to upload a template to a report definition in the BI Publisher catalog based on the bipSessionToken of a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

boolean uploadTemplateForReportInSession(String reportAbsolutePath, String templateName, String templateType, String locale, byte[] templateData, String bipSessionToken);

*Table 4–30 Parameters for uploadTemplateForReportInSession() Method*

| Parameter | Description |
|---|---|
| String reportAbsolutePath | The path to the report to which to upload the template. For example: /HR Manager/Employee Reports/Employee Listing.xdo |
| String templateFileName | The file name of the template to upload. |
| String templateName | The name of the template to upload. |
| String locale | The locale to assign to the template (for example, `en_US`). |

***Table 4–30 (Cont.) Parameters for uploadTemplateForReportInSession() Method***

| Parameter | Description |
|---|---|
| byte[] templateData | The contents of the template file to upload. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 4.31 uploadXLIFFForReport() Method

Use uploadXLIFFForReport() method to upload a translation file (XLIFF) to a layout definition in the BI Publisher catalog.

### Signature

boolean uploadXLIFFForReport(String reportAbsolutePath, byte[] xliffData, String layoutFileName, String locale, String userID, String password);

***Table 4–31 Parameters for uploadXLIFFForReport() Method***

| Parameter | Description |
|---|---|
| String reportAbsolutePath | The path to the report to which to upload the XLIFF. For example: /HR Manager/Employee Reports/Employee Listing.xdo |
| byte[] xliffData | The XLIFF fie to upload. |
| String layoutFileName | The file name of the layout to which to associate the XLIFF file. For example: employee_listing.rtf. |
| String locale | The locale to assign to the XLIFF (for example, `en_US`). |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 4.32 uploadXLIFFForReportInSession() Method

Use uploadXLIFFForReport() method to upload a translation file (XLIFF) to a layout definition in the BI Publisher catalog based on the bipSessionToken of a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

boolean uploadXLIFFForReportInSession(String reportAbsolutePath, byte[] xliffData, String layoutFileName, String locale, String bipSessionToken);

***Table 4–32 Parameters for uploadXLIFFForReportInSession() Method***

| Parameter | Description |
|---|---|
| String reportAbsolutePath | The path to the report to which to upload the XLIFF. For example: /HR Manager/Employee Reports/Employee Listing.xdo |
| byte[] xliffData | The XLIFF fie to upload. |

***Table 4–32 (Cont.) Parameters for uploadXLIFFForReportInSession() Method***

| **Parameter** | **Description** |
|---|---|
| String layoutFileName | The file name of the layout to which to associate the XLIFF file. For example: employee_listing.rtf. |
| String locale | The locale to assign to the XLIFF (for example, `en_US`). |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

# 5 SecurityService

This chapter provides details on the SecurityService methods to manage BI Publisher server security operations, such as authentication, impersonation, login, logout, and account management.

This chapter includes the following sections:

**Note:** For information on debugging applications built with BI Publisher Web services, see Section 1.4, "Debugging Web Service Applications."

**Note:** SecurityService is available to the BI Publisher Security Model only. If your BI Publisher deployment uses another security model (for example, LDAP, Oracle E-Business Suite, or Oracle Fusion Apps), you cannot use the SecurityService API.

## 5.1 assignRolesToUser() Method

Use assignRolesToUser() method to assign new roles to a user in BI Publisher.

**Signature**

String[] assignRolesToUser(String userName, String[] roleNames, String adminUser, String adminPassword);

***Table 5–1 Parameters for assignRolesToUser() Method***

| Parameter | Description |
|---|---|
| String userName | The user to which to add the role or roles. |
| String[] roleNames | The name of the role to add to the user. For example, "Financial Users". |
| String adminUser | Specifies a BI Publisher user name for a user with administration privileges. |
| String adminPassword | Specifies the password for the administration user name. |

## 5.2 createRole() Method

Use createRole() method to create a role in BI Publisher.

**Signature**

boolean createRole(String roleName, String description, String adminUser, String adminPassword);

***Table 5–2 Parameters for createRole() Method***

| Parameter | Description |
|---|---|
| String roleName | The name of the role to create. For example, "Financial Users". |
| String description | The description of the role. |
| String adminUser | Specifies a BI Publisher user name for a user with administration privileges. |
| String adminPassword | Specifies the password for the administration user name. |

## 5.3 createUser() Method

Use createUser() method to create a user in BI Publisher. This method returns a boolean value of the success of the method.

**Signature**

boolean createUser(String userName, String password, String adminUser, String adminPassword);

***Table 5–3 Parameters for createUser() Method***

| Parameter | Description |
|---|---|
| String userName | The user name to create. |
| String password | The password for the newly created user. |
| String adminUser | Specifies a BI Publisher user name for a user with administration privileges. |
| String adminPassword | Specifies the password for the administration user name. |

## 5.4 deleteRole() Method

Use deleteRole() method to delete a role from BI Publisher. This method returns a boolean value of the success of the method.

**Signature**

boolean deleteRole(String roleName, String adminUser, String adminPassword);

***Table 5–4 Parameters for deleteRole() Method***

| Parameter | Description |
|---|---|
| String roleName | The user name to delete. |
| String adminUser | Specifies a BI Publisher user name for a user with administration privileges. |
| String adminPassword | Specifies the password for the administration user name. |

## 5.5 deleteUser() Method

Use deleteUser() method to delete a user from BI Publisher. This method returns a boolean value of the success of the method.

**Signature**

boolean deleteUser(String userName, String adminUser, String adminPassword);

***Table 5–5 Parameters for deleteUser() Method***

| Parameter | Description |
|---|---|
| String userName | The user name to delete. |
| String adminUser | Specifies a BI Publisher user name for a user with administration privileges. |
| String adminPassword | Specifies the password for the administration user name. |

## 5.6 getBIPHTTPSessionInterval() Method

This method returns the number of seconds an HTTP session interval is.

**Signature**

int getBIPHTTPSessionInterval(void);

## 5.7 getObjectSecurityXML() Method

This method extracts the report-level permissions (from security.xml) for a BIEE integrated catalog.

### Signature

byte[] getObjectSecurityXML(String adminUsername, String adminPassword, String objectAbsolutePath, boolean isRecursive);

*Table 5–6 Parameters for getObjectSecurityXML() Method*

| Parameter | Description |
|---|---|
| String adminUsername | The user name for a BI Publisher user with administrator privileges. |
| String adminPassword | The password associated with the adminUserName. |
| String objectAbsolutePath | The absolute path to the catalog object for which to retrieve the permissions description. |
| boolean isRecursive | Whether or not objectAbsolutePath is recursive. |

## 5.8 getSecurityModel() Method

This method returns BI Publisher's security model in place.

### Signature

String getSecurityModel(void);

## 5.9 hasObjectAccess() Method

This method verifies if the specified user has access to the report object referenced by reportAbsolutePath. This method first authenticates user with the specified credentials. Upon successful authentication, it verifies the user's privileges to access the report object.

### Signature

boolean hasObjectAccess(String reportAbsolutePath, String roleName, String userID, String password);

*Table 5–7 Parameters for hasObjectAccess() Method*

| Parameter | Description |
|---|---|
| String reportAbsolutePath | The path to the report object for which you want to verify the user's access privileges. For example: /HR Manager/Employee Reports/Employee Listing.xdo |
| String roleName | For future use. Ignore this parameter as it is not yet functional. |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 5.10 hasObjectAccessInSession() Method

This method verifies if a pre-authenticated bipSession has the privilege to access the report object relative to reportAbsolutePath.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

**Signature**

boolean hasObjectAccessInSession(string reportAbsolutePath, string roleName, string bipSessionToken);

***Table 5–8 Parameters for hasObjectAccessInSession() Method***

| Parameter | Description |
|---|---|
| String reportAbsolutePath | The absolute path to the report object. |
| String roleName | The role associated with the given user. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 5.11 impersonate() Method

This method enables an admin account to act on the behalf of a user account. This is very useful if the user doesn't have a known password to be authenticated by BI Publisher server. This method logs in using admin account privilege, then switches the owner of the BI Publisher server session to the passed-in username. Therefore, bipSession token later will be verified by passed-in username. All further BI Publisher operations are performed through give n username.

**Signature**

String impersonate(String adminUsername, String adminPassword,String username);

***Table 5–9 Parameters for impersonate() Method***

| Parameter | Description |
|---|---|
| String adminUserName | Specifies a BI Publisher user name for a user with administration privileges |
| String adminPassword | Specifies the password for the administration user name. |
| String username | The username of the user account that will be granted administrator privileges. |

## 5.12 isUserExists() Method

Use isUserExists() method to test if a user name exists in the BI Publisher security model. This method returns the result as a boolean value.

**Signature**

boolean isUserExists(String userName, String adminUser, String adminPassword);

***Table 5–10 Parameters for isUserExists() Method***

| Parameter | Description |
|---|---|
| String userName | The user name to test. |

*Table 5–10 (Cont.) Parameters for isUserExists() Method*

| Parameter | Description |
|---|---|
| String adminUser | Specifies a BI Publisher user name for a user with administration privileges. |
| String adminPassword | Specifies the password for the administration user name. |

## 5.13 login() Method

Use the login() method to log in to BI Publisher and perform other BI Publisher actions using Web Services. The login() method returns a String, which will become the BI Publisher session ID

### Signature

String login(String userID, String password);

*Table 5–11 Parameters for login() Method*

| Parameter | Description |
|---|---|
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 5.14 logout() Method

This method, in effect, logs the specified user out of the system by invalidating the user's bipSessionToken. After successful logout, the bipSessionToken string is no longer valid.

### Signature

boolean logout(String bipSessionToken);

*Table 5–12 Parameters for logout() Method*

| Parameter | Description |
|---|---|
| String bipSessionToken | The BI Publisher session ID. |

## 5.15 notifyBIEEPreferencesUpdated() Method

This method is provided for BIEE user preference integration purpose only.

### Signature

boolean notifyBIEEPreferencesupdated(bieeSessionID);

*Table 5–13 Parameters for notifyBIEEPreferencesUpdated() Method*

| Parameter | Description |
|---|---|
| String bieeSessionID | The session ID for Oracle Business Intelligence/BI Publisher integration. |

## 5.16 notifyBIEEPreferencesUpdatedWithString() Method

This method is provided for BIEE user preference integration purpose only.

### Signature

boolean notifyBIEEPreferencesUpdatedWithString(String bieeSessionID, String userPrefesXML);

***Table 5–14 Parameters for notifyBIEEPreferencesUpdatedWithString() Method***

| Parameter | Description |
|---|---|
| String bieeSessionID | The session ID for Oracle Business Intelligence/BI Publisher integration. |
| String userPrefesXML | The XML data that contain user preferences. |

## 5.17 removeRolesFromUser() Method

Use removeRolesFromUser() method to remove roles from a user in BI Publisher.

### Signature

String[] removeRolesFromUser(String userName, String[] roleNames, String adminUser, String adminPassword);

***Table 5–15 Parameters for removeRolesFromUser() Method***

| Parameter | Description |
|---|---|
| String userName | The user from which to delete the role or roles. |
| String[] roleNames | The name of the role to delete from the user. For example, "Financial Users". |
| String adminUser | Specifies a BI Publisher user name for a user with administration privileges. |
| String adminPassword | Specifies the password for the administration user name. |

## 5.18 updateRole() Method

Use updateRole() method to update the description of a role that currently exists in BI Publisher.

### Signature

boolean updateRole(String currentRoleName, String newDescription, String adminUser, String adminPassword);

***Table 5–16 Parameters for updateRole() Method***

| Parameter | Description |
|---|---|
| String currentRoleName | The name of the role to update. |
| String newDescription | The updated description of the role to apply. |
| String adminUser | Specifies a BI Publisher user name for a user with administration privileges. |
| String adminPassword | Specifies the password for the administration user name. |

## 5.19 updateUser() Method

Use updateUser() method to update a user's password in BI Publisher. This method returns a boolean value of the success of the method.

### Signature

boolean updateUser(String currentUsername, String newPassword, String adminUser, String adminPassword);

***Table 5–17 Parameters for updateUser() Method***

| Parameter | Description |
|---|---|
| String currentUserName | The user name to update. |
| String newPassword | The new password to assign to the user name. |
| String adminUser | Specifies a BI Publisher user name for a user with administration privileges. |
| String adminPassword | Specifies the password for the administration user name. |

## 5.20 validateLogin() Method

Use the validateLogin() method to validate that a UserID and Password have the privilege to access the Oracle BI Publisher report server.

### Signature

boolean validateLogin(String userID, String password);

***Table 5–18 Parameters for validateLogin() Method***

| Parameter | Description |
|---|---|
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

# 6 CatalogService

This chapter describes the CatalogService methods to interact with the BI Publisher server top-level catalog. CatalogService manages all report objects, including folders, reports, data models, style templates, and sub-templates, and provides methods for common operations, such as create, delete, copy, and rename.

This chapter includes the following sections:

- Section 6.25, "uploadObject() Method"
- Section 6.26, "uploadObjectInSession() Method"
- Section 6.27, "uploadXLIFF() Method"
- Section 6.28, "uploadXLIFFInSession() Method"

**Note:** For information on debugging applications built with BI Publisher Web services, see Section 1.4, "Debugging Web Service Applications."

## 6.1 copyObject() Method

Use copyObject() method to copy an object in the BI Publisher catalog.

### Signature

boolean copyObject(String srcOjectAbsolutePath, String destObjectAbsolutePath, String newName, String userID, String password);

***Table 6–1 Parameters for copyObject() Method***

| Parameter | Description |
|---|---|
| String srcOjectAbsolutePath | The path to the catalog object to copy. |
| String destObjectAbsolutePath | The path to the location in the catalog to which to copy the object. |
| String newName | The name to assign the new object. |
| String userID | Specifies a BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 6.2 copyObjectInSession() Method

This method copies the report object referenced by srcObjectAbsolutePath to a destination folder specified by destFolderAbsolutePath.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

boolean copyObjectInSession(String srcObjectAbsolutePath, String destFolderAbsolutePath, String newName, String bipSessionToken);

***Table 6–2 Parameters for copyObjectInSession() Method***

| Parameter | Description |
|---|---|
| String srcOjectAbsolutePath | The path to the catalog object to copy. |
| String destObjectAbsolutePath | The path to the location in the catalog to which to copy the object. |
| String newName | The name to assign the new object. |

*Table 6–2 (Cont.) Parameters for copyObjectInSession() Method*

| Parameter | Description |
|---|---|
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 6.3 createFolder() Method

Use createFolder() method to create a folder in the BI Publisher catalog.

### Signature

String createFolder(String folderAbsolutePath, String userID, String password);

*Table 6–3 Parameters for createFolder() Method*

| Parameter | Description |
|---|---|
| String folderAbsolutePath | The path to the folder that you want to create. For example: /HR Manager/Employee Reports/ |
| String userID | Specifies a BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 6.4 createFolderInSession() Method

Use createFolderInSession() method to create a folder in the BI Publisher catalog for a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

String createFolderInSession(String folderAbsolutePath, String bipSessionToken);

*Table 6–4 Parameters for createFolderInSession() Method*

| Parameter | Description |
|---|---|
| String folderAbsolutePath | The path to the folder that you want to create. For example: /HR Manager/Employee Reports/ |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 6.5 createObject() Method

Use createObject() method to create an object in BI Publisher catalog.

### Signature

String createObject(String folderAbsolutePathURL, String objectName, String objectType, String objectDescription, byte[] objectData, String userID, String password);

***Table 6–5 Parameters for createObject() Method***

| Parameter | Description |
|---|---|
| String folderAbsolutePathURL | The absolute path to the folder in the catalog in which to place the new object. |
| String objectName | The name of the new object. |
| String objectType | The type of catalog object. Valid values are:<br>`xdm` (data model)<br>`xdo` (report)<br>`xsb` (sub-template)<br>`xss` (style template) |
| String objectDescription | Specifies the description of the new object. |
| byte[] objectData | The byte data of the object. |
| String userID | Specifies a BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 6.6 createObjectInSession() Method

Use createObjectInSession() method to create an object in BI Publisher catalog for a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

String createObjectInSession(String folderAbsolutePathURL, String objectName, String objectType, String objectDescription, byte[] objectData, String bipSessionToken);

***Table 6–6 Parameters for createObjectInSession() Method***

| Parameter | Description |
|---|---|
| String folderAbsolutePathURL | The absolute path to the folder in the catalog in which to place the new object. |
| String objectName | The name of the new object. |
| String objectType | The type of catalog object. Valid values are:<br>`xdm` (data model)<br>`xdo` (report)<br>`xsb` (sub-template)<br>`xss` (style template) |
| String objectDescription | Specifies the description of the new object. |
| byte[] objectData | The byte data of the object. |

*Table 6–6 (Cont.) Parameters for createObjectInSession() Method*

| Parameter | Description |
|---|---|
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 6.7 deleteObject() Method

Use deleteObject() method to delete an object from the BI Publisher catalog.

### Signature

boolean deleteObject(String reportObjectAbsolutePath, String userID, String password);

*Table 6–7 Parameters for deleteObject() Method*

| Parameter | Description |
|---|---|
| String reportObjectAbsolutePath | The path to the object in the catalog to delete. |
| String userID | Specifies a BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 6.8 deleteObjectInSession() Method

Use deleteObjectInSession() method to delete an object from the BI Publisher catalog for a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

boolean deleteObjectInSession(String objectAbsolutePath, String bipSessionToken);

*Table 6–8 Parameters for deleteObjectInSession() Method*

| Parameter | Description |
|---|---|
| String objectAbsolutePath | The path to the object in the catalog to delete. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 6.9 downloadObject() Method

Use downloadObject() method to download an object from the BI Publisher catalog. This method returns the requested object in binary.

### Signature

byte[] downloadObject(String reportAbsolutePath, String userID, String password);

*Table 6–9 Parameters for downloadObject() Method*

| Parameter | Description |
|---|---|
| String reportAbsolutePath | The path to the object in the catalog to download. |
| String userID | Specifies a BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 6.10 downloadObjectInSession() Method

Use downloadObjectInSession() method to download an object from the BI Publisher catalog for a given user. This method returns the requested object in binary.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

byte[] downloadObjectInSession(String reportAbsolutePath, String bipSessionToken);

*Table 6–10 Parameters for downloadObjectInSession() Method*

| Parameter | Description |
|---|---|
| String reportAbsolutePath | The path to the object in the catalog to download. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 6.11 downloadXLIFF() Method

Use downloadXLIFF() method to download a translation file (XLIFF) from the catalog. This method returns the requested XLIFF file in binary.

### Signature

byte[] downloadXLIFF(String objectAbsolutePath, String userID, String password);

*Table 6–11 Parameters for downloadXLIFF() Method*

| Parameter | Description |
|---|---|
| String objectAbsolutePath | The path to the XLIFF object to download. |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 6.12 downloadXLIFFInSession() Method

Use downloadXLIFFInSession() method to download a translation file (XLIFF) from the catalog downloadXLIFFInSession a given user. This method returns the requested XLIFF file in binary.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

byte[] downloadXLIFFInSession(String objectAbsolutePath, String locale, String bipSessionToken);

***Table 6–12 Parameters for downloadXLIFFInSession() Method***

| Parameter | Description |
|---|---|
| String objectAbsolutePath | The path to the XLIFF object to download. |
| String locale | The locale of the XLIFF object (for example, `en_US`). |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 6.13 getFolderContents() Method

Use getFolderContents to get all of the items in a folder. This will return all the reports and folders contained in the specified folder. You can then use these items to determine what reports you might want to execute or what folders you may want to further search to identify a report.

See CatalogContents for a description of the return object.

### Signature

CatalogContents getFolderContents(String folderAbsolutePath, String userID, String password);

***Table 6–13 Parameters for getFolderContents() Method***

| Parameter | Description |
|---|---|
| String folderAbsolutePath | The path to the folder for which to retrieve the contents. For example: /HR Manager/Employee Reports/ |
| String userID | Specifies a BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 6.14 getFolderContentsInSession() Method

Use getFolderContentsInSession() to get all of the items in a folder for a given user. This will return all the reports and folders contained in the specified folder. You can then use these items to determine what reports you might want to execute or what folders you may want to further search to identify a report.

See CatalogContents for a description of the return object.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

CatalogContents getFolderContentsInSession(String folderAbsolutePath, String bipSessionToken);

***Table 6–14 Parameters for getFolderContentsInSession() Method***

| **Parameter** | **Description** |
|---|---|
| String folderAbsolutePath | The path to the folder for which to retrieve the contents. For example: /HR Manager/Employee Reports/ |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 6.15 getObject() Method

Use getObject() method to download an object from the catalog. This method returns the requested object file in binary.

### Signature

byte[] getObject(String reportObjectAbsolutePath, String locale, String userID, String password);

***Table 6–15 Parameters for getObject() Method***

| **Parameter** | **Description** |
|---|---|
| String reportObjectAbsolutePath | The path to the object to download. |
| String locale | The locale of the object to get. |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 6.16 getObjectInfo() Method

Use getObjectInfo() method to get information about an object in the BI Publisher catalog. This method returns the CatalogObjectInfo object. See CatalogObjectInfo.

### Signature

CatalogObjectInfo getObjectInfo(String reportObjectAbsolutePath, String userID, String password);

***Table 6–16 Parameters for getObjectInfo() Method***

| **Parameter** | **Description** |
|---|---|
| String reportObjectAbsolutePath | The path to the report object about which to get information. |
| String userID | Specifies a BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 6.17 getObjectInfoInSession() Method

Use getObjectInfoInSession() method to get information about an object in the BI Publisher catalog for a given user. This method returns the CatalogObjectInfo object. See CatalogObjectInfo.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

CatalogObjectInfo getObjectInfoInSession(String objectAbsolutePath, String bipSessionToken);

***Table 6–17 Parameters for getObjectInfoInSession() Method***

| Parameter | Description |
|---|---|
| String reportObjectAbsolutePath | The path to the report object about which to get information. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 6.18 getObjectInSession() Method

Use getObjectInSession() method to download an object from the catalog for a given user. This method returns the requested object file in binary.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

byte[] getObjectInSession(String objectAbsolutePath, String bipSessionToken);

***Table 6–18 Parameters for getObjectInSession() Method***

| Parameter | Description |
|---|---|
| String reportObjectAbsolutePath | The path to the object to retrieve. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 6.19 objectExist() Method

Use objectExist() method to determine if an object exists in the BI Publisher catalog.

### Signature

boolean objectExist(String reportObjectAbsolutePath, String userID, String password);

***Table 6–19 Parameters for objectExist() Method***

| Parameter | Description |
|---|---|
| String reportOjectAbsolutePath | The path to the object to test for in the catalog. For example: /HR Manager/Employee Reports/Employee Data Model.xdm |
| String userID | Specifies a BI Publisher user name. |

*Table 6–19 (Cont.) Parameters for objectExist() Method*

| Parameter | Description |
|---|---|
| String password | Specifies the password for the user name. |

## 6.20 objectExistInSession() Method

Use objectExist() method to determine if an object exists in the BI Publisher catalog for a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

boolean objectExistInSession(String reportObjectAbsolutePath, String bipSessionToken);

*Table 6–20 Parameters for objectExistInSession() Method*

| Parameter | Description |
|---|---|
| String reportOjectAbsolutePath | The path to the object to test for in the catalog. For example: /HR Manager/Employee Reports/Employee Data Model.xdm |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 6.21 renameObject() Method

Use renameObject() method to rename an object in the BI Publisher catalog.

### Signature

boolean renameObject(String reportObjectAbsolutePath, String newName, String userID, String password);

*Table 6–21 Parameters for renameObject() Method*

| Parameter | Description |
|---|---|
| String reportObjectAbsolutePath | The path to the object in the catalog to rename. |
| String newName | The new name to assign the object. |
| String userID | Specifies a BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 6.22 renameObjectInSession() Method

Use renameObject() method to rename an object in the BI Publisher catalog for a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

boolean renameObjectInSession(String objectAbsolutePath, String newName, String bipSessionToken);

*Table 6–22 Parameters for renameObjectInSession() Method*

| Parameter | Description |
|---|---|
| String reportObjectAbsolutePath | The path to the object in the catalog to rename. |
| String newName | The new name for the object. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 6.23 updateObject() Method

Use updateObject() method to update an object in the BI Publisher catalog.

### Signature

boolean updateObject(String reportObjectAbsolutePath, byte[] objectData, String userID, String password);

*Table 6–23 Parameters for updateObject() Method*

| Parameter | Description |
|---|---|
| String reportOjectAbsolutePath | The path to the object to update in the catalog. For example: /HR Manager/Employee Reports/Employee Data Model.xdm |
| byte[] objectData | The data with which to update the object. |
| String userID | Specifies a BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 6.24 updateObjectInSession() Method

Use updateObject() method to update an object in the BI Publisher catalog for a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

boolean updateObjectInSession(String objectAbsolutePath, byte[] objectData, String bipSessionToken);

*Table 6–24 Parameters for updateObjectInSession() Method*

| Parameter | Description |
|---|---|
| String reportOjectAbsolutePath | The path to the object to update in the catalog. For example: /HR Manager/Employee Reports/Employee Data Model.xdm |
| byte[] objectData | The data with which to update the object. |

***Table 6–24 (Cont.) Parameters for updateObjectInSession() Method***

| **Parameter** | **Description** |
|---|---|
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 6.25 uploadObject() Method

Use uploadObject() method to upload a new object to the BI Publisher catalog.

### Signature

String uploadObject(String reportObjectAbsolutePathURL, String objectType, byte[] objectZippedData, String userID, String password);

***Table 6–25 Parameters for uploadObject() Method***

| **Parameter** | **Description** |
|---|---|
| String reportObjectAbsolutePathURL | The path to the object in the catalog. |
| String objectType | The type of object to upload. Valid values are:<br>`xdm` (data model)<br>`xdo` (report)<br>`xsb` (sub-template)<br>`xss` (style template) |
| byte[] objectZippedData | The object to upload in zipped format. |
| String userID | Specifies a BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 6.26 uploadObjectInSession() Method

Use uploadObject() method to upload a new object to the BI Publisher catalog for a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

String uploadObject(String reportObjectAbsolutePathURL, String objectType, byte[] objectZippedData, String userID, String bipSessionToken);

***Table 6–26 Parameters for uploadObjectInSession() Method***

| **Parameter** | **Description** |
|---|---|
| String reportObjectAbsolutePathURL | The path to the object in the catalog. |

*Table 6–26 (Cont.) Parameters for uploadObjectInSession() Method*

| Parameter | Description |
|---|---|
| String objectType | The type of object to upload. Valid values are:<br>`xdm` (data model)<br>`xdo` (report)<br>`xsb` (sub-template)<br>`xss` (style template) |
| byte[] objectZippedData | The object to upload in zipped format. |
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

## 6.27 uploadXLIFF() Method

Use uploadXLIFF() method to upload a translation file (XLIFF) to the catalog.

### Signature

boolean uploadXLIFF(String objectAbsolutePath, byte[] xliffData, String locale, String userID, String password);

*Table 6–27 Parameters for uploadXLIFF() Method*

| Parameter | Description |
|---|---|
| String objectAbsolutePath | The path to the XLIFF object to upload. |
| byte[] xliffData | The XLIFF fie to upload. |
| String locale | The locale to assign to the XLIFF (for example, `en_US`). |
| String userID | Specifies the BI Publisher user name. |
| String password | Specifies the password for the user name. |

## 6.28 uploadXLIFFInSession() Method

Use uploadXLIFF() method to upload a translation file (XLIFF) to the catalog for a given user.

For more information about in-session methods, see Section 1.3, "About In-Session Methods."

### Signature

boolean uploadXLIFF(String objectAbsolutePath, byte[] xliffData, String locale, String bipSessionToken);

*Table 6–28 Parameters for uploadXLIFFInSession() Method*

| Parameter | Description |
|---|---|
| String objectAbsolutePath | The path to the XLIFF object to upload. |
| byte[] xliffData | The XLIFF fie to upload. |
| String locale | The locale to assign to the XLIFF (for example, `en_US`). |

***Table 6–28 (Cont.) Parameters for uploadXLIFFInSession() Method***

| Parameter | Description |
|---|---|
| String bipSessionToken | The proprietary token string generated for the user by the BI Publisher server. With the bipSessionToken string, the user no longer needs to provide user credentials. The BI Publisher server can validate this token string and restore the BI Publisher server session to perform needed operation. |

# Part II

# Oracle BI Publisher Java APIs

The Oracle BI Publisher Java APIs provide developers the ability to embed the powerful document generation and delivery capabilities directly into custom applications. BI Publisher provides a collection of document generation APIs for the various template types that it supports. BI Publisher also provides APIs for merging documents and for delivering them to a wide range of destinations, including through custom delivery channels.

This part contains the following chapters on the Oracle BI Publisher Java APIs:

7

# Using the BI Publisher Java APIs

This chapter describes the BI Publisher Java APIs that can be called from a custom application to generate and process documents.

It includes the following sections:

**Note:** The information in this chapter is intended to be used with the *Oracle Fusion Middleware Java API Reference for Oracle Business Intelligence Publisher*, which is available in the Oracle Fusion Middleware Business Intelligence Documentation Library. Also, this chapter assumes you are familiar with Java programming, XML, and XSL technologies.

## 7.1 BI Publisher Core APIs

BI Publisher is made up of the following core API components:

- PDF Form Processing Engine

  Merges a PDF template with XML data (and optional metadata) to produce PDF document output. See Section 7.4, "PDF Form Processing Engine."

- RTF Processor Engine

Converts an RTF template to XSL in preparation for input to the FO Engine. See Section 7.5, "RTF Processor Engine."

- FO Processor Engine

  Merges XSL and XML to produce any of the following output formats: Excel (HTML), PDF, RTF, or HTML. See Section 7.6, "FO Processor Engine."

- PDF Document Merger

  Provides optional postprocessing of PDF files to merge documents, add page numbering, and set watermarks. See Section 7.7, "PDF Document Merger."

- eText Processor

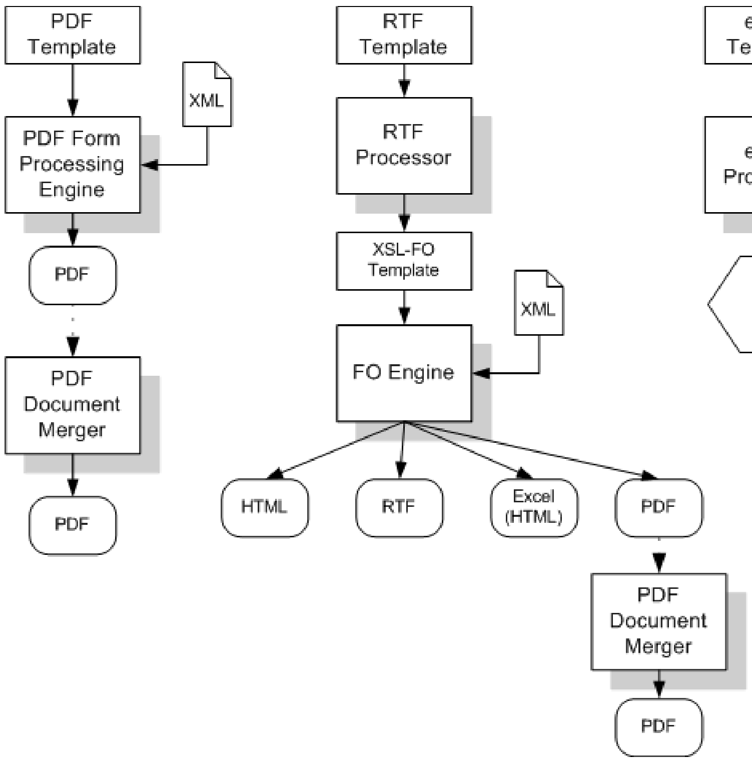
  Converts RTF eText templates to XSL and merges the XSL with XML to produce text output for EDI and EFT transmissions. See Section 7.10, "eText Processor."

- Document Processor Engine (XML APIs)

  Provides batch processing functionality to access a single API or multiple APIs by passing a single XML file to specify template names, data sources, languages, output type, output names, and destinations. See Section 7.11, "Document Processor Engine."

The following diagram illustrates the template type and output type options for each core processing engine:

***Figure 7–1 Template and Output Types for BI Publisher Core Processing Engines***



## 7.2 Prerequisites

To use the BI Publisher APIs, ensure that `xdocore.jar` is in your class path. `xdocore.jar` contains the main library for the BI Publisher APIs.

In addition, the following libraries are required:

- aolj.jar - supports various BI Publisher functions
- collections.jar - you only need this if you are working with the delivery APIs or bursting engine.
- dvt-jclient.jar - a charting library
- dvt-utils.jar - a charting library
- groovy-all-1.6.3.jar - Groovy
- i18nAPI_v3.jar - the i18n library used for localization functions
- jewt4.jar - a charting support library
- mail.jar - used for SMTP delivery
- ojdl.jar - used for Oracle Diagnostic Logging
- orai18n-collation.jar - used by XDK
- orai18n-mapping.jar - used by XDK
- orai18n.jar - contains character set and globalization support files used by XDK
- osdt_cert.jar - security library for SMIME support
- osdt_cms.jar - security library for SMIME support
- osdt_core.jar - security library for SMIME support
- osdt_smime.jar - security library for SMIME support
- share.jar - a charting support library
- versioninfo.jar - contains version information for BI Publisher
- xdoparser.jar - the scalable XML parser and XSLT 2.0 engine (10g)
- xdoparser11g.jar - the scalable XML parser and XSLT 2.0 engine (11g)
- xmlparserv2.jar - 11g XDK (SAX and DOM)

## 7.3 Obtaining the Libraries

If you are using Oracle JDeveloper, then the charting and XML Parser libraries are available to you. However, it is recommended that you create a directory with all of the required JAR files to use as a custom library in your project. This will help prevent unexpected errors after deployment.

The easiest method to obtain the libraries is to download and install the Template Builder for Microsoft Word Add-in. Download the Template Builder for Word from the Home page, under **Get Started**, click **Download BI Publisher Tools**, then click **Template Builder for Word**.

The JAR files are packaged with the Template Builder in the jlib library under the install directory.

A sample path to jlib would be:

C:\Program Files\Oracle\BI Publisher\BI Publisher Desktop\Template Builder for Word\jlib

## 7.4 PDF Form Processing Engine

This section discusses how to use the RTP Processor Engine, and includes the following topics:

- Section 7.4.1, "Overview of the PDF Form Processing Engine"
- Section 7.4.2, "Merging a PDF Template with XML Data"
- Section 7.4.3, "Merging XML Data with a PDF Template Using Input/Output File Name"
- Section 7.4.4, "Merging XML Data with a PDF Template Using Input/Output Streams"
- Section 7.4.5, "Merging an XML Data String with a PDF Template"
- Section 7.4.6, "Retrieving a List of Field Names"

### 7.4.1 Overview of the PDF Form Processing Engine

The PDF Form Processing Engine creates a PDF document by merging a PDF template with an XML data file. This can be done using file names, streams, or an XML data string.

As input to the PDF Processing Engine you can optionally include an XML-based Template MetaInfo (.xtm) file. This is a supplemental template to define the placement of overflow data.

The FO Processing Engine also includes utilities to provide information about your PDF template. You can:

- Retrieve a list of field names from a PDF template
- Convert XML data into XFDF using XSLT

### 7.4.2 Merging a PDF Template with XML Data

XML data can be merged with a PDF template to produce a PDF output document in three ways:

- Using input/output file names
- Using input/output streams
- Using an input XML data string

### 7.4.3 Merging XML Data with a PDF Template Using Input/Output File Name

Input:

- Template file name (String)
- XML file name (String)

Output:

- PDF file name (String)

***Example 7–1 Sample Code for Merging XML Data with PDF Templates Using Input/Output File Names***

```
import oracle.xdo.template.FormProcessor;
.
.
```

```
    FormProcessor fProcessor = new FormProcessor();

    fProcessor.setTemplate(args[0]);  // Input File (PDF) name
    fProcessor.setData(args[1]);     // Input XML data file name
    fProcessor.setOutput(args[2]);   // Output File (PDF) name
fProcessor.process();
```

### 7.4.4 Merging XML Data with a PDF Template Using Input/Output Streams

Input:

- PDF Template (Input Stream)
- XML Data (Input Stream)

Output:

- PDF (Output Stream)

***Example 7–2 Sample Code for Merging XML Data with PDF Templates Using Input/Output Streams***

```
import java.io.*;
import oracle.xdo.template.FormProcessor;
.
.
.
  FormProcessor fProcessor = new FormProcessor();

  FileInputStream fIs = new FileInputStream(originalFilePath); // Input File
  FileInputStream fIs2 = new FileInputStream(dataFilePath); // Input Data
  FileInputStream fIs3 = new FileInputStream(metaData); // Metadata XML Data
  FileOutputStream fOs = new FileOutputStream(newFilePath); // Output File

  fProcessor.setTemplate(fIs);
  fProcessor.setData(fIs2);   // Input Data
  fProcessor.setOutput(fOs);
  fProcessor.process();

  fIs.close();
  fOs.close();
```

### 7.4.5 Merging an XML Data String with a PDF Template

Input:

- Template file name (String)
- XML data (String)

Output:

- PDF file name (String)

***Example 7–3 Sample Code for Merging XML Data Strings with PDF Templates***

```
import oracle.xdo.template.FormProcessor;
.
.
```

```
.
FormProcessor fProcessor = new FormProcessor();

fProcessor.setTemplate(originalFilePath);     // Input File (PDF) name
fProcessor.setDataString(xmlContents);       // Input XML string
fProcessor.setOutput(newFilePath);           // Output File (PDF) name
 fProcessor.process();
```

### 7.4.6 Retrieving a List of Field Names

Use the FormProcessor.getFieldNames() API to retrieve the field names from a PDF template. The API returns the field names into an Enumeration object.

Input:

- PDF Template

Output:

- Enumeration Object

***Example 7–4 Sample Code for Retrieving a List of Field Names***

```
import java.util.Enumeration;
import oracle.xdo.template.FormProcessor;
.
.
.
FormProcessor fProcessor = new FormProcessor();
fProcessor.setTemplate(filePath);        // Input File (PDF) name
Enumeration enum = fProcessor.getFieldNames();
while(enum.hasMoreElements())
{
  String formName = (String)enum.nextElement();
  System.out.println("name : " + formName + " , value : " +
fProcessor.getFieldValue(formName));
}
```

### 7.4.7 Generating XFDF Data

XML Forms Data Format (XFDF) is a format for representing forms data and annotations in a PDF document. XFDF is the XML version of Forms Data Format (FDF), a simplified version of PDF for representing forms data and annotations. Form fields in a PDF document include edit boxes, buttons, and radio buttons.

Use this class to generate XFDF data. When you create an instance of this class, an internal XFDF tree is initialized. Use append() methods to append a FIELD element to the XFDF tree by passing a String name-value pair. You can append data as many times as you want.

This class also enables you to append XML data by calling appendXML() methods. Note that you must set the appropriate XSL style sheet by calling setStyleSheet() method before calling appendXML() methods. You can append XML data as many times as you want.

You can retrieve the internal XFDF document at any time by calling one of the following methods: toString(), toReader(), toInputStream(), or toXMLDocument().

The following is a sample of XFDF data:

***Example 7–5 Sample XFDF Data***

```
<?xml version="1.0" encoding="UTF-8"?>
<xfdf xmlns="http://ns.adobe.com/xfdf/" xml:space="preserve">
<fields>
 <field name="TITLE">
  <value>Purchase Order</value>
  </field>
 <field name="SUPPLIER_TITLE">
   <value>Supplie</value>
 </field>
  ...
 </fields>
```

The following code example shows how you can use the API:

***Example 7–6 Sample Code for Retrieving Internal XFDF Documents***

```
import oracle.xdo.template.FormProcessor;
import oracle.xdo.template.pdf.xfdf.XFDFObject;
.
.
.
FormProcessor fProcessor = new FormProcessor();
fProcessor.setTemplate(filePath);      // Input File (PDF) name
XFDFObject xfdfObject = new XFDFObject(fProcessor.getFieldInfo());
System.out.println(xfdfObject.toString());
```

## 7.4.8 Converting XML Data into XFDF Format Using XSLT

Use an XSL style sheet to convert standard XML to the XFDF format. Following is an example of the conversion of sample XML data to XFDF:

Assume your starting XML has a ROWSET/ROW format as follows:

```
<ROWSET>
     <ROW num="0">
       <SUPPLIER>Supplier</SUPPLIER>
       <SUPPLIERNUMBER>Supplier Number</SUPPLIERNUMBER>
       <CURRCODE>Currency</CURRCODE>
     </ROW>
...
</ROWSET>
```

From this XML you want to generate the following XFDF format:

```
 <fields>
  <field name="SUPPLIER1">
    <value>Supplier</value>
  </field>
  <field name="SUPPLIERNUMBER1">
    <value>Supplier Number</value>
  </field>
  <field name="CURRCODE1">
    <value>Currency</value>
  </field>
...
</fields>
```

The following XSLT will perform the transformation:

***Example 7–7 Sample XLST for Transforming XML Data into XFDF Format***

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<fields>
<xsl:apply-templates/>
</fields>
</xsl:template>
   <!-- Count how many ROWs(rows) are in the source XML file. -->
   <xsl:variable name="cnt" select="count(//row|//ROW)" />
   <!-- Try to match ROW (or row) element.
   <xsl:template match="ROW/*|row/*">
      <field>
         <!-- Set "name" attribute in "field" element. -->
         <xsl:attribute name="name">
            <!-- Set the name of the current element (column name)as a value of
the current name attribute. -->
            <xsl:value-of select="name(.)" />
            <!-- Add the number at the end of the name attribute value if more
than 1 rows found in the source XML file.-->
            <xsl:if test="$cnt > 1">
               <xsl:number count="ROW|row" level="single" format="1"/>
            </xsl:if>
         </xsl:attribute>
         <value>
            <!--Set the text data set in the current column data as a text of the
"value" element. -->
            <xsl:value-of select="." />
         </value>
      </field>
   </xsl:template>
</xsl:stylesheet>
```

You can then use the XFDFObject to convert XML to the XFDF format using an XSLT as follows:

***Example 7–8 Sample Code for Executing Transformation of XML Data into XFDF Format***

```
import java.io.*;
import oracle.xdo.template.pdf.xfdf.XFDFObject;
.
.
.
XFDFObject xfdfObject  = new XFDFObject();

xfdfObject .setStylesheet(new BufferedInputStream(new FileInputStream(xslPath)));
// XSL file name
xfdfObject .appendXML( new File(xmlPath1));   // XML data file name
xfdfObject .appendXML( new File(xmlPath2));   // XML data file name

System.out.print(xfdfObject .toString());
```

## 7.5 RTF Processor Engine

This section discusses how to use the RTP Processor Engine, and includes the following topics:

- Section 7.5.1, "Pairing with XLIFF FIle"
- Section 7.5.2, "Generating XSL"

### 7.5.1 Pairing with XLIFF FIle

The RTFProcessor can generate the pairing XLIFF file. The API example is as follows:

***Example 7–9 Sample Code for Generating Pairing XLIFF Files***

```
public static void main(String[] args)
{
RTFProcessor rtfp = new RTFProcessor(args[0]); //input RTF template
rtfp.setOutput(args[1]); // XSL output file
rtfp.setXLIFFOutput(args[2]); // XLIFF output file
rtfp.process();
System.exit(0);
}
```

To generate the translated report, call FOProcessor as follows:

***Example 7–10 Sample Code for Generating Translated Reports***

```
FOProcessor p1 = new FOProcessor();
p1.setXLIFF(xliff);// set xliff file, which is generated from RTFProcessor
p1.setData(xml); // set data file
p1.setTemplate(xsl); // set xsl file
p1.setOutput(pdf);
p1.generate();
```

### 7.5.2 Generating XSL

The RTF processor engine takes an RTF template as input. The processor parses the template and creates an XSL-FO template. This can then be passed along with a data source (XML file) to the FO Engine to produce PDF, HTML, RTF, or Excel (HTML) output.

Use either input/output file names or input/output streams as shown in the following examples:

#### 7.5.2.1 Generating XSL with Input/Output File Names

Input:

- RTF file name (String)

Output:

- XSL file name (String)

***Example 7–11 Sample Code for Generating XSL with Input/Output File Names***

```
import oracle.xdo.template.FOProcessor;
.
.
.
public static void main(String[] args)
 {
  RTFProcessor rtfProcessor = new RTFProcessor(args[0]); //input template
  rtfProcessor.setOutput(args[1]);  // output file
  rtfProcessor.process();
  System.exit(0);
 }
```

#### 7.5.2.2 Generating XSL with Input/Output Stream

Input:

- RTF (InputStream)

Output:

- XSL (OutputStream)

***Example 7–12 Sample Code for Generating XSL with Input/Output Streams***

```
import oracle.xdo.template.FOProcessor;
.
.
.
  public static void main(String[] args)
  {
    FileInputStream   fIs  = new FileInputStream(args[0]);  //input template
    FileOutputStream  fOs  = new FileOutputStream(args[1]); // output

    RTFProcessor rtfProcessor = new RTFProcessor(fIs);
    rtfProcessor.setOutput(fOs);
    rtfProcessor.process();
    // Closes inputStreams outputStream
    System.exit(0);
  }
```

## 7.6 FO Processor Engine

This section discusses how to use the FO Processor Engine, and includes the following topics:

- Section 7.6.1, "Major Features of the FO Processor"
- Section 7.6.2, "Generating Output from an XML File and an XSL File"
- Section 7.6.3, "Generating Output Using File Names"
- Section 7.6.4, "Generating Output Using Streams"
- Section 7.6.5, "Generating Output from an Array of XSL Templates and XML Data"
- Section 7.6.6, "Using the XSL-FO Utility"

### 7.6.1 Major Features of the FO Processor

The FO Processor Engine provides the following features:

- Section 7.6.1.1, "Bidirectional Text"
- Section 7.6.1.2, "Font Fallback Mechanism"
- Section 7.6.1.3, "Variable Header and Footer"
- Section 7.6.1.4, "Horizontal Table Break"

#### 7.6.1.1 Bidirectional Text

BI Publisher utilizes the Unicode BiDi algorithm for BiDi layout. Based on specific values for the properties writing-mode, direction, and unicode bidi, the FO Processor supports the BiDi layout.

The writing-mode property defines how word order is supported in lines and order of lines in text. That is: right-to-left, top-to-bottom or left-to-right, top-to-bottom. The direction property determines how a string of text will be written: that is, in a specific direction, such as right-to-left or left-to-right. The unicode bidi controls and manages override behavior.

#### 7.6.1.2 Font Fallback Mechanism

The FO Processor supports a two-level font fallback mechanism. This mechanism provides control over what default fonts to use when a specified font or glyph is not found. BI Publisher provides appropriate default fallback fonts automatically without requiring any configuration. BI Publisher also supports user-defined configuration files that specify the default fonts to use. For glyph fallback, the default mechanism will only replace the glyph and not the entire string.

#### 7.6.1.3 Variable Header and Footer

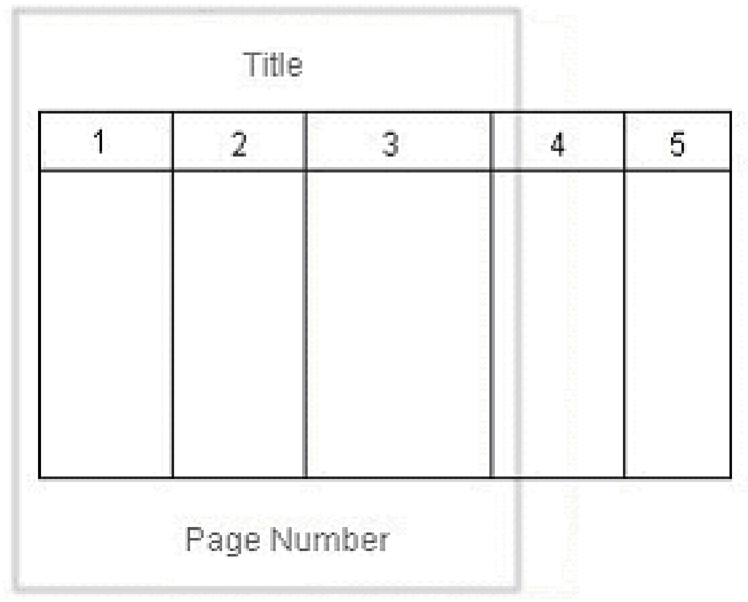
For headers and footers that require more space than what is defined in the template, the FO Processor extends the regions and reduces the body region by the difference between the value of the page header and footer and the value of the body region margin.

#### 7.6.1.4 Horizontal Table Break

This feature supports a "Z style" of horizontal table break. The horizontal table break is not sensitive to column span, so that if the column-spanned cells exceed the page (or area width), the FO Processor splits it and does not apply any intelligent formatting to the split cell.

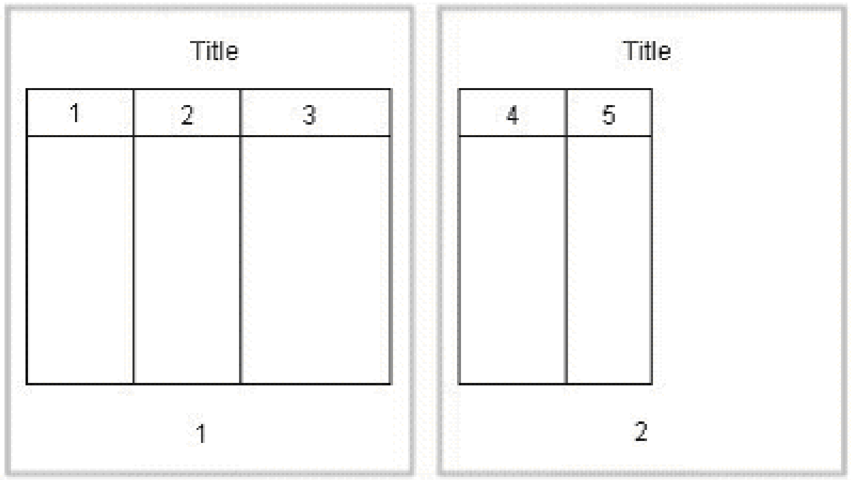
The following figure shows a table that is too wide to display on a single page:

*Figure 7–2 Example of Wide Table*



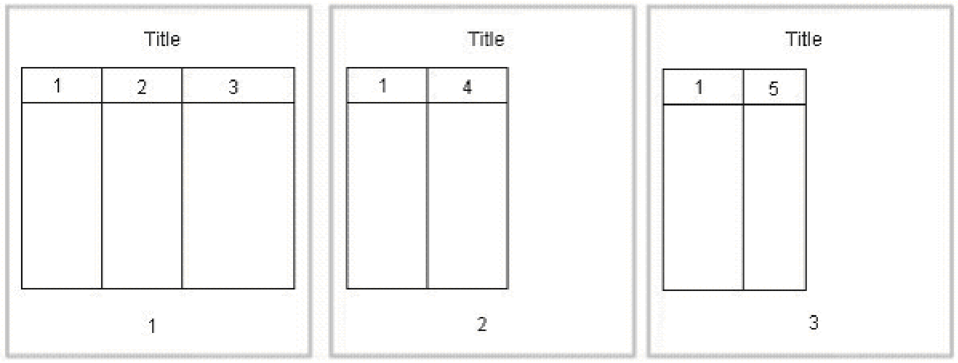
The following figure shows one option of how the horizontal table break will handle the wide table. In this example, a horizontal table break is inserted after the third column.

*Figure 7–3 Example of Horizontal Table Break on Wide Table*



The following figure shows another option. The table breaks after the third column, but includes the first column with each new page.

*Figure 7–4 Example of Horizontal Table Break and Column Repeating on Wide Table*



## 7.6.2 Generating Output from an XML File and an XSL File

The FO Processor Engine is BI Publisher's implementation of the W3C XSL-FO standard. It does not represent a complete implementation of every XSL-FO component.

The FO Processor can generate output in PDF, RTF, HTML, or Excel (HTML) from either of the following input types:

- Template (XSL) and Data (XML) combination
- FO object

Input types can be passed as file names, streams, or in an array. Set the output format by setting the setOutputFormat() method to one of the following:

- FORMAT_EXCEL
- FORMAT_HTML

- FORMAT_PDF
- FORMAT_RTF
- FORMAT_PPTX
- FORMAT_MHTML
- FORMAT_PPTMHT
- FORMAT_EXCEL_MHTML
- FORMAT_PDFZ

An XSL-FO utility is also provided that creates XSL-FO from the following inputs:

- XSL file and XML file
- Two XML files and two XSL files
- Two XSL-FO files (merge)

The FO object output from the XSL-FO utility can then be used as input to the FO processor.

## 7.6.3 Generating Output Using File Names

The following example shows how to use the FO Processor to create an output file using file names.

Input:

- XML file name (String)
- XSL file name (String)

Output:

- Output file name (String)

***Example 7–13 Sample Code for Generating Output Using File Names***

```
import oracle.xdo.template.FOProcessor;
.
.
.
  public static void main(String[] args)
  {

    FOProcessor processor = new FOProcessor();
    processor.setData(args[0]);     // set XML input file
    processor.setTemplate(args[1]); // set XSL input file
    processor.setOutput(args[2]);  //set output file
    processor.setOutputFormat(FOProcessor.FORMAT_PDF);
    // Start processing
    try
    {
       processor.generate();
    }
    catch (XDOException e)
    {
       e.printStackTrace();
       System.exit(1);
    }

    System.exit(0);
  }
```

## 7.6.4 Generating Output Using Streams

The processor can also be used with input/output streams as shown in the following example:

Input:

- XML data (InputStream)
- XSL data (InputStream)

Output:

- Output stream (OutputStream)

***Example 7–14 Sample Code for Generating Output Using Streams***

```
import java.io.InputStream;
import java.io.OutputStream;
import oracle.xdo.template.FOProcessor;
.
.
.
  public void runFOProcessor(InputStream xmlInputStream,
                             InputStream xslInputStream,
                             OutputStream pdfOutputStream)
  {

    FOProcessor processor = new FOProcessor();
    processor.setData(xmlInputStream);
    processor.setTemplate(xslInputStream);
    processor.setOutput(pdfOutputStream);
    // Set output format (for PDF generation)
    processor.setOutputFormat(FOProcessor.FORMAT_PDF);
    // Start processing
    try
    {
       processor.generate();
    }
    catch (XDOException e)
    {
       e.printStackTrace();
       System.exit(1);
    }

    System.exit(0);

  }
```

## 7.6.5 Generating Output from an Array of XSL Templates and XML Data

An array of data and template combinations can be processed to generate a single output file from the multiple inputs. The number of input data sources must match the number of templates that are to be applied to the data. For example, an input of File1.xml, File2.xml, File3.xml and File1.xsl, File2.xsl, and File3.xsl will produce a single File1_File2_File3.pdf.

Input:

- XML data (Array)
- XSL data (template) (Array)

Output:

- File Name (String)

***Example 7–15 Sample Code for Generating Output from XSL Template Arrays and XML Data***

```
import java.io.InputStream;
import java.io.OutputStream;
import oracle.xdo.template.FOProcessor;
.
.
.
   public static void main(String[] args)
  {

    String[] xmlInput = {"first.xml", "second.xml", "third.xml"};
    String[] xslInput = {"first.xsl", "second.xsl", "third.xsl"};


    FOProcessor processor = new FOProcessor();
    processor.setData(xmlInput);
    processor.setTemplate(xslInput);

    processor.setOutput("/tmp/output.pdf);          //set (PDF) output file
    processor.setOutputFormat(FOProcessor.FORMAT_PDF); processor.process();
    // Start processing
    try
    {
       processor.generate();
    }
    catch (XDOException e)
    {
       e.printStackTrace();
       System.exit(1);
    }

  }
```

## 7.6.6 Using the XSL-FO Utility

Use the XSL-FO Utility to create an XSL-FO output file from input XML and XSL files, or to merge two XSL-FO files. You can use the output from this utility to generate your final output. See Section 7.6.2, "Generating Output from an XML File and an XSL File."

### 7.6.6.1 Creating XSL-FO from an XML File and an XSL File

Input:

- XML file
- XSL file

Output:

- XSL-FO (InputStream)

***Example 7–16 Sample Code for Creating XSL-FO from XML and XSL Files***

```
import oracle.xdo.template.fo.util.FOUtility;
.
.
.
  public static void main(String[] args)
  {
    InputStream foStream;

    // creates XSL-FO InputStream from XML(arg[0])
    // and XSL(arg[1]) filepath String
    foStream = FOUtility.createFO(args[0], args[1]);
    if (mergedFOStream == null)
    {
      System.out.println("Merge failed.");
      System.exit(1);
    }

    System.exit(0);
  }
```

### 7.6.6.2 Creating XSL-FO from Two XML Files and Two XSL files

Input:

- XML File 1
- XML File 2
- XSL File 1
- XSL File 2

Output:

- XSL-FO (InputStream)

***Example 7–17 Sample Code for Creating XSL-FO from Two XML Files and Two XSL Files***

```
import oracle.xdo.template.fo.util.FOUtility;
.
.
.
  public static void main(String[] args)
  {
    InputStream firstFOStream, secondFOStream, mergedFOStream;
    InputStream[] input = InputStream[2];

    // creates XSL-FO from arguments
    firstFOStream = FOUtility.createFO(args[0], args[1]);

    // creates another XSL-FO from arguments
    secondFOStream = FOUtility.createFO(args[2], args[3]);

    // set each InputStream into the InputStream Array
    Array.set(input, 0, firstFOStream);
    Array.set(input, 1, secondFOStream);

    // merges two XSL-FOs
    mergedFOStream = FOUtility.mergeFOs(input);
    if (mergedFOStream == null)
    {
      System.out.println("Merge failed.");
      System.exit(1);
    }
    System.exit(0);
  }
```

#### 7.6.6.3 Merging Two XSL-FO Files

Input:

- Two XSL-FO file names (Array)

Output:

- One XSL-FO (InputStream)

***Example 7–18 Sample Code for Merging Two XSL-FO Files***

```
import oracle.xdo.template.fo.util.FOUtility;
.
.
.
  public static void main(String[] args)
  {
    InputStream mergedFOStream;

    // creates Array
    String[] input = {args[0], args[1]};

    // merges two FO files
    mergedFOStream = FOUtility.mergeFOs(input);
    if (mergedFOStream == null)
    {
      System.out.println("Merge failed.");
      System.exit(1);
    }
    System.exit(0);
  }
```

#### 7.6.6.4 Generating Output from an FO File

The FO Processor can also be used to process an FO object to generate your final output. An FO object is the result of the application of an XSL-FO style sheet to XML data. These objects can be generated from a third party application and fed as input to the FO Processor.

The processor is called using a similar method to those already described, but a template is not required as the formatting instructions are contained in the FO.

#### 7.6.6.5 Generating Output Using File Names

Input:

- FO file name (String)

Output:

- PDF file name (String)

***Example 7–19 Sample Code for Generating Output Using File Names***

```
import oracle.xdo.template.FOProcessor;
.
.
.
  public static void main(String[] args) {

    FOProcessor processor = new FOProcessor();
    processor.setData(args[0]);     // set XSL-FO input file
    processor.setTemplate((String)null);
    processor.setOutput(args[2]);  //set (PDF) output file
    processor.setOutputFormat(FOProcessor.FORMAT_PDF);
    // Start processing
    try
    {
       processor.generate();
    }
    catch (XDOException e)
    {
       e.printStackTrace();
       System.exit(1);
    }

    System.exit(0);
  }
```

### 7.6.6.6 Generating Output Using Streams

Input:

- FO data (InputStream)

Output:

- Output (OutputStream)

***Example 7–20 Sample Code for Generating Output Using Streams***

```
import java.io.InputStream;
import java.io.OutputStream;
import oracle.xdo.template.FOProcessor;
.
.
.
  public void runFOProcessor(InputStream xmlfoInputStream,
                             OutputStream pdfOutputStream)
  {

    FOProcessor processor = new FOProcessor();
    processor.setData(xmlfoInputStream);
    processor.setTemplate((String)null);

    processor.setOutput(pdfOutputStream);
    // Set output format (for PDF generation)
    processor.setOutputFormat(FOProcessor.FORMAT_PDF);
    // Start processing
    try
    {
       processor.generate();
    }
    catch (XDOException e)
    {
       e.printStackTrace();
       System.exit(1);
    }
  }
```

#### 7.6.6.7 Generating Output with an Array of FO Data

Pass multiple FO inputs as an array to generate a single output file. A template is not required, therefore set the members of the template array to null, as shown in the example.

Input:

- FO data (Array)

Output:

- Output File Name (String)

***Example 7–21 Sample Code for Generating Output with an Array of FO Data***

```
import java.lang.reflect.Array;
import oracle.xdo.template.FOProcessor;
.
.
.
  public static void main(String[] args)
  {

    String[] xmlInput = {"first.fo", "second.fo", "third.fo"};
    String[] xslInput = {null, null, null};   // null needs for xsl-fo input

    FOProcessor processor = new FOProcessor();
    processor.setData(xmlInput);
    processor.setTemplate(xslInput);

    processor.setOutput("/tmp/output.pdf);          //set (PDF) output file
    processor.setOutputFormat(FOProcessor.FORMAT_PDF); processor.process();
    // Start processing
    try
    {
       processor.generate();
    }
    catch (XDOException e)
    {
       e.printStackTrace();
       System.exit(1);
    }

  }
```

## 7.7 PDF Document Merger

The PDF Document Merger class provides a set of utilities to manipulate PDF documents. Using these utilities, you can merge documents, add page numbering, set backgrounds, and add watermarks.

## 7.7.1 Merging PDF Documents

Many business documents are composed of several individual documents that need to be merged into a single final document. The PDFDocMerger class supports the merging of multiple documents to create a single PDF document. This can then be manipulated further to add page numbering, watermarks, or other background images.

### 7.7.1.1 Merging PDF Documents with Input/Output File Names

The following code demonstrates how to merge (concatenate) two PDF documents using physical files to generate a single output document.

Input:

- PDF_1 file name (String)
- PDF_2 file name (String)

Output:

- PDF file name (String)

***Example 7–22 Sample Code for Merging PDF Documents with Input/Output File Names***

```
import java.io.*;
import oracle.xdo.common.pdf.util.PDFDocMerger;
.
.
.
  public static void main(String[] args)
  {
    try
    {
      // Last argument is PDF file name for output
      int inputNumbers = args.length - 1;

      // Initialize inputStreams
      FileInputStream[] inputStreams = new FileInputStream[inputNumbers];
      inputStreams[0] = new FileInputStream(args[0]);
      inputStreams[1] = new FileInputStream(args[1]);

      // Initialize outputStream
      FileOutputStream outputStream = new FileOutputStream(args[2]);

      // Initialize PDFDocMerger
      PDFDocMerger docMerger = new PDFDocMerger(inputStreams, outputStream);

      // Merge PDF Documents and generates new PDF Document
      docMerger.mergePDFDocs();
      docMerger = null;

      // Closes inputStreams and outputStream
    }
    catch(Exception exc)
    {
      exc.printStackTrace();
    }
  }
```

#### 7.7.1.2 Merging PDF Documents with Input/Output Streams

Input:

- PDF Documents (InputStream Array)

Output:

- PDF Document (OutputStream)

***Example 7–23 Merging PDF Documents with Input/Output Streams***

```
import java.io.*;
import oracle.xdo.common.pdf.util.PDFDocMerger;
.
.
.
  public boolean mergeDocs(InputStream[] inputStreams, OutputStream outputStream)
  {
    try
    {
      // Initialize PDFDocMerger
      PDFDocMerger docMerger = new PDFDocMerger(inputStreams, outputStream);

      // Merge PDF Documents and generates new PDF Document
      docMerger.mergePDFDocs();
      docMerger = null;

      return true;
    }
    catch(Exception exc)
    {
      exc.printStackTrace();
      return false;
    }
  }
```

#### 7.7.1.3 Merging with Background to Place Page Numbering

The following code demonstrates how to merge two PDF documents using input streams to generate a single merged output stream.

To add page numbers:

1. Create a background PDF template document that includes a PDF form field in the position that you would like the page number to appear on the final output PDF document.
2. Name the form field `@pagenum@`.
3. Enter the number in the field from which to start the page numbering. If you do not enter a value in the field, the start page number defaults to 1.

Input:

- PDF Documents (InputStream Array)
- Background PDF Document (InputStream)

Output:

- PDF Document (OutputStream)

***Example 7–24 Sample Code for Merging PDF Documents with Background to Place Page Numbering***

```
import java.io.*;
import oracle.xdo.common.pdf.util.PDFDocMerger;
.
.
.
 public static boolean mergeDocs(InputStream[] inputStreams, InputStream
backgroundStream, OutputStream outputStream)

 {
    try
    {
      // Initialize PDFDocMerger
      PDFDocMerger docMerger = new PDFDocMerger(inputStreams, outputStream);

      // Set Background
      docMerger.setBackground(backgroundStream);

      // Merge PDF Documents and generates new PDF Document
      docMerger.mergePDFDocs();
      docMerger = null;

      return true;
    }
    catch(Exception exc)
    {
      exc.printStackTrace();
      return false;
    }
  }
```

### 7.7.1.4 Adding Page Numbers to Merged Documents

The FO Processor supports page numbering natively through the XSL-FO templates, but if you are merging multiple documents you must use this class to number the complete document from beginning to end.

The following code example places page numbers in a specific point on the page, formats the numbers, and sets the start value using the following methods:

- setPageNumberCoordinates (x, y) - sets the x and y coordinates for the page number position. The following example sets the coordinates to 300, 20.
- setPageNumberFontInfo (font name, size) - sets the font and size for the page number. If you do not call this method, the default "Helvetica", size 8 is used. The following example sets the font to "Courier", size 8.
- setPageNumberValue (n, n) - sets the start number and the page on which to begin numbering. If you do not call this method, the default values 1, 1 are used.

Input:

- PDF Documents (InputStream Array)

Output:

- PDF Document (OutputStream)

***Example 7–25 Sample Code for Adding Page Numbers to Merged PDF Documents***

```
import java.io.*;
import oracle.xdo.common.pdf.util.PDFDocMerger;
.
.
.
  public boolean mergeDocs(InputStream[] inputStreams, OutputStream outputStream)
  {
    try
    {
      // Initialize PDFDocMerger
      PDFDocMerger docMerger = new PDFDocMerger(inputStreams, outputStream);

      // Calls several methods to specify Page Number

      // Calling setPageNumberCoordinates() method is necessary to set Page
Numbering
      // Please refer to javadoc for more information
      docMerger.setPageNumberCoordinates(300, 20);


      // If this method is not called, then the default font"(Helvetica, 8)" is
used.
      docMerger.setPageNumberFontInfo("Courier", 8);

      // If this method is not called, then the default initial value "(1, 1)" is
used.
      docMerger.setPageNumberValue(1, 1);

      // Merge PDF Documents and generates new PDF Document
      docMerger.mergePDFDocs();
      docMerger = null;

      return true;
    }
    catch(Exception exc)
    {
      exc.printStackTrace();
      return false;
    }
  }
```

## 7.7.2 Setting a Text or Image Watermark

Some documents that are in a draft phase require that a watermark indicating "DRAFT" be displayed throughout the document. Other documents might require a background image on the document. The following code sample shows how to use the PDFDocMerger class to set a watermark.

### 7.7.2.1 Setting a Text Watermark

Use the SetTextDefaultWatermark() method to set a text watermark with the following attributes:

- Text angle (in degrees): 55
- Color: light gray (0.9, 0.9, 0.9)
- Font: Helvetica

- Font Size: 100
- The start position is calculated based on the length of the text

Alternatively, use the SetTextWatermark() method to set each attribute separately. Use the SetTextWatermark() method as follows:

- SetTextWatermark ("Watermark Text", x, y) - declare the watermark text, and set the x and y coordinates of the start position. In the following example, the watermark text is "Draft" and the coordinates are 200f, 200f.
- setTextWatermarkAngle (n) - sets the angle of the watermark text. If this method is not called, 0 will be used.
- setTextWatermarkColor (R, G, B) - sets the RGB color. If this method is not called, light gray (0.9, 0.9, 0.9) will be used.
- setTextWatermarkFont ("font name", font size) - sets the font and size. If you do not call this method, Helvetica, 100 will be used.

The following example shows how to set these properties and then call the PDFDocMerger.

Input:

- PDF Documents (InputStream)

Output:

- PDF Document (OutputStream)

***Example 7–26 Sample Code for Setting a Text Watermark in PDF Documents***

```
import java.io.*;
import oracle.xdo.common.pdf.util.PDFDocMerger;
.
.
.
  public boolean mergeDocs(InputStream inputStreams, OutputStream outputStream)
  {
    try
    {
      // Initialize PDFDocMerger
      PDFDocMerger docMerger = new PDFDocMerger(inputStreams, outputStream);

      // You can use setTextDefaultWatermark() without these detailed setting
      docMerger.setTextWatermark("DRAFT", 200f, 200f); //set text and place
      docMerger.setTextWatermarkAngle(80);              //set angle
      docMerger.setTextWatermarkColor(1.0f, 0.3f, 0.5f);  // set RGB Color

      // Merge PDF Documents and generates new PDF Document
      docMerger.mergePDFDocs();
      docMerger = null;

      return true;
    }
    catch(Exception exc)
    {
      exc.printStackTrace();
      return false;
    }
  }
```

#### 7.7.2.2 Setting Image Watermark

An image watermark can be set to cover the entire background of a document, or just to cover a specific area (for example, to display a logo). Specify the placement and size of the image using rectangular coordinates as follows:

```
float[] rct = {LowerLeft X, LowerLeft Y, UpperRight X,
UpperRight Y}
```

For example:

```
float[] rct = {100f, 100f, 200f, 200f}
```

The image will be sized to fit the rectangular area defined.

To use the actual image size, without sizing it, define the LowerLeft X and LowerLeft Y positions to define the placement and specify the UpperRight X and UpperRight Y coordinates as -1f. For example:

```
float[] rct = {100f, 100f, -1f, -1f}
```

Input:

- PDF Documents (InputStream)
- Image File (InputStream)

Output:

- PDF Document (OutputStream)

***Example 7–27 Sample Code for Setting an Image Watermark in PDF Documents***

```
import java.io.*;
import oracle.xdo.common.pdf.util.PDFDocMerger;
.
.
.
  public boolean mergeDocs(InputStream inputStreams, OutputStream outputStream,
String imageFilePath)
  {
    try
    {
      // Initialize PDFDocMerger
      PDFDocMerger docMerger = new PDFDocMerger(inputStreams, outputStream);

      FileInputStream wmStream = new FileInputStream(imageFilePath);
      float[] rct = {100f, 100f, -1f, -1f};
      pdfMerger.setImageWatermark(wmStream, rct);

      // Merge PDF Documents and generates new PDF Document
      docMerger.mergePDFDocs();
      docMerger = null;

      // Closes inputStreams
      return true;
    }
    catch(Exception exc)
    {
      exc.printStackTrace();
      return false;
    }
  }
```

## 7.8 PDF Bookbinder Processor

The PDFBookBinder processor is useful for the merging of multiple PDF documents into a single document consisting of a hierarchy of chapters, sections, and subsections and a table of contents for the document. The processor also generates PDF style "bookmarks"; the outline structure is determined by the chapter and section hierarchy. The processor is extremely powerful allowing you complete control over the combined document.

### 7.8.1 Usage

The table of contents formatting and style is defined by an RTF template created in Microsoft Word. The chapters are passed into the program as separate PDF files (one chapter, section, or subsection corresponds to one PDF file). Templates may also be specified at the chapter level for insertion of dynamic or static content, page numbering, and placement of hyperlinks within the document.

The templates can be in RTF or PDF format. RTF templates are more flexible by allowing you to leverage BI Publisher's support for dynamic content. PDF templates are much less flexible, making it difficult to achieve desirable effects such as the reflow of text areas when inserting page numbers and other types of dynamic content.

The templates can be rotated (at right angles) or be made transparent. A PDF template can also be specified at the book level, enabling the ability to specify global page numbering, or other content such as backgrounds and watermarks. You can also pass as parameters a title page, cover page, and closing pages for each chapter or section.

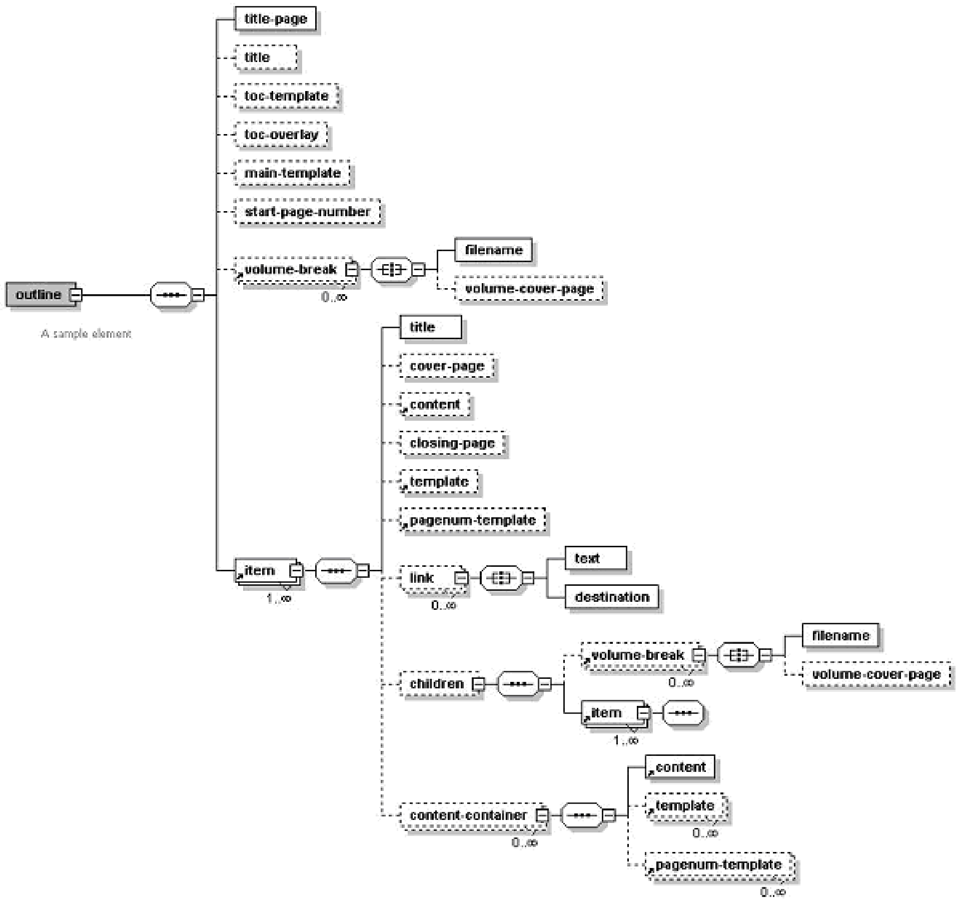
### 7.8.2 XML Control File

The structure of the book's chapters, sections, and subsections is represented as XML and passed in as a command line parameter; or it can also be passed in at the API level. All of the chapter and section files, and all the templates files and their respective parameters, are specified inside this XML structure. Therefore, the only two required parameters are an XML file and a PDF output file.

You can also specify volume breaks inside the book structure. Specifying volume breaks will split the content up into separate output files for easier file and printer management.

The structure of the XML control file is represented in the following diagram:

***Figure 7–5 Structure of XML Control File***



To specify template and content file locations in your XML structure, you can specify a path relative to your local file system or you can specify a URL referring to the template or content location. Secure HTTP protocol is supported, as are specially recognized BI Publisher protocols, such as:

- xdoxsl:/// used to load subtemplate.
- "xdo://" - used to load other resources such as images.

## 7.8.3 Command Line Options

Following is an example of the command line usage:

***Example 7–28 Sample of Command Line Options***

```
java oracle.xdo.template.pdf.book.PDFBookBinder [-debug <true or false>] [-tmp
<temp dir>] -xml <input xml> -pdf <output pdf>
```

where

`-xml <file>` is the file name of the input XML file containing the table of contents XML structure.

`-pdf <file>` is the final generated PDF output file.

`-tmp <directory>` is the temporary directory for better memory management. (This is optional, if not specified, the system environment variable `"java.io.tmpdir"` will be used.)

`-log <file>` sets the output log file (optional, default is `System.out`).

`-debug <true or false>` turns debugging off or on.

### 7.8.4 API Method Call

The following is an example of an API method call:

***Example 7–29 Sample API Method Call***

```
String xmlInputPath = "c:\\tmp\\toc.xml";
String pdfOutputPath = "c:\\tmp\\final_book.pdf";
PDFBookBinder bookBinder = new PDFBookBinder(xmlInputPath,
 pdfOutputPath);

bookBinder.setConfig(new Properties());
bookBinder.process();
```

## 7.9 PDF Digital Signature Engine

This section discusses how to use the PDF Digital Signature Engine, and includes the following topics:

- Section 7.9.1, "Overview of the PDF Digital Signature Engine"
- Section 7.9.2, "Signing PDF Documents"
- Section 7.9.3, "Delivering Signed PDF Documents."
- Section 7.9.4, "Verifying Signed PDF Documents"

### 7.9.1 Overview of the PDF Digital Signature Engine

The PDF Digital Signature Engine creates signed PDF documents by processing unsigned PDF documents with a signature field name and a password-protected Personal Information Exchange (PFX) file. PFX files adhere to the Public Key Cryptography Standards #12 (PCKS-12) format and contain a digital certificate and a corresponding private key.

To create signed PDF documents, see Section 7.9.2, "Signing PDF Documents."

To distribute or deliver signed PDF documents, use the Schedule Service. See Chapter 3, "ScheduleService."

To verify signed PDF documents, see Section 7.9.4, "Verifying Signed PDF Documents."

### 7.9.2 Signing PDF Documents

Signing a PDF document requires the following items:

- A digital certificate. To obtain a digital certificate, see "Implementing Digital Signatures" in *Oracle Fusion Middleware Developer's Guide for Oracle Business Intelligence Publisher*.
- A PFX file that contains your digital certificate.To create a PFX file, see "Implementing Digital Signatures" in *Oracle Fusion Middleware Developer's Guide for Oracle Business Intelligence Publisher*.

- A PDF file with a signature field. If your PDF file does not contain a signature field, you can add one using the addSignatureField() method. Example 7–30 provides sample code for adding a signature field to a PDF file. This method saves the signature field in the PKCS-1 Secure Hash Algorithm #1 (SHA-1) format.

After you obtain or create the items listed above, you are ready to sign a PDF document.

To sign a PDF document, process your PDF file with your PFX file using the PDFSignature Java API that Oracle BI Publisher provides. Example 7–30 provides sample code for this purpose.

***Example 7–30 Sample Code for Creating Signed PDF Documents***

```
String workDir = "C:/projects/";
    String inPDF = workDir + "VerySimpleContent.pdf";
    String outPDF = workDir + "VerySimpleContent_signed.pdf";
    String pkcs12File = workDir + "YourName.pfx";

    try
    {
      PDFSignature pdfSignature = new PDFSignature(inPDF, outPDF);
      pdfSignature.init("password123", pkcs12File);
      // If your PDF document does not have a signature field, uncomment the
      // following line of code, which adds a signature field with the name
      // "Signature1".
      // pdfSignature.addSignatureField(1, PDFSignature.PDF_SIGNFIELD_UPPER_RIGHT,
"Signature1", -1, -1);
      pdfSignature.sign("Signature1", "Reason to Sign");
      pdfSignature.cleanup();
    }
    catch(Throwable t)
    {
      t.printStackTrace();
    }
```

## 7.9.3 Delivering Signed PDF Documents

To distribute or deliver signed PDF documents, use the Schedule Service. See Chapter 3, "ScheduleService."

## 7.9.4 Verifying Signed PDF Documents

You can verify signed PDF documents by processing them with your digital certificate. Example 7–31 provides sample code for this purpose.

***Example 7–31 Sample Code for Verifying Signed PDF Documents***

```
    String workDir = "C:/projects/";
    String inPDF = workDir + "VerySimpleContent_signedWithAcrobat.pdf";
    String trustedRootCert = workDir + "VerisignFreeCertificate.cer";
    File trustedRootCertFile = new File(trustedRootCert);
    Vector trustedCerts = new Vector();
    trustedCerts.addElement(trustedRootCertFile);

    try
    {
      PDFSignature pdfSignature = new PDFSignature(inPDF);
      pdfSignature.init();
      SignatureFields signFields = pdfSignature.getSignatureFields();
      Vector signedFieldNames = signFields.getSignatureFieldNames();
      int size = signedFieldNames.size();
      for(int i = 0 ; i < size ; i++)
      {
        String signFieldName = (String)signedFieldNames.elementAt(i);
        SignatureField signField = signFields.getSignatureField(signFieldName);
        boolean isValid = signField.verifyDocument();
        System.out.println("Valid? : " + isValid);
        boolean isCertValid = signField.verifyCertificates(trustedCerts, null);
        System.out.println("Trusted? : " + isCertValid);
      }
    }
    catch(Throwable t)
    {
      t.printStackTrace();
    }
```

## 7.10 eText Processor

The eText Processor enables you to convert RTF eText templates to XSL, and merge the XSL with XML to produce text output for EDI and EFT transmissions.

### 7.10.1 Converting RTF eText Templates to XSL

The following is an example of an API method call that converts an RTF eText template to XSL:

***Example 7–32 Sample Code for Converting RTF eText Templates to XSL***

```
String rtf = "test.rtf"; // etext template filename
String xsl = "out.xsl"; //  xsl-fo filename
Properties prop = new Properties();

try
     {
         EFTProcessor p = new EFTProcessor();

         p.setTemplate(rtf);
   p.setConfig(prop);
         p.setOutput(xsl);
         p.process();
     }
     catch (Exception e)
     {
       e.printStackTrace();
     }
```

### 7.10.2 Producing Text Output for EDI and EFT Transmissions

The following is an example of an API method call that merges XSL with XML to produce eText output:

***Example 7–33 Sample Code for Producing Text Output for EDI and EFT Transmissions***

```
String rtf = "test.rtf"; // etext template filename
String xml = "data.xml"; //xml data filename
String etext = "etext.txt"; //etext output filename
Properties prop = new Properties();
```

```
try
     {
         EFTProcessor p = new EFTProcessor();
         p.setConfig(prop);
         p.setTemplate(rtf);
         p.setData(xml);
         p.setOutput(etext);
         p.process();
     }
     catch (Exception e)
     {
       e.printStackTrace();
     }
```

### 7.10.3 Getting Sequence Numbers

The following APIs enable you to retrieve sequence numbers for eText templates:

- `EFTProcessor.getPeriodicSequenceNumbers()` - returns a hashtable that contains the sequence names and last sequence number of the sequence.
- `EFTProcessor.getPeriodicSequenceNames()` - returns a vector that contains all the PERIODIC_SEQUENCE sequence names defined in the eText template.

The following example shows how to use these APIs:

***Example 7–34 Sample Code for Getting Sequence Numbers***

```
try{
   EFTProcessor eftp = new EFTProcessor();
   eftp.setConfig(prop); // set property
   eftp.setTemplate(rtf); // set template
   eftp.setOutput(etext); // set etext output
   eftp.setXSL(xsl);
   eftp.setData(xml); // set xml data
   eftp.process();

// Vector sequenceNames = eftp.getPeriodicSequenceNames();
// get periodic sequence names
Hashtable seqNumbers = eftp.getPeriodicSequenceNumbers();
// get periodic sequence numbers
   Enumeration keys = seqNumbers.keys();
   while (keys.hasMoreElements()) {
         String key = keys.nextElement().toString();
         String value = seqNumbers.get(key).toString();
         System.out.println("EFT test: key=" + key + " value=" + value);
        }
  }
   catch( Exception e)
   {
         e.printStackTrace();
  }
```

## 7.11 Document Processor Engine

The Document Processor Engine provides batch processing functionality to access a single API or multiple APIs by passing a single XML instance document to specify template names, data sources, languages, output type, output names, and destinations.

This solution enables batch printing with BI Publisher, in which a single XML document can be used to define a set of invoices for customers, including the preferred output format and delivery channel for those customers. The XML format is very flexible allowing multiple documents to be created or a single master document.

This section:

- Describes the hierarchy and elements of the Document Processor XML file
- Provides sample XML files to demonstrate specific processing options
- Provides example code to invoke the processors

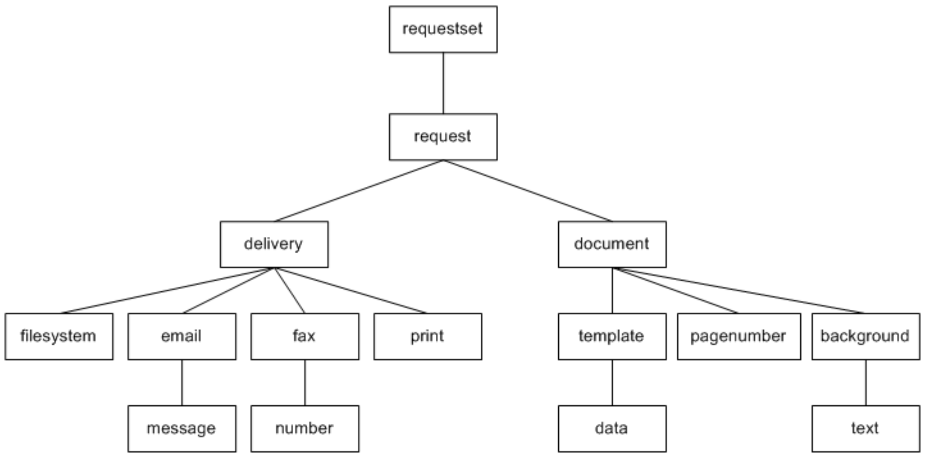
## 7.11.1 Hierarchy and Elements of the Document Processor XML File

The Document Processor XML file has the following element hierarchy:

```
Requestset
   request
      delivery
         filesystem
         print
         fax
            number
         email
            message
      document
         background
            text
         pagenumber
         template
            data
```

This hierarchy is displayed in the following illustration:

***Figure 7–6 Hierarchy and Elements of the Document Processor XML File***



The following table describes each of the elements:

*Table 7–1 Elements in Document Processor XML File Hierarchy*

| Element | Attributes | Description |
|---|---|---|
| `requestset` | `xmlns`<br>`version` | Root element must contain [`xmlns:xapi="http://xmlns.oracle.com/oxp/xapi/"`] block<br>The `version` is not required, but defaults to "1.0". |
| `request` | N/A | Element that contains the data and template processing definitions. |
| `delivery` | N/A | Defines where the generated output is sent. |
| `document` | `output-type` | Specify one output that can have several template elements. The `output-type` attribute is optional. Valid values are:<br>pdf (Default)<br>rtf<br>html<br>excel<br>text |
| `filesystem` | `output` | Specify this element to save the output to the file system. Define the directory path in the `output` attribute. |
| `print` | ■ `printer`<br>■ `server-alias` | The `print` element can occur multiple times under `delivery` to print one document to several printers. Specify the `printer` attribute as a URI, such as:`"ipp://myprintserver:631/printers/printername"` |
| `fax` | ■ `server`<br>■ `server-alias` | Specify a URI in the `server` attribute, for example: `"ipp://myfaxserver1:631/printers/myfaxmachine"` |
| `number` |  | The `number` element can occur multiple times to list multiple fax numbers. Each element occurrence must contain only one number. |
| `email` | ■ `server`<br>■ `port`<br>■ `from`<br>■ `reply-to`<br>■ `server-alias` | Specify the outgoing mail server (SMTP) in the `server` attribute.<br>Specify the mail server port in the `port` attribute. |

***Table 7–1 (Cont.) Elements in Document Processor XML File Hierarchy***

| Element | Attributes | Description |
|---|---|---|
| `message` | ■ `to`<br>■ `cc`<br>■ `bcc`<br>■ `attachment`<br>■ `subject` | The `message` element can be placed several times under the `email` element. You can specify character data in the `message` element.<br>You can specify multiple e-mail addresses in the `to`, `cc`, and `bcc` attributes separated by a comma.<br>The `attachment` value is either true or false (default). If `attachment` is true, then a generated document will be attached when the e-mail is sent.<br>The `subject` attribute is optional. |
| `background` | `where` | If the background text is required on a specific page, then set the `where` value to the page numbers required. The page index starts at 1. The default value is 0, which places the background on all pages. |
| `text` | ■ `title`<br>■ `default` | Specify the watermark text in the `title` value.<br>A `default` value of "yes" automatically draws the watermark with forward slash type. The default value is yes. |
| `pagenumber` | ■ `initial-page-index`<br>■ `initial-value`<br>■ `x-pos`<br>■ `y-pos` | The `initial-page-index` default value is 0.<br>The `initial-value` default value is 1.<br>"Helvetica" is used for the page number font.<br>The`x-pos` provides lower left x position.<br>The `y-pos` provides lower left y position. |
| `template` | ■ `locale`<br>■ `location`<br>■ `type` | Contains template information.<br>Valid values for the `type` attribute are<br>pdf<br>rtf<br>xsl-fo<br>etext<br>The default value is "pdf". |

*Table 7–1 (Cont.) Elements in Document Processor XML File Hierarchy*

| Element | Attributes | Description |
|---|---|---|
| `data` | `location` | Define the `location` attribute to specify the location of the data, or attach the actual XML data with sub-elements. The default value of `location` is "inline". It the `location` points to either an XML file or a URL, then the data should contain an XML declaration with the proper encoding.<br><br>If the `location` attribute is not specified, the `data` element should contain the sub-elements for the actual data. This must not include an XML declaration. |

## 7.11.2 XML File Samples

Following are sample XML files that show:

- Simple XML shape
- Defining two data sets
- Defining multiple templates and data
- Retrieving templates over HTTP
- Retrieving data over HTTP
- Generating more than one output
- Defining page numbers

### 7.11.2.1 Defining two data sets

The following example shows how to define two data sources to merge with one template to produce one output file delivered to the file system:

***Example 7–35 Sample XML for Defining Two Data Sets***

```
<?xml version="1.0" encoding="UTF-8"?>
<xapi:requestset xmlns:xapi="http://xmlns.oracle.com/oxp/xapi">
  <xapi:request>
    <xapi:delivery>
      <xapi:filesystem output="d:\tmp\outfile.pdf"/>
    </xapi:delivery>

    <xapi:document output-type="pdf">
      <xapi:template type="pdf"
                  location="d:\mywork\template1.pdf">
        <xapi:data>
          <field1>The first set of data</field1>
        </xapi:data>
        <xapi:data>
          <field1>The second set of data</field1>
        </xapi:data>
      </xapi:template>
    </xapi:document>
  </xapi:request>
</xapi:requestset>
```

#### 7.11.2.2 Defining multiple templates and data

The following example builds on the previous examples by applying two data sources to one template and two data sources to a second template, and then merging the two into a single output file. Note that when merging documents, the `output-type` must be "pdf".

***Example 7–36 Sample XML for Defining Multiple Templates and Data***

```
<?xml version="1.0" encoding="UTF-8"?>
<xapi:requestset xmlns:xapi="http://xmlns.oracle.com/oxp/xapi">
  <xapi:request>
    <xapi:delivery>
      <xapi:filesystem output="d:\tmp\outfile3.pdf"/>
    </xapi:delivery>

    <xapi:document output-type="pdf">
      <xapi:template type="pdf"
                  location="d:\mywork\template1.pdf">
        <xapi:data>
          <field1>The first set of data</field1>
        </xapi:data>
        <xapi:data>
          <field1>The second set of data</field1>
        </xapi:data>
      </xapi:template>

      <xapi:template type="pdf"
                  location="d:\mywork\template2.pdf">
        <xapi:data>
          <field1>The third set of data</field1>
        </xapi:data>
        <xapi:data>
          <field1>The fourth set of data</field1>
        </xapi:data>
      </xapi:template>
    </xapi:document>
  </xapi:request>
</xapi:requestset>
```

#### 7.11.2.3 Retrieving templates over HTTP

This sample is identical to the previous example, except in this case the two templates are retrieved over HTTP:

***Example 7–37 Sample XML for Retrieving Templates over HTTP***

```
<?xml version="1.0" encoding="UTF-8"?>
<xapi:requestset xmlns:xapi="http://xmlns.oracle.com/oxp/xapi">
  <xapi:request>
    <xapi:delivery>
      <xapi:filesystem output="d:\temp\out4.pdf"/>
    </xapi:delivery>

    <xapi:document output-type="pdf">
      <xapi:template type="pdf"
       location="http://your.server:9999/templates/template1.pdf">
        <xapi:data>
```

```
          <field1>The first page data</field1>
        </xapi:data>
        <xapi:data>
          <field1>The second page data</field1>
        </xapi:data>
      </xapi:template>
      <xapi:template type="pdf"
       location="http://your.server:9999/templates/template2.pdf">
        <xapi:data>
          <field1>The third page data</field1>
        </xapi:data>
        <xapi:data>
          <field1>The fourth page data</field1>
        </xapi:data>
      </xapi:template>
    </xapi:document>
  </xapi:request>
</xapi:requestset>
```

### 7.11.2.4 Retrieving data over HTTP

This sample builds on the previous example and shows one template with two data sources, all retrieved through HTTP; and a second template retrieved through HTTP with its two data sources embedded in the XML:

***Example 7–38 Sample XML for Retrieving Data over HTTP***

```
<?xml version="1.0" encoding="UTF-8"?>
<xapi:requestset xmlns:xapi="http://xmlns.oracle.com/oxp/xapi">
 <xapi:request>
  <xapi:delivery>
   <xapi:filesystem output="d:\temp\out5.pdf"/>
    </xapi:delivery>

    <xapi:document output-type="pdf">
     <xapi:template type="pdf"
      location="http://your.server:9999/templates/template1.pdf">
      <xapi:data location="http://your.server:9999/data/data_1.xml"/>
      <xapi:data location="http://your.server:9999/data/data_2.xml"/>
      </xapi:template>

      <xapi:template type="pdf"
       location="http://your.server:9999/templates/template2.pdf">
        <xapi:data>
          <field1>The third page data</field1>
        </xapi:data>
        <xapi:data>
          <field1>The fourth page data</field1>
        </xapi:data>
      </xapi:template>
    </xapi:document>
  </xapi:request>
</xapi:requestset>
```

### 7.11.2.5 Generating more than one output

The following sample shows the generation of two outputs: `out_1.pdf` and `out_2.pdf`. Note that a `request` element is defined for each output.

***Example 7–39 Sample XML for Generating More than One Output***

```
<?xml version="1.0" encoding="UTF-8"?>
<xapi:requestset xmlns:xapi="http://xmlns.oracle.com/oxp/xapi">
  <xapi:request>
    <xapi:delivery>
      <xapi:filesystem output="d:\temp\out_1.pdf"/>
    </xapi:delivery>
    <xapi:document output-type="pdf">
      <xapi:template type="pdf"
                  location="d:\mywork\template1.pdf">
        <xapi:data>
          <field1>The first set of data</field1>
        </xapi:data>
        <xapi:data>
          <field1>The second set of data</field1>
        </xapi:data>
      </xapi:template>
    </xapi:document>
  </xapi:request>

  <xapi:request>
    <xapi:delivery>
      <xapi:filesystem output="d:\temp\out_2.pdf"/>
    </xapi:delivery>
    <xapi:document output-type="pdf">
      <xapi:template type="pdf"
                  location="d:mywork\template2.pdf">
        <xapi:data>
          <field1>The third set of data</field1>
        </xapi:data>
        <xapi:data>
          <field1>The fourth set of data</field1>
        </xapi:data>
      </xapi:template>
    </xapi:document>
  </xapi:request>

</xapi:requestset>
```

### 7.11.2.6 Defining page numbers

The following sample shows the use of the `pagenumber` element to define page numbers on a PDF output document. The first document that is generated will begin with an initial page number value of 1. The second output document will begin with an initial page number value of 3. The `pagenumber` element can reside anywhere within the `document` element tags.

Note that page numbering that is applied using the `pagenumber` element will not replace page numbers that are defined in the template.

***Example 7–40 Sample XML for Defining Page Numbers***

```
<?xml version="1.0" encoding="UTF-8"?>
<xapi:requestset xmlns:xapi="http://xmlns.oracle.com/oxp/xapi">
  <xapi:request>
    <xapi:delivery>
      <xapi:filesystem output="d:\temp\out7-1.pdf"/>
    </xapi:delivery>
    <xapi:document output-type="pdf">
```

```
        <xapi:pagenumber initial-value="1" initial-page-index="1"
         x-pos="300" y-pos="20" />
        <xapi:template type="pdf"
          location="d:\mywork\template1.pdf">
          <xapi:data>
            <field1>The first page data</field1>
          </xapi:data>
          <xapi:data>
            <field1>The second page data</field1>
          </xapi:data>
        </xapi:template>
      </xapi:document>
    </xapi:request>

    <xapi:request>
      <xapi:delivery>
        <xapi:filesystem output="d:\temp\out7-2.pdf"/>
      </xapi:delivery>
      <xapi:document output-type="pdf">
        <xapi:template type="pdf"
             location="d:\mywork\template2.pdf">
          <xapi:data>
            <field1>The third page data</field1>
          </xapi:data>
          <xapi:data>
            <field1>The fourth page data</field1>
          </xapi:data>
        </xapi:template>
        <xapi:pagenumber initial-value="3" initial-page-index="1"
         x-pos="300" y-pos="20" />
      </xapi:document>
    </xapi:request>

  </xapi:requestset>
```

### 7.11.3 Invoke Processors

The following code samples show how to invoke the document processor engine using an input file name and an input stream.

#### 7.11.3.1 Invoking Processors with Input File Name

Input:

- Data file name (String)
- Directory for Temporary Files (String)

***Example 7–41 Sample Code for Invoking Processors with Input File Names***

```
import oracle.xdo.batch.DocumentProcessor;
.
.
.
  public static void main(String[] args)
  {
.
.
.
    try
    {
      // dataFile --- File path of the Document Processor XML
      // tempDir  --- Temporary Directory path
      DocumentProcessor  docProcessor = new DocumentProcessor(dataFile, tempDir);
      docProcessor.process();
    }
    catch(Exception e)
    {
e.printStackTrace();
       System.exit(1);
    }
    System.exit(0);
  }
```

#### 7.11.3.2 Invoking Processors with InputStream

Input:

- Data file (InputStream)
- Directory for Temporary Files (String)

***Example 7–42 Sample Code for Invoking Processors with Input Streams***

```
import oracle.xdo.batch.DocumentProcessor;
import java.io.InputStream;
.
.
.
  public static void main(String[] args)
  {
.
.
.
    try
    {
      // dataFile --- File path of the Document Processor XML
      // tempDir  --- Temporary Directory path
      FileInputStream fIs = new FileInputStream(dataFile);

      DocumentProcessor  docProcessor = new DocumentProcessor(fIs, tempDir);
      docProcessor.process();
      fIs.close();
    }
    catch(Exception e)
    {
e.printStackTrace();
       System.exit(1);
    }
    System.exit(0);
  }
```

## 7.12 BI Publisher Properties

The FO Processor supports PDF security and other properties that can be applied to your final documents. Security properties include making a document unprintable and applying password security to an encrypted document.

Other properties allow you to define font subsetting and embedding. If your template uses a font that would not normally be available to BI Publisher at run time, you can use the font properties to specify the location of the font. At run time BI Publisher will retrieve and use the font in the final document. For example, this property might be used for check printing for which a MICR font is used to generate the account and routing numbers on the checks.

### 7.12.1 Setting Properties

The properties can be set in the following ways:

- At run time, specify the property as a Java Property object to pass to the FO Processor.
- Set the property in a configuration file.
- Set the property in the template (RTF templates only).

#### 7.12.1.1 Passing Properties to the FO Engine

To pass a property as a Property object, set the name/value pair for the property before calling the FO Processor, as shown in the following example:

Input:

- XML file name (String)
- XSL file name (String)

Output:

- PDF file name (String)

***Example 7–43 Sample Code for Passing Properties to the FO Engine***

```
import oracle.xdo.template.FOProcessor;
.
.
.
  public static void main(String[] args)
  {

    FOProcessor processor = new FOProcessor();
    processor.setData(args[0]);     // set XML input file
    processor.setTemplate(args[1]); // set XSL input file
    processor.setOutput(args[2]);  //set (PDF) output file
    processor.setOutputFormat(FOProcessor.FORMAT_PDF);
    Properties prop = new Properties();
    /* PDF Security control: */
    prop.put("pdf-security", "true");
    /* Permissions password: */
    prop.put("pdf-permissions-password", "abc");
    /* Encryption level: */
    prop.put("pdf-encription-level", "0");
    processor.setConfig(prop);
    // Start processing
    try
    {
       processor.generate();
    }
    catch (XDOException e)
    {
       e.printStackTrace();
       System.exit(1);
    }

    System.exit(0);
  }
```

### 7.12.1.2 Passing a Configuration File to the FO Processor

The following code shows an example of passing the location of a configuration file.

Input:

- XML file name (String)
- XSL file name (String)

Output:

- PDF file name (String)

***Example 7–44 Sample Code for Passing a Configuration File to the FO Processor***

```
import oracle.xdo.template.FOProcessor;
.
.
.
  public static void main(String[] args)
  {
    FOProcessor processor = new FOProcessor();
    processor.setData(args[0]);     // set XML input file
 processor.setTemplate(args[1]); // set XSL input file
    processor.setOutput(args[2]);  //set (PDF) output file
    processor.setOutputFormat(FOProcessor.FORMAT_PDF);
    processor.setConfig("/tmp/xmlpconfig.xml");
    // Start processing
    try
    {
       processor.generate();
    }    catch (XDOException e)
    {       e.printStackTrace();
            System.exit(1);
    }
      System.exit(0);
  }
```

### 7.12.1.3 Passing Properties to the Document Processor

Input:

- Data file name (String)
- Directory for Temporary Files (String)

Output:

- PDF FIle

***Example 7–45 Sample Code for Passing Properties to the Document Processor***

```
import oracle.xdo.batch.DocumentProcessor;
.
.
.
  public static void main(String[] args)
  {
.
.
.
    try
    {
      // dataFile --- File path of the Document Processor XML
      // tempDir  --- Temporary Directory path
      DocumentProcessor  docProcessor = new DocumentProcessor(dataFile, tempDir);
      Properties prop = new Properties();
      /* PDF Security control: */
      prop.put("pdf-security", "true");
      /* Permissions password: */
      prop.put("pdf-permissions-password", "abc");
      /* encryption level: */
      prop.put("pdf-encription-level", "0");
      processor.setConfig(prop);
      docProcessor.process();
    }
    catch(Exception e)
    {
e.printStackTrace();
       System.exit(1);
    }
    System.exit(0);
  }
```

## 7.13 Advanced Barcode Formatting

For the advanced formatting to work in the template, you must provide a Java class with the appropriate methods to format the data at run time. Many font vendors offer the code with their fonts to perform the formatting; these must be incorporated as methods into a class that is available to the BI Publisher formatting libraries at run time. There are some specific interfaces that you must provide in the class for the library to call the correct method for encoding.

If you use one of the three barcodes provided with BI Publisher, you do not need to provide the Java class. For more information see "Using the Barcode Fonts Shipped with BI Publisher" in the *Oracle Fusion Middleware Report Designer's Guide for Oracle Business Intelligence Publisher*.

You must implement the following methods in this class:

```
/**
 * Return a unique ID for this barcode encoder
 * @return the id as a string
 */
  public String getVendorID();

/**
 * Return true if this encoder support a specific type of barcode
 * @param type the type of the barcode
 * @return true if supported
 */
  public boolean isSupported(String type);

/**
 * Encode a barcode string by given a specific type
 * @param data the original data for the barcode
 * @param type the type of the barcode
 * @return the formatted data
 */
  public String encode(String data, String type);
```

Place this class in the classpath for the middle tier JVM in which BI Publisher is running.

For E-Business Suite users, the class must be placed in the classpath for the middle tier and any concurrent nodes that are present.

If in the register-barcode-vendor command the `barcode_vendor_id` is not provided, BI Publisher will call the getVendorID() and use the result of the method as the ID for the vendor.

The following is an example class that supports the code128 a, b, and c encodings:

The following code sample can be copied and pasted for use in your system. Note that due to publishing constraints you will need to correct line breaks and ensure that you delete quotes that display as "smart quotes" and replace them with simple quotes.

***Example 7–46 Sample Code for Advanced Barcode Formatting***

```
package oracle.xdo.template.rtf.util.barcoder;

import java.util.Hashtable;
import java.lang.reflect.Method;
import oracle.xdo.template.rtf.util.XDOBarcodeEncoder;
import oracle.xdo.common.log.Logger;
// This class name will be used in the register vendor
// field in the template.

public class BarcodeUtil  implements XDOBarcodeEncoder
// The class implements the XDOBarcodeEncoder interface
{
// This is the barcode vendor id that is used in the
// register vendor field and format-barcode fields
  public static final String BARCODE_VENDOR_ID = "XMLPBarVendor";
// The hashtable is used to store references to
// the encoding methods
  public static final Hashtable ENCODERS = new Hashtable(10);
// The BarcodeUtil class needs to be instantiated
  public static final BarcodeUtil mUtility = new BarcodeUtil();
// This is the main code that is executed in the class,
// it is loading the methods for the encoding into the hashtable.
// In this case we are loading the three code128 encoding
// methods we have created.
  static {
    try {
      Class[] clazz = new Class[] { "".getClass() };

      ENCODERS.put("code128a",mUtility.getClass().getMethod("code128a", clazz));
      ENCODERS.put("code128b",mUtility.getClass().getMethod("code128b", clazz));
      ENCODERS.put("code128c",mUtility.getClass().getMethod("code128c", clazz));
     } catch (Exception e) {
```

```
// This is using the BI Publisher logging class to push
// errors to the XMLP log file.
      Logger.log(e,5);
    }
  }

// The getVendorID method is called from the template layer
// at runtime to ensure the correct encoding method are used
    public final String getVendorID()
    {
        return BARCODE_VENDOR_ID;
    }
//The isSupported method is called to ensure that the
// encoding method called from the template is actually
//  present in this class.
// If not then XMLP will report this in the log.
    public final boolean isSupported(String s)
    {
        if(s != null)
            return ENCODERS.containsKey(s.trim().toLowerCase());
        else
            return false;
    }

// The encode method is called to then call the appropriate
// encoding method, in this example the code128a/b/c methods.

   public final String encode(String s, String s1)
    {
        if(s != null && s1 != null)
        {
            try
            {
                Method method = (Method)ENCODERS.get(s1.trim().toLowerCase());
                if(method != null)
                    return (String)method.invoke(this, new Object[] {
                        s
                    });
                else
                    return s;
            }
            catch(Exception exception)
            {
                  Logger.log(exception,5);
            }
            return s;
        } else
        {
            return s;
        }
    }

  /** This is the complete method for Code128a */

  public static final String code128a( String DataToEncode )
  {
    char C128_Start = (char)203;
    char C128_Stop = (char)206;
    String Printable_string = "";
    char CurrentChar;
    int CurrentValue=0;
    int weightedTotal=0;
    int CheckDigitValue=0;
    char C128_CheckDigit='w';

    DataToEncode = DataToEncode.trim();
    weightedTotal = ((int)C128_Start) - 100;
    for( int i = 1; i <= DataToEncode.length(); i++ )
      {
//get the value of each character
CurrentChar = DataToEncode.charAt(i-1);
if( ((int)CurrentChar) < 135 )
  CurrentValue = ((int)CurrentChar) - 32;
if( ((int)CurrentChar) > 134 )
  CurrentValue = ((int)CurrentChar) - 100;

CurrentValue = CurrentValue * i;
weightedTotal = weightedTotal + CurrentValue;
      }
    //divide the WeightedTotal by 103 and get the remainder,
    //this is the CheckDigitValue
    CheckDigitValue = weightedTotal % 103;
    if( (CheckDigitValue < 95) && (CheckDigitValue > 0) )
      C128_CheckDigit = (char)(CheckDigitValue + 32);
    if( CheckDigitValue > 94 )
      C128_CheckDigit = (char)(CheckDigitValue + 100);
    if( CheckDigitValue == 0 ){
      C128_CheckDigit = (char)194;
    }

    Printable_string = C128_Start + DataToEncode + C128_CheckDigit + C128_Stop + "
";
    return Printable_string;
  }

/** This is the complete method for Code128b ***/

  public static final String code128b( String DataToEncode )
  {
    char C128_Start = (char)204;
    char C128_Stop = (char)206;
    String Printable_string = "";
    char CurrentChar;
    int CurrentValue=0;
    int weightedTotal=0;
    int CheckDigitValue=0;
    char C128_CheckDigit='w';

    DataToEncode = DataToEncode.trim();
    weightedTotal = ((int)C128_Start) - 100;
    for( int i = 1; i <= DataToEncode.length(); i++ )
      {
//get the value of each character
CurrentChar = DataToEncode.charAt(i-1);
if( ((int)CurrentChar) < 135 )
  CurrentValue = ((int)CurrentChar) - 32;
if( ((int)CurrentChar) > 134 )
  CurrentValue = ((int)CurrentChar) - 100;

CurrentValue = CurrentValue * i;
```

```
weightedTotal = weightedTotal + CurrentValue;
      }
    //divide the WeightedTotal by 103 and get the remainder,
    //this is the CheckDigitValue
    CheckDigitValue = weightedTotal % 103;
    if( (CheckDigitValue < 95) && (CheckDigitValue > 0) )
      C128_CheckDigit = (char)(CheckDigitValue + 32);
    if( CheckDigitValue > 94 )
      C128_CheckDigit = (char)(CheckDigitValue + 100);
    if( CheckDigitValue == 0 ){
      C128_CheckDigit = (char)194;
    }

    Printable_string = C128_Start + DataToEncode + C128_CheckDigit + C128_Stop + "
";
    return Printable_string;
  }


  /** This is the complete method for Code128c **/

  public static final String code128c( String s )
  {
    char C128_Start = (char)205;
    char C128_Stop = (char)206;
    String Printable_string = "";
    String DataToPrint = "";
    String OnlyCorrectData = "";
    int i=1;
    int CurrentChar=0;
    int CurrentValue=0;
    int weightedTotal=0;
    int CheckDigitValue=0;
    char C128_CheckDigit='w';
    DataToPrint = "";
    s = s.trim();
    for(i = 1; i <= s.length(); i++ )
      {
//Add only numbers to OnlyCorrectData string
CurrentChar = (int)s.charAt(i-1);
if((CurrentChar < 58) && (CurrentChar > 47))
  {
    OnlyCorrectData = OnlyCorrectData + (char)s.charAt(i-1);
  }
      }
    s = OnlyCorrectData;
    //Check for an even number of digits, add 0 if not even
    if( (s.length() % 2) == 1 )
      {
s = "0" + s;
      }
    //<<<< Calculate Modulo 103 Check Digit and generate
    // DataToPrint >>>>
    //Set WeightedTotal to the Code 128 value of
// the start character
    weightedTotal = ((int)C128_Start) - 100;
    int WeightValue = 1;
    for( i = 1; i <= s.length(); i += 2 )
      {
//Get the value of each number pair (ex: 5 and 6 = 5*10+6 =56)
```

```
//And assign the ASCII values to DataToPrint
CurrentChar = ((((int)s.charAt(i-1))-48)*10) + (((int)s.charAt(i))-48);
if((CurrentChar < 95) && (CurrentChar  > 0))
  DataToPrint = DataToPrint + (char)(CurrentChar + 32);
if( CurrentChar > 94 )
  DataToPrint = DataToPrint + (char)(CurrentChar + 100);
if( CurrentChar == 0)
  DataToPrint = DataToPrint + (char)194;
//multiply by the weighting character
//add the values together to get the weighted total
weightedTotal = weightedTotal + (CurrentChar * WeightValue);
WeightValue = WeightValue + 1;
      }
    //divide the WeightedTotal by 103 and get the remainder,
    //this is the CheckDigitValue
    CheckDigitValue = weightedTotal % 103;
    if((CheckDigitValue < 95) && (CheckDigitValue > 0))
      C128_CheckDigit = (char)(CheckDigitValue + 32);
    if( CheckDigitValue > 94 )
      C128_CheckDigit = (char)(CheckDigitValue + 100);
    if( CheckDigitValue == 0 ){
      C128_CheckDigit = (char)194;
    }
    Printable_string = C128_Start + DataToPrint + C128_CheckDigit + C128_Stop + "
";
    Logger.log(Printable_string,5);
    return Printable_string;
  }
}
```

Once you create the class and place it in the correct classpath, your template creators can start using it to format the data for barcodes. You must give them the following information to include in the template commands:

- The class name and path.

  In this example:

  `oracle.xdo.template.rtf.util.barcoder.BarcodeUtil`

- The barcode vendor ID you created.

  In this example: `XMLPBarVendor`

- The available encoding methods.

  In this example, code128a, code128b and code128c They can then use this information to successfully encode their data for barcode output.

They can then use this information to successfully encode their data for barcode output.

# 8 Using the Delivery Manager Java APIs

This chapter describes the BI Publisher delivery manager APIs.

It includes the following sections:

## 8.1 Using the Delivery Manager

The Delivery Manager is a set of Java APIs that enables you to control the delivery of your BI Publisher documents. Use the Delivery Manager to:

- Deliver documents through established or custom delivery channels

- Redeliver documents

To use the Delivery Manager follow these steps:

1. Create a DeliveryManager instance.
2. Create a DeliveryRequest instance using the `createRequest()` method.
3. Add the request properties (such as DeliveryRequest destination). Most properties require a String value. For more information, see the supported properties for each delivery channel.
4. Set your document to the DeliveryRequest.
5. Call `submit()` to submit the delivery request.

One delivery request can handle one document and one destination. This facilitates monitoring and resubmitting, if necessary.

DeliveryRequest enables you to set documents in the following two ways:

- Set InputStream of the document to DeliveryRequest. The DeliveryRequest will read the InputStream when you call `submit()` for the first time. The DeliveryRequest does not close the InputStream so you must ensure to close it.
- Set the file name of the document to DeliveryRequest.

The Delivery Manager supports streamlined delivery when you set the direct mode. See Section 8.14, "Direct and Buffering Modes."

The follow delivery channels are described in this document:

- E-mail
- Printer
- Local Printer
- Fax
- RightFax
- WebDAV
- FTP
- Secure FTP
- HTTP
- AS2

## 8.2 Delivering Documents by E-Mail

The following sample demonstrates delivery through e-mail:

***Example 8–1 Sample Code for Delivering Documents through E-Mail***

```
// create delivery manager instance
     DeliveryManager dm = new DeliveryManager();
     // create a delivery request
     DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_SMTP_EMAIL);

     // set email subject
     req.addProperty(DeliveryPropertyDefinitions.SMTP_SUBJECT, "test mail");
     // set SMTP server host
     req.addProperty(
       DeliveryPropertyDefinitions.SMTP_HOST, "mysmtphost");
     // set the sender email address
     req.addProperty(DeliveryPropertyDefinitions.SMTP_FROM,
"myname@foobar.com.example");
     // set the destination email address
     req.addProperty(
       DeliveryPropertyDefinitions.SMTP_TO_RECIPIENTS, "user1@foobar.com.example,
user2@foobar.com.example" );
     // set the content type of the email body
     req.addProperty(DeliveryPropertyDefinitions.SMTP_CONTENT_TYPE,
"application/pdf");
     // set the document file name appeared in the email
     req.addProperty(DeliveryPropertyDefinitions.SMTP_CONTENT_FILENAME,
"test.pdf");
     // set the document to deliver
     req.setDocument("/document/test.pdf");

     // submit the request
     req.submit();
     // close the request
     req.close();
```

The following table lists the supported properties:

***Table 8–1 Properties for E-Mail Delivery***

| Property | Description |
|---|---|
| SMTP_TO_RECIPIENTS | Required<br>Enter multiple recipients separated by a comma (example: "user1@foobar.com.example, user2@foobar.com.example") |
| SMTP_CC_RECIPIENTS | Optional<br>Enter multiple recipients separated by a comma. |
| SMTP_BCC_RECIPIENTS | Optional<br>Enter multiple recipients separated by a comma. |
| SMTP_FROM | Required<br>Enter the e-mail address of the sending party. |
| SMTP_REPLY_TO | Optional<br>Enter the reply-to e-mail address. |
| SMTP_SUBJECT | Required<br>Enter the subject of the e-mail. |
| SMTP_CHARACTER_ ENCODING | Optional<br>Default is "UTF-8". |
| SMTP_ATTACHMENT | Optional<br>If you are including an attachment, enter the attachment object name. |
| SMTP_CONTENT_ FILENAME | Optional<br>Enter the file name of the attachment (example: invoice.pdf) |
| SMTP_CONTENT_ DISPOSITION | Content disposition of the attachment. Value should be either "inline" or "attachment". Default is "attachment". |
| SMTP_CONTENT_TYPE | Required<br>Enter the MIME type. |

***Table 8–1 (Cont.) Properties for E-Mail Delivery***

| Property | Description |
|---|---|
| SMTP_SMTP_HOST | Required<br>Enter the SMTP host name. |
| SMTP_SMTP_PORT | Optional<br>Enter the SMTP port. Default is 25. |
| SMTP_SECURE_ CONNECTION | This property controls secure connection method to use.<br>Valid values are:<br>■ "none" - default<br>■ "tls" - use STARTTLS when server supports the command.<br>■ "tls_required" - use STARTTLS and abort if server does not support the command.<br>■ "ssl" - for Secure Sockets Layer |
| SMTP_SMTP_ USERNAME | Optional<br>If the SMTP server requires authentication, enter your username for the server. |
| SMTP_SMTP_ PASSWORD | Optional<br>If the SMTP server requires authentication, enter the password for the username you entered. |
| SMTP_ATTACHMENT_ FIRST | Optional<br>If your e-mail contains an attachment and you want the attachment to appear first, enter "true". If you do not want the attachment to appear first, enter "false". |

## 8.2.1 Defining Multiple Recipients

The e-mail delivery server channel supports multiple documents and multiple destinations per request. The following example demonstrates multiple TO and CC addresses:

***Example 8–2 Sample Code for Defining Multiple Recipients***

```
    // set the TO email addresses
     req.addProperty(
       DeliveryPropertyDefinitions.SMTP_TO_RECIPIENTS,
        "user1@foobar.com.example, user2@foobar.com.example,
user3@foobar.com.example");

     // set the CC email addresses
     req.addProperty(
       DeliveryPropertyDefinitions.SMTP_CC_RECIPIENTS,
        "user4@foobar.com.example, user5@foobar.com.example,
user6@foobar.com.example");
```

## 8.2.2 Attaching Multiple Documents to One Request

Use the Attachment utility class (`oracle.apps.xdo.delivery.smtp.Attachment`) to attach multiple documents into one request. Sample usage is as follows:

***Example 8–3 Sample Code for Attaching Multiple Documents to One Request***

```
  :
  :
// create Attachment instance
Attachment m = new Attachment();

// add PDF attachment
m.addAttachment(
    "/pdf_doc/invoice.pdf",      // file to deliver
    "invoice.pdf",               // file name as appears in email
    "application/pdf");          // content type

// add RTF attachment
m.addAttachment(
    "/rtf_doc/product.rtf",      // file to deliver
    "product.rtf",               // file name appears in the email
    "application/rtf");          // content type

// add XML attachment
m.addAttachment(
    "/xml_doc/data.xml",          // file to deliver
    "data.xml",                   // file name appears in the email
    "text/xml");                  // content type

// If you want to attach HTML doucments, use addHtmlAttachment().
// This method automatically resolves the image references
// in your HTML document and attaches those images.
m.addHtmlAttachment("/html_doc/invoice.html");

// add the attachment to the request
req.addProperty(DeliveryPropertyDefinitions.SMTP_ATTACHMENT, m);

  :
  :
```

## 8.2.3 Attaching HTML Documents

You can attach HTML documents into one request. If you have references to image files located in the local file system in your HTML document, the Attachment utility automatically attaches those image files also. The sample usage is as follows:

***Example 8–4 Sample Code for Attaching HTML Documents***

```
Attachment m = new Attachment();
m.addHtmlAttachment("/path/to/my.html");
  :
  :

req.addProperty(DeliveryPropertyDefinitions.SMTP_ATTACHMENT, m);
```

## 8.2.4 Displaying Attachments at the Top of E-Mail

If you want to show your attachment at the top of an e-mail, set the property SMTP_ ATTACHMENT_FIRST to "true". Sample usage is as follows.

***Example 8–5 Sample Code for Displaying Attachments at the Top of E-Mail***

```
Attachment m = new Attachment();
m.addHtmlAttachment("/path/to/my.html");
  :
  :
req.addProperty(DeliveryPropertyDefinitions.SMTP_ATTACHMENT_FIRST, "true");
  :
```

### 8.2.5 Using a String Object as the E-Mail Body

You can use a String object for the e-mail body. This may be useful if you want to include a message with your attached files. The following sample code will deliver the message "Please find the attached invoice." in the e-mail body and one PDF document "invoice.pdf" as an attachment.

***Example 8–6 Sample Code for Using a String Object as the E-Mail Body***

```
   // create delivery manager instance
   DeliveryManager dm = new DeliveryManager();
   // create a delivery request
   DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_SMTP_EMAIL);

   // set email subject
   req.addProperty(DeliveryPropertyDefinitions.SMTP_SUBJECT, "Invoice");
   // set SMTP server host
   req.addProperty(
     DeliveryPropertyDefinitions.SMTP_HOST, "mysmtphost");
   // set the sender email address
   req.addProperty(DeliveryPropertyDefinitions.SMTP_FROM,
"myname@foobar.com.example");
   // set the destination email address
   req.addProperty(
     DeliveryPropertyDefinitions.SMTP_TO_RECIPIENTS, "user1@foobar.com.example,
user2@foobar.com.example" );
   // set the document to deliver
   req.setDocument("Please find the attached invoice. ", "UTF-8");

   // create Attachment
   Attachment m = new Attachment();
   // add attachments
   m.addAttachment(
     "/pdf_doc/invoice.pdf",            // file to deliver
     "invoice.pdf",                     // file name appears in the email
     "application/pdf");                // content type
   // add the attachment to the request
   req.addProperty(DeliveryPropertyDefinitions.SMTP_ATTACHMENT, m);

   // submit the request
   req.submit();
   // close the request
   req.close();

     :
     :
```

### 8.2.6 Using an HTML Document as the E-Mail Body

You can also use an HTML document for the e-mail body. The utility automatically resolves the local image references in your HTML document and attaches those images.

Sample usage is as follows:

***Example 8–7 Sample Code for Using an HTML Document as the E-Mail Body***

```
   // create delivery manager instance
   DeliveryManager dm = new DeliveryManager();
   // create a delivery request
   DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_SMTP_EMAIL);

   // set email subject
   req.addProperty(DeliveryPropertyDefinitions.SMTP_SUBJECT, "Invoice");
   // set SMTP server host
   req.addProperty(
     DeliveryPropertyDefinitions.SMTP_HOST, "mysmtphost");
   // set the sender email address
   req.addProperty(DeliveryPropertyDefinitions.SMTP_FROM,
"myname@foobar.com.example");
   // set the destination email address
   req.addProperty(
     DeliveryPropertyDefinitions.SMTP_TO_RECIPIENTS, "user1@foobar.com.example,
user2@foobar.com.example" );

   // set the content type of the email body
   req.addProperty(DeliveryPropertyDefinitions.SMTP_CONTENT_TYPE, "text/html");
   // set the document file name appeared in the email
   req.addProperty(DeliveryPropertyDefinitions.SMTP_CONTENT_FILENAME,
"body.html");
   // set the document to deliver
   req.setDocument("/document/invoice.html");

   // submit the request
   req.submit();
   // close the request
   req.close();

     :
     :
```

### 8.2.7 Providing User Name and Password for Authentication

If the SMTP server requires authentication, you can specify the username and password to the delivery request.

***Example 8–8 Sample Code for Providing User Name and Password for Authentication***

```
 :
   req.addProperty(DeliveryPropertyDefinitions.SMTP_USERNAME, "scott");
   req.addProperty(DeliveryPropertyDefinitions.SMTP_PASSWORD, "tiger");
     :
```

## 8.3 Delivering Your Document to a Printer

The Delivery Manager supports Internet Printing Protocol (IPP) as defined in RFC 2910 and 2911 for the delivery of documents to IPP-supported printers or servers, such as CUPS.

Common Unix Printing System (CUPS) is a free, server-style, IPP-based software that can accept IPP requests and dispatch those requests to both IPP and non-IPP based devices, such as printers and fax machines. See `http://www.cups.org/` for more information about CUPS.

To print out your document with the IPP, you need to transform your document into the format that the target IPP printers or servers can understand before the delivery. For example, if the target printer is a Postscript printer, you must transform your document to Postscript format. Usually, printers do not natively understand PDF, RTF, Excel or Word document formats. The Delivery API itself does not provide the document format transformation functionality, but it does offer document filter support for this purpose. See Section 8.16, "Document Filter Support."

Following is a code sample for delivery to a printer:

***Example 8–9   Sample Code for Delivering Documents to a Printer***

```
// create delivery manager instance
     DeliveryManager dm = new DeliveryManager();
     // create a delivery request
     DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_IPP_PRINTER);

     // set IPP printer host
     req.addProperty(DeliveryPropertyDefinitions.IPP_HOST, "myhost");
     // set IPP printer port
     req.addProperty(DeliveryPropertyDefinitions.IPP_PORT, "631");
     // set IPP printer name
     req.addProperty(DeliveryPropertyDefinitions.IPP_PRINTER_NAME,
"/printers/myprinter");
     // set the document format
     req.addProperty(DeliveryPropertyDefinitions.IPP_DOCUMENT_FORMAT,
       DeliveryPropertyDefinitions.IPP_DOCUMENT_FORMAT_POSTSCRIPT);
     // set the document
     req.setDocument("/document/invoice.ps");

     // submit the request
     req.submit();
     // close the request
     req.close();
```

The following properties are supported. A string value is required for each property, unless otherwise noted. Note that printer-specific properties such as IPP_SIDES, IPP_COPIES and IPP_ORIENTATION depend on the printer capabilities. For example, if the target printer does not support duplex printing, the IPP_SIDES setting will have no effect.

***Table 8–2   Properties for Delivering Documents to Printers***

| Property | Description |
|---|---|
| IPP_HOST | Required<br>Enter the host name. |

*Table 8–2 (Cont.) Properties for Delivering Documents to Printers*

| Property | Description |
|---|---|
| IPP_PORT | Optional<br>Default is 631. |
| IPP_PRINTER_NAME | Required<br>Enter the name of the printer that is to receive the output.<br>■ If you use CUPS with the default setup, enter the printer name as `/printers/<printer-name>`<br>■ If you use the Microsoft Internet Information Service (IIS) with the default setup, enter the printer name as `/printers/<printer-name>/.printer` |
| IPP_AUTHTYPE | Optional<br>Valid values for authentication type are:<br>IPP_AUTHTYPE_NONE - no authentication (default)<br>IPP_AUTHTYPE_BASIC - use HTTP basic authentication<br>IPP_AUTHTYPE_DIGEST - use HTTP digest authentication |
| IPP_USERNAME | Optional<br>Enter the username for HTTP authentication. |
| IPP_PASSWORD | Optional<br>Enter the password for HTTP authentication. |
| IPP_ENCTYPE | Optional<br>The encryption type can be set to either of the following:<br>IPP_ENCTYPE_NONE - no encryption (default)<br>IPP_ENCTYPE_SSL - use Secure Socket Layer |
| IPP_USE_FULL_URL | Optional<br>Set to "true" to send the full URL for the HTTP request header. Valid values are "true" or "false" (default). |
| IPP_USE_CHUNKED_ BODY | Optional<br>Valid values are "true" (default) to use HTTP chunked transfer coding for the message body, or "false". |
| IPP_ATTRIBUTE_ CHARSET | Optional<br>Attribute character set of the IPP request. Default is "UTF-8". |
| IPP_NATURAL_ LANGUAGE | Optional<br>The natural language of the IPP request. Default is "en". |
| IPP_JOB_NAME | Optional<br>Job name of the IPP request. |
| IPP_COPIES | Optional<br>Define the number of copies to print (example: "1", "5", "10"). Default is 1. |

*Table 8–2 (Cont.) Properties for Delivering Documents to Printers*

| Property | Description |
|---|---|
| IPP_SIDES | Optional<br><br>Enable two-sided printing. This setting will be ignored if the target printer does not support two-sided printing. Valid values are:<br>■ IPP_SIDES_ONE_SIDED - default<br>■ IPP_SIDES_TWO_SIDED_LONG_EDGE - prints both sides of paper for binding long edge.<br>■ IPP_SIDES_TWO_SIDED_SHORT_EDGE - prints both sides of paper for binding short edge.<br>■ IPP_SIDES_DUPLEX: Same as IPP_SIDES_TWO_SIDED_ LONG_EDGE.<br>■ IPP_SIDES_TUMBLE: Same as IPP_SIDES_TWO_SIDED_ SHORT_EDGE. |
| IPP_ORIENTATIONS | Optional<br><br>Sets the paper orientation. This setting will be ignored if the target printer does not support orientation settings. Valid values are:<br><br>IPP_ORIENTATIONS_PORTRAIT (default)<br><br>IPP_ORIENTATIONS_LANDSCAPE |
| IPP_DOCUMENT_ FORMAT | Optional<br><br>The target printer must support the specified format. Valid values are:<br><br>IPP_DOCUMENT_FORMAT_POSTSCRIPT<br><br>IPP_DOCUMENT_FORMAT_PLAINTEXT<br><br>IPP_DOCUMENT_FORMAT_PDF<br><br>IPP_DOCUMENT_FORMAT_OCTETSTREAM (default) |
| IPP_MEDIA | You can choose either the paper size or the tray number. If you do not specify this option, the default media of the target printer will be used. It will be ignored if the target printer doesn't support the media option. Valid values are:<br>■ IPP_MEDIA_TRAY1: Media on tray 1<br>■ IPP_MEDIA_TRAY2: Media on tray 2<br>■ IPP_MEDIA_TRAY3: Media on tray 3<br>■ IPP_MEDIA_A3: A3 Media<br>■ IPP_MEDIA_A4: A4 Media<br>■ IPP_MEDIA_A5: A5 Media<br>■ IPP_MEDIA_B4: B4 Media<br>■ IPP_MEDIA_B5: B5 Media |
| IPP_PAGE_RANGES | Specify page ranges to print. By default, all pages are printed. Example valid values are:<br>■ "3": prints only page 3.<br>■ "2-5" : prints pages 2-5.<br>■ "1,3-5": print page 1 and 3-5. |

### 8.3.1 Printing over an HTTP Proxy Server

To deliver documents to IPP printers or fax machines over an HTTP proxy server, you may encounter delivery problems due to differences in the HTTP implementations between CUPS and the proxy servers. Setting the following two properties can resolve most of these problems:

- DeliveryPropertyDefinitions.IPP_USE_FULL_URL - set to "true"
- DeliveryPropertyDefinitions.IPP_USE_CHUNKED_BODY - set to "false"

If you use CUPS with the default setup, the typical property settings are as follows:

- `IPP_HOST : <host-name>`
- `IPP_PORT : 631`
- `IPP_PRINTER_NAME : /printers/<printer-name>`

If you use the Microsoft Internet Information Service (IIS) with the default setup, the typical property settings are as follows:

- `IPP_HOST : <host-name>`
- `IPP_PORT : 80`
- `IPP_PRINTER_NAME : /printers/<printer-name>/.printer`

## 8.4 Delivering Your Document to a Local Printer

The Delivery Manager supports delivery of documents to "local" printers attached to the system where the Delivery Manager runs.

Following is a code sample for delivery to a local printer.

***Example 8–10 Sample Code for Delivering Documents to a Local Printer***

```
      // create delivery manager instance
      DeliveryManager dm = new DeliveryManager();
      // create a delivery request
      DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_PRINTER);
      // set target printer name as HOST - if no HOST is set default printer is
used
      //req.addProperty(PRINTER_HOST, "PrinterName");
      // set cotnent type - the content type must be supported by the printer
      req.addProperty(CONTENT_TYPE, CONTENT_TYPE_POSTSCRIPT);
      // set the document
      req.setDocument("/document/invoice.ps");
      // submit the request
      req.submit();
      // close the request
      req.close();
```

The following table lists the supported properties. Note that support of printer-specific properties such as PRINTER_SIDES, PRINTER_COPIES, PRINTER_MEDIA, PRINTER_ORIENTATION, PRINTER_PAGE_RANGES and PRINTER_SIDES depends on the printer and local printing system's capabilities. For example, on Windows, these properties are ignored unless a you also use a filter that supports adding these properties to your document.

***Table 8–3 Properties for Delivering Documents to Local Printers***

| Property | Description |
|---|---|
| PRINTER_CONTENT_ TYPE | Optional<br><br>The document content type (example: "application/pdf"). |
| PRINTER_COPIES | Optional<br><br>Specify the number of copies to print (example: "1", "5", "10"). Default is 1. |
| PRINTER_HOST | Optional<br><br>Printer name (name of the printer on the operating system or local printing system) to send the documents to. If HOST is not specified, the default local printer is used. |
| PRINTER_MEDIA | Optional<br><br>You can choose either the paper size or the tray number. If you do not specify this option, the default media of the target printer will be used. It will be ignored if the target printer doesn't support the media option.<br><br>Valid values are:<br>■ PRINTER_MEDIA_TRAY1: Media on tray 1<br>■ PRINTER_MEDIA_TRAY2: Media on tray 2<br>■ PRINTER_MEDIA_TRAY3: Media on tray 3<br>■ PRINTER_MEDIA_A3: A3 Media<br>■ PRINTER_MEDIA_A4: A4 Media<br>■ PRINTER_MEDIA_A5: A5 Media<br>■ PRINTER_MEDIA_B4: B4 Media<br>■ PRINTER_MEDIA_B5: B5 Media |
| PRINTER_ORIENTATIONS | Optional<br><br>Sets the paper orientation. This setting will be ignored if the target printer does not support orientation settings. Valid values are: PRINTER_ORIENTATIONS_PORTRAIT (default) PRINTER_ORIENTATIONS_LANDSCAPE |
| PRINTER_PAGE_RANGES | Specify page ranges to print. By default, all pages are printed. Example valid values are:<br>■ "3": prints only page 3.<br>■ "2-5": prints pages 2-5.<br>■ "1,3-5": print page 1 and 3-5. |
| PRINTER_SIDES | Optional<br><br>Enable two-sided printing. This setting will be ignored if the target printer does not support two-sided printing. Valid values are:<br>■ PRINTER_SIDES_ONE_SIDED - default<br>■ PRINTER_SIDES_TWO_SIDED_LONG_EDGE - prints both sides of paper for binding long edge.<br>■ PRINTER_SIDES_TWO_SIDED_SHORT_EDGE - prints both sides of paper for binding short edge.<br>■ PRINTER_SIDES_DUPLEX: Same as PRINTER_SIDES_ TWO_SIDED_LONG_EDGE.<br>■ PRINTER_SIDES_TUMBLE: Same as PRINTER_SIDES_ TWO_SIDED_SHORT_EDGE. |

## 8.5 Delivering Your Documents to a Fax Server

The delivery manager supports the delivery of documents to fax modems configured on CUPS. You can configure fax modems on CUPS with efax. For information about efax, see the website: `http://www.cce.com/efax/`

FAX4CUPS is freely available from various websites. Search for FAX4CUPS in your internet search engine to find a site that provides this software.

Sample code for fax delivery is as follows:

***Example 8–11 Sample Code for Delivering Documents to a Fax Server***

```
// create delivery manager instance
     DeliveryManager dm = new DeliveryManager();
     // create a delivery request
     DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_IPP_FAX);

     // set IPP fax host
     req.addProperty(DeliveryPropertyDefinitions.IPP_HOST, "myhost");
     // set IPP fax port
     req.addProperty(DeliveryPropertyDefinitions.IPP_PORT, "631");
     // set IPP fax name
     req.addProperty(DeliveryPropertyDefinitions.IPP_PRINTER_NAME,
"/printers/myfax");
     // set the document format
     req.addProperty(DeliveryPropertyDefinitions.IPP_DOCUMENT_FORMAT,
"application/postscript");
     // set the phone number to send
     req.addProperty(DeliveryPropertyDefinitions.IPP_PHONE_NUMBER, "9999999");
     // set the document
     req.setDocument("/document/invoice.pdf");

     // submit the request
     req.submit();
     // close the request
     req.close();
```

The supported properties are the same as those supported for printer documents, plus the following:

***Table 8–4 Properties for Delivering Documents to Fax Servers***

| Property | Description |
|---|---|
| IPP_PHONE_NUMBER | Required<br>Enter the fax number. |

## 8.6 Delivering Your Documents to a RightFax Server

The Delivery Manager supports the delivery of documents to OpenText Fax Server, RightFax Edition (formerly Captaris RightFax) 9.3 or above. The XML interface on HTTP port must be enabled on RightFax server to enable this integration.

Following is a code sample for delivery to RightFax server:

***Example 8–12 Sample Code for Delivering Documents to a RightFax Server***

```
// create delivery manager instance
DeliveryManager dm = new DeliveryManager();
// create a delivery request
```

```
DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_RIGHTFAX);
// set RightFax host
req.addProperty(DeliveryPropertyDefinitions.RIGHTFAX_HTTP_HOST, "myhost");
// set RightFax server port
req.addProperty(DeliveryPropertyDefinitions.RIGHTFAX_HTTP_PORT, "80");
// set the target remote directory
req.addProperty(DeliveryPropertyDefinitions.RIGHTFAX_HTTP_REMOTE_DIRECTORY,
"/RFWebCon.dll");
// sender information
req.addProperty(DeliveryPropertyDefinitions.RIGHTFAX_SENDER_NAME, "Lex De Hann");
req.addProperty(DeliveryPropertyDefinitions.RIGHTFAX_SENDER_COMPANY, "Company,
Ltd.");
req.addproperty(DeliveryPropertyDefinitions.RIGHTFAX_SENDER_PHONE", "555-9976");
// destionation
req.addProperty(DeliveryPropertyDefinitions.RIGHTFAX_TO_FAXNUM, "555-1111");
req.addProperty(DeliveryPropertyDefinitions.RIGHTFAX_TO_NAME, "Jane Bennett");
req.addProperty(DeliveryPropertyDefinitions.RIGHTFAX_TO_COMPANY, "Acme, Inc.");
// set the document
req.setDocument("/document/invoice.pdf");
// submit the request
req.submit();
// close the request
req.close();
```

The following table lists the supported properties:

***Table 8–5 Properties for Delivering Documents to RightFax Servers***

| Property | Description |
|---|---|
| RIGHTFAX_HTTP_HOST | Required<br>HTTP host of the RightFax server |
| RIGHTFAX_HTTP_PORT | Optional<br>HTTP port of the RightFax server. Default=80. |
| RIGHTFAX_HTTP_REMOTE_DIRECTORY | Optional<br>Enter the remote directory name (example: /RFWebCon.dll) of the RightFax XML interface. |
| RIGHTFAX_HTTP_AUTHTYPE | Optional<br>HTTP authentication type of the RightFax server URL. Valid values are RIGHTFAX_HTTP_AUTHTYPE_NONE, RIGHTFAX_HTTP_AUTHTYPE_BASIC, RIGHTFAX_HTTP_AUTHTYPE_DIGEST. Default value is RIGHTFAX_AUTHTYPE_NONE. |
| RIGHTFAX_HTTP_USERNAME | Optional<br>HTTP username for the RightFax server url. Required when RIGHTFAX_HTTP_AUTH_TYPE is set to values other than RIGHTFAX_HTTP_AUTHTYPE_NONE. |
| RIGHTFAX_HTTP_PASSWORD | Optional<br>HTTP password for the RightFax server url. Required when RIGHTFAX_HTTP_AUTH_TYPE is set to values other than RIGHTFAX_HTTP_AUTHTYPE_NONE. |
| RIGHTFAX_HTTP_ENCTYPE | Optional<br>The encryption type can be set to either of the following:<br>RIGHTFAX_HTTP_ENCTYPE_NONE – no encryption (default)<br>RIGHTFAX_HTTP_ENCTYPE_SSL – use Secure Socket Layer |

*Table 8–5 (Cont.) Properties for Delivering Documents to RightFax Servers*

| Property | Description |
|---|---|
| RIGHTFAX_HTTP_USE_FULL_URL | Optional<br>Set to "true" to send the full URL for the HTTP request header. Valid values are "true" or "false" (default). |
| RIGHTFAX_HTTP_USE_CHUNKED_BODY | Optional<br>Valid values are "true" (default) to use HTTP chunked transfer coding for the message body, or "false". |
| RIGHTFAX_HTTP_TIMEOUT | Optional<br>Enter a length of time in milliseconds after which to terminate the request if a connection is not made to the HTTP server. The default is 60000 (1 minute). |
| RIGHTFAX_HTTP_PROXY_HOST | Optional<br>Enter the proxy server host name. |
| RIGHTFAX_HTTP_PROXY_PORT | Optional<br>Enter the proxy server port number. Default=80. |
| RIGHTFAX_HTTP_PROXY_AUTHTYPE | Optional<br>Valid value is either of the following. RIGHTFAX_HTTP_PROXY_AUTHTYPE_NONE – no authentication RIGHTFAX_HTTP_PROXY_AUTHTYPE_BASIC – Use HTTP basic authentication RIGHTFAX_HTTP_PROXY_AUTHTYPE_DIGEST – Use HTTP digest authentication. |
| RIGHTFAX_HTTP_PROXY_USERNAME | Optional<br>Enter the username for proxy authentication. |
| RIGHTFAX_HTTP_PROXY_PASSWORD | Optional<br>Enter the password for HTTP proxy authentication. |
| RIGHTFAX_SENDER_FROM_NAME | Optional<br>Enter the name of the sender. |
| RIGHTFAX_SENDER_EMP_ID | Optional<br>Enter the employee id of the sender. |
| RIGHTFAX_SENDER_FROM_COMPANY | Optional<br>Enter the name of the sender's company. |
| RIGHTFAX_SENDER_FROM_DEPARTMENT | Optional<br>Enter the name of the sender's department. |
| RIGHTFAX_SENDER_FROMO_PHONE | Optional<br>Enter sender's phone number. |
| RIGHTFAX_SENDER_RETURN_EMAIL | Optional<br>Enter sender's return email address. |
| RIGHTFAX_SENDER_BILLINFO1 | Optional<br>Enter the billing code of the fax owner. |
| RIGHTFAX_SENDER_BILLINFO2 | Optional<br>Enter the secondary billing code of the fax owner. |
| RIGHTFAX_SENDER_RF_USER | Required<br>Enter the name of the sender's RightFax user name. |

***Table 8–5 (Cont.) Properties for Delivering Documents to RightFax Servers***

| **Property** | **Description** |
|---|---|
| RIGHTFAX_FAX_TO_NUMBER | Required<br>Enter the fax number where the document will be sent. |
| RIGHTFAX_FAX_TO_NAME | Optional<br>Enter the recipient's name. |
| RIGHTFAX_FAX_TO_COMPANY | Optional<br>Enter the recipient's company name. |
| RIGHTFAX_FAX_ALT_FAX_NUM | Optional<br>Enter the alternative fax number. |
| RIGHTFAX_FAX_TO_CONTACTNUM | Optional<br>Enter the contact phone number of the recipient. |
| RIGHTFAX_FAX_COVERSHEET | Optional<br>Enter the cover sheet template for the current document. The file name can be either a full path on the RightFax server computer or a path relative to RightFax\Production\Covers. |
| RIGHTFAX_COVERTEXT | Optional<br>Enter the text that should appear on the cover sheet. |
| RIGHTFAX_COVERTEXT_TYPE | Optional<br>Enter the type of the cover sheet text. Valid values are:<br>■ TXT (default)<br>■ RTF |
| RIGHTFAX_COVERTEXT_ENCODING | Optional<br>Enter the encoding of the cover sheet text. Valid values are:<br>■ NONE (default)<br>■ BASE64<br>■ QUOTEDPRINTABLE |
| RIGHTFAX_DOCUMENT_FORMAT | Optional<br>Valid values are:<br>■ PDF (default)<br>■ PS<br>■ TEXT |

## 8.7 Delivering Your Documents to a WebDAV Server

The following is sample code for delivery to a Web-based Distributed Authoring and Versioning (WebDAV) server:

***Example 8–13 Sample Code for Delivering Documents to a WebDAV Server***

```
// create delivery manager instance
     DeliveryManager dm = new DeliveryManager();
     // create a delivery request
     DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_WEBDAV);

     // set document content type
     req.addProperty(DeliveryPropertyDefinitions.WEBDAV_CONTENT_TYPE,
"application/pdf");
     // set the WebDAV server hostname
     req.addProperty(DeliveryPropertyDefinitions.WEBDAV_HOST, "mywebdavhost");
     // set the WebDAV server port number
     req.addProperty(DeliveryPropertyDefinitions.WEBDAV_PORT, "80");
     // set the target remote directory
     req.addProperty(DeliveryPropertyDefinitions.WEBDAV_REMOTE_DIRECTORY,
"/content/");
     // set the remote filename
     req.addProperty(DeliveryPropertyDefinitions.WEBDAV_REMOTE_FILENAME,
"xdotest.pdf");

     // set username and password to access WebDAV server
     req.addProperty(DeliveryPropertyDefinitions.WEBDAV_USERNAME, "xdo");
     req.addProperty(DeliveryPropertyDefinitions.WEBDAV_PASSWORD, "xdo");
     // set the document
     req.setDocument("/document/test.pdf");

     // submit the request
     req.submit();
     // close the request
     req.close();
```

The following properties are supported. A String value is required for each, unless otherwise noted.

***Table 8–6 Properties for Delivering Documents to WebDAV Servers***

| Property | Description |
|---|---|
| WEBDAV_CONTENT_TYPE | Required<br>Enter the document content type (example: "application/pdf"). |
| WEBDAV_HOST | Required<br>Enter the server host name. |
| WEBDAV_PORT | Optional<br>Enter the server port number.<br>Default is 80. |
| WEBDAV_REMOTE_DIRECTORY | Required.<br>Enter the remote directory name (example: "/myreports/"). |
| WEBDAV_REMOTE_FILENAME | Required.<br>Enter the remote file name. |
| WEBDAV_AUTHTYPE | Optional<br>Valid values for authentication type are:<br>WEBDAV_AUTHTYPE_NONE - no authentication (default)<br>WEBDAV_AUTHTYPE_BASIC - use HTTP basic authentication<br>WEBDAV_AUTHTYPE_DIGEST - use HTTP digest authentication |
| WEBDAV_USERNAME | Optional<br>Enter the username for HTTP authentication. |
| WEBDAV_PASSWORD | Optional<br>Enter the password for HTTP authentication. |

*Table 8–6 (Cont.) Properties for Delivering Documents to WebDAV Servers*

| Property | Description |
|---|---|
| WEBDAV_ENCTYPE | Optional<br>Valid values for encryption type are:<br>WEBDAV_ENCTYPE_NONE - no encryption (default)<br>WEBDAV_ENCTYPE_SSL - use Secure Socket Layer |
| WEBDAV_USE_FULL_URL | Optional<br>Set to "true" to send the full URL for the HTTP request header. Valid values are "true" or "false" (default). |
| WEBDAV_USE_CHUNKED_BODY | Optional<br>Valid values are "true" (default) to use HTTP chunked transfer coding for the message body, or "false". |
| WEBDAV_URL_CHARACTER_ENCODING | Encoding of the URL. It will be used if you use non-ASCII characters in the URL. Set the Java-supported encoding string for the value. |

## 8.8 Delivering Your Document over the File Transfer Protocol (FTP)

The following is sample code for delivery to an FTP server:

*Example 8–14 Sample Code for Delivering Documents over FTP*

```
// create delivery manager instance
     DeliveryManager dm = new DeliveryManager();
     // create a delivery request
     DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_FTP);

     // set hostname of the FTP server
     req.addProperty(DeliveryPropertyDefinitions.FTP_HOST, "myftphost");
     // set port# of the FTP server
     req.addProperty(DeliveryPropertyDefinitions.FTP_PORT, "21");
     // set username and password to access WebDAV server
     req.addProperty(DeliveryPropertyDefinitions.FTP_USERNAME, "xdo");
     req.addProperty(DeliveryPropertyDefinitions.FTP_PASSWORD, "xdo");
     // set the remote directory that you want to send your document to
     req.addProperty(DeliveryPropertyDefinitions.FTP_REMOTE_DIRECTORY, "pub");
     // set the remote file name
     req.addProperty(DeliveryPropertyDefinitions.FTP_REMOTE_FILENAME, "test.pdf");
     // set the document
     req.setDocument("/document/test.pdf");

     // submit the request
     req.submit();
     // close the request
     req.close();
```

The following properties are supported. A String value is required unless otherwise noted.

*Table 8–7 Properties for Delivering Documents over FTP*

| Property | Description |
|---|---|
| FTP_HOST | Required<br>Enter the server host name. |

*Table 8–7 (Cont.) Properties for Delivering Documents over FTP*

| Property | Description |
|---|---|
| FTP_PORT | Optional<br>Enter the server port number. Default is 21. |
| FTP_USERNAME | Required<br>Enter the login user name to the FTP server. |
| FTP_PASSWORD | Required<br>Enter the login password to the FTP server. |
| FTP_REMOTE_DIRECTORY | Required<br>Enter the directory to which to deliver the document (example: /pub/). To deliver the document to the user's home directory, enter '.' |
| FTP_REMOTE_FILENAME | Required<br>Enter the document file name for the remote server. |
| FTP_BINARY_MODE | Optional<br>Valid values are "true" (default) or "false". |
| FTP_PASSIVE_MODE | Optional<br>Valid values are "true" or "false" (default). |

## 8.9 Delivering Your Documents over Secure FTP

Secure FTP is the protocol based on the Secure Shell technology (ssh) and it is widely used to transfer files in a secure manner. Both Secure Shell and Secure FTP are defined by the Internet Engineering Task Force (IETF) and the specifications are available on their Web site: `http://www.ietf.org`. The delivery system supports the delivery of documents to secure FTP servers.

The following tables lists the supported properties. A string value is required for each property unless otherwise noted.

*Example 8–15 Sample Code for Delivering Documents over SFTP*

```
// create delivery manager instance
     DeliveryManager dm = new DeliveryManager();
     // create a delivery request
     DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_SFTP);
     // set hostname of the SFTP server
     req.addProperty(DeliveryPropertyDefinitions.SFTP_HOST, "mysftphost");
     // set username and password to access server
     req.addProperty(DeliveryPropertyDefinitions.SFTP_USERNAME, "myname");
     req.addProperty(DeliveryPropertyDefinitions.SFTP_PASSWORD, "mypassword");
     // set the remote directory that you want to send your document to
     req.addProperty(DeliveryPropertyDefinitions.SFTP_REMOTE_DIRECTORY, "pub");
     // set the remote file name
     req.addProperty(DeliveryPropertyDefinitions.SFTP_REMOTE_FILENAME,
"test.pdf");
     // set the document
     req.setDocument("/document/test.pdf");

     // submit the request
     req.submit();
     // close the request
     req.close();
```

*Table 8–8 Properties for Delivering Documents over SFTP*

| Property | Description |
|---|---|
| SFTP_HOST | Required<br>Enter the target server host name. |
| SFTP_PORT | Optional<br>Enter the target server SSH port number. Default is 22. |
| SFTP_USERNAME | Required<br>Enter the login user name. |
| SFTP_PASSWORD | Required if you choose the SFTP_AUTH_TYPE_PASSWORD authentication type.<br>Enter the login password. |
| SFTP_REMOTE_ DIRECTORY | Required<br>Enter the directory to which to deliver the document (example: /pub/). To deliver the document to the user's home directory, enter '.' |
| SFTP_REMOTE_ FILENAME | Required<br>Enter the document file name on the remote server. |
| SFTP_AUTH_TYPE | Set either of the following:<br>SFTP_AUTH_TYPE_PASSWORD (Default) Requires providing password at login.<br>SFTP_AUTH_TYPE_PUBLIC_KEY - public key authorization type. |
| SFTP_PRIVATE_KEY_ FILE | Enter the client private key file. Required if you choose SFTP_ AUTH_TYPE_PUBLIC_KEY. |
| SFTP_PRIVATE_KEY_ PASSWORD | Enter the client private key password. Required if you choose SFTP_AUTH_TYPE_PUBLIC_KEY. |
| SFTP_FILE_PERMISSION | Enter the permissions to set for the file being created. Default is 0755. |

## 8.9.1 Authentication Modes

The secure FTP delivery supports two authentication modes: password authentication and public key authentication. Set the property SFTP_AUTH_TYPE to choose the mode. The default mode is password authentication.

The password authentication mode requires the username and password to log in to the secure FTP server. The following example shows sample code:

*Example 8–16 Sample Code for Password Authentication*

```
:
         :
     // set password auth type
      req.addProperty(DeliveryPropertyDefinitions.SFTP_AUTH_TYPE,
                      DeliveryPropertyDefinitions.SFTP_AUTH_TYPE_PASSWORD);
      // set username and password to access server
      req.addProperty(DeliveryPropertyDefinitions.SFTP_USERNAME, "myname");
      req.addProperty(DeliveryPropertyDefinitions.SFTP_PASSWORD, "mypassword");
         :
         :
```

The public key authorization mode requires the username, your private key and password for the private key. This is a more secure method than the password authentication. Note that to use the public key authentication mode, you must set up the public key in the ssh/secure FTP server in advance. The following example shows sample code:

***Example 8–17 Sample Code for Public Key Authentication***

```
         :
         :
      // set public key auth type
      req.addProperty(DeliveryPropertyDefinitions.SFTP_AUTH_TYPE,
                      DeliveryPropertyDefinitions.SFTP_AUTH_TYPE_PUBLIC_KEY);
      // set username
      req.addProperty(DeliveryPropertyDefinitions.SFTP_USERNAME, "myname");
      // set the client's private key file
      req.addProperty(DeliveryPropertyDefinitions.SFTP_PRIVATE_KEY_FILE,
                      "/path/to/the/key");
      // set the client's private key password
      req.addProperty(DeliveryPropertyDefinitions.SFTP_PRIVATE_KEY_PASSWORD,
"myPrivateKeyPass");
         :
         :
```

## 8.10 Delivering Your Documents over Hypertext Transfer Protocol (HTTP)

The Delivery Manager supports delivery of documents to HTTP servers. The following sample sends a document through the HTTP POST method. Note that the receiving HTTP server must be able to accept your custom HTTP request in advance (for example through a custom servlet or CGI program).

***Example 8–18 Sample Code for Delivering Documents over HTTP***

```
// create delivery manager instance
     DeliveryManager dm = new DeliveryManager();
     // create a delivery request
     DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_HTTP);

     // set request method
     req.addProperty(DeliveryPropertyDefinitions.HTTP_METHOD,
DeliveryPropertyDefinitions.HTTP_METHOD_POST);
     // set document content type
     req.addProperty(DeliveryPropertyDefinitions.HTTP_CONTENT_TYPE,
"application/pdf");
     // set the HTTP server hostname
     req.addProperty(DeliveryPropertyDefinitions.HTTP_HOST, "myhost");
     // set the HTTP server port number
     req.addProperty(DeliveryPropertyDefinitions.HTTP_PORT, "80");
     // set the target remote directory
     req.addProperty(DeliveryPropertyDefinitions.HTTP_REMOTE_DIRECTORY,
"/servlet/");
     // set the remote filename (servlet class)
     req.addProperty(DeliveryPropertyDefinitions.HTTP_REMOTE_FILENAME,
"uploadDocument");

     // set the document
     req.setDocument("/document/test.pdf");
```

```
// submit the request
req.submit();
// close the request
req.close();
```

The following table lists the properties that are supported. A String value is required for each property unless otherwise noted.

***Table 8–9 Properties for Delivering Documents over HTTP***

| **Property** | **Description** |
|---|---|
| HTTP_METHOD | Optional<br>Sets the HTTP request method. Valid values are:<br>HTTP_METHOD_POST (Default)<br>HTTP_METHOD_PUT |
| HTTP_CONTENT_TYPE | Optional<br>The document content type (example: "application/pdf"). |
| HTTP_HOST | Required<br>Enter the server host name. |
| HTTP_PORT | Optional<br>Enter the server port number. The default is 80. |
| HTTP_REMOTE_DIRECTORY | Required<br>Enter the remote directory name (example: "/home/"). |
| HTTP_REMOTE_FILENAME | Required<br>Enter the file name to save the document as in the remote directory. |
| HTTP_AUTHTYPE | Optional<br>Valid values for authentication type are:<br>HTTP_AUTHTYPE_NONE - no authentication (default)<br>HTTP_AUTHTYPE_BASIC - use basic HTTP authentication<br>HTTP_AUTHTYPE_DIGEST - use digest HTTP authentication |
| HTTP_USERNAME | Optional<br>If the server requires authentication, enter the username. |
| HTTP_PASSWORD | Optional<br>If the server requires authentication, enter the password for the username. |

***Table 8–9 (Cont.) Properties for Delivering Documents over HTTP***

| Property | Description |
|---|---|
| HTTP_ENCTYPE | Optional<br>Enter the encryption type:<br>HTTP_ENCTYPE_NONE - no encryption (default)<br>HTTP_ENCTYPE_SSL - use Secure Socket Layer |
| HTTP_USE_FULL_URL | Optional<br>Set to "true" to send the full URL for the HTTP request header. Valid values are "true" or "false" (default). |
| HTTP_USE_CHUNKED_BODY | Optional<br>Valid values are "true" (default) to use HTTP chunked transfer coding for the message body, or "false". |
| HTTP_TIMEOUT | Optional<br>Enter a length of time in milliseconds after which to terminate the request if a connection is not made to the HTTP server. The default is 60000 (1 minute). |
| HTTP_URL_CHARACTER_ENCODING | Encoding of the URL. It will be used if you use non-ASCII characters in the URL. Set the Java-supported encoding string for the value. |

## 8.11 Delivering Documents over AS2

AS2 is one of the standard protocols defined in the Electronic Data Interchange-Internet Integration (EDI-INT). AS2 is based on HTTP and other internet standard technologies and is designed to exchange data over the internet in a secure manner. The AS2 specification is defined in RFC4130 (available at `http://www.ietf.org/`). The delivery system supports the delivery of documents to AS2 servers. Sample code is as follows:

***Example 8–19 Sample Code for Delivering Documents over AS2***

```
// create delivery manager instance
    DeliveryManager dm = new DeliveryManager();
    // create a delivery request
    DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_AS2);

    // set AS2 message properties
    req.addProperty(DeliveryPropertyDefinitions.AS2_FROM, "Me");
    req.addProperty(DeliveryPropertyDefinitions.AS2_TO, "You");
    req.addProperty(DeliveryPropertyDefinitions.AS2_SUBJECT, "My EDI Message");
    req.addProperty(DeliveryPropertyDefinitions.AS2_CONTENT_TYPE,
"applications/EDIFACT");

    // set HTTP properties
    req.addProperty(DeliveryPropertyDefinitions.AS2_HTTP_HOST, "as2hsot");
    req.addProperty(DeliveryPropertyDefinitions.AS2_HTTP_REMOTE_DIRECTORY, "/");
    req.addProperty(DeliveryPropertyDefinitions.AS2_HTTP_REMOTE_FILENAME, "as2");

    // set the document
    req.setDocument("/document/myEDIdoc");
```

```
// submit the request
DeliveryResponse res = req.submit();
// close the request
req.close();
```

The following table lists the supported properties. A string value is required for each property unless otherwise noted.

***Table 8–10 Properties for Delivering Documents over AS2***

| Property | Description |
|---|---|
| AS2_FROM | Required.<br>Enter the AS2 message sender. |
| AS2_TO | Required.<br>Enter the AS2 message recipient. |
| AS2_SUBJECT | Required.<br>Enter the message subject. |
| AS2_MESSAGE_ COMPRESSION | Default value is False. Enter True to compress the message. |
| AS2_MESSAGE_ SIGNATURE | Default value is False. Enter True to sign the message. |
| AS2_MESSAGE_ ENCRYPTION | Default value is False. Enter True to encrypt the message. |
| AS2_CONTENT_TYPE | Required.<br>Enter the content type of the document. Valid values are:<br>■ application/EDIFACT<br>■ application/xml |
| AS2_ENC_ALGO | The AS2 encryption algorithm. Set one of the following:<br>■ AS2_ENC_ALGO_RC2_40<br>■ AS2_ENC_ALGO_RC2_64<br>■ AS2_ENC_ALGO_RC2_128<br>■ AS2_ENC_ALGO_DES<br>■ AS2_ENC_ALGO_DES_EDE3 (Default)<br>■ AS2_ENC_ALGO_AES_128<br>■ AS2_ENC_ALGO_AES_192<br>■ AS2_ENC_ALGO_AES_256 |
| AS2_DIGEST_ALGO | Enter the AS2 digest algorithm for signing the messages. Set either of the following:<br>■ AS2_DIGEST_ALGO_MD5 (Default)<br>■ AS2_DIGEST_ALGO_SHA1 |
| AS2_ASYNC_ADDRESS | Enter the asynchronous address to which MDN notifications should be set. |
| AS2_ASYNC_EMAIL_ SERVER_HOST | Enter the email server host for asynchronous email MDN. |
| AS2_ASYNC_EMAIL_ SERVER_PORT | Enter the email server port for asynchronous email MDN. |
| AS2_ASYNC_EMAIL_ SERVER_USERNAME | Enter the email server USERNAME for asynchronous email MDN. |

***Table 8–10 (Cont.) Properties for Delivering Documents over AS2***

| **Property** | **Description** |
|---|---|
| AS2_ASYNC_EMAIL_SERVER_PASSWORD | Enter the email server PASSWORD for asynchronous email MDN. |
| AS2_ASYNC_EMAIL_SERVER_FOLDER_NAME | Enter the IMAP folder name for asynchronous email MDN. |
| AS2_SENDER_PKCS12_FILE | Location of the sender's PKCS12 (public/private key) file. |
| AS2_SENDER_PKCS12_PASSWORD | Password for the sender's PKCS12 (public/private key). |
| AS2_RECEIVER_CERTIFICATES_FILE | Location of the receiver's certificates file. |
| AS2_DELIVERY_RECEIPT_DIRECTORY | Directory to store the delivery receipts. This directory must be specified if to receive delivery receipts. |
| AS2_HTTP_HOST | Required.<br>Enter the server host name. |
| AS2_HTTP_PORT | Enter the server HTTP port number. The default is 80. |
| AS2_HTTP_REMOTE_DIRECTORY | Required.<br>Enter the remote directory name. (Example: /home/) |
| AS2_HTTP_REMOTE_FILENAME | Required.<br>Enter the remote file name. |
| AS2_HTTP_AUTHTYPE | Enter the HTTP authentication type. Valid values are:<br>■ AS2_HTTP_AUTHTYPE_NONE - no authentication (Default)<br>■ AS2_HTTP_AUTHTYPE_BASIC - Use HTTP basic authentication.<br>■ AS2_HTTP_AUTHTYPE_DIGEST - user HTTP digest authentication. |
| AS2_HTTP_USERNAME | Enter the username for HTTP authentication. |
| AS2_HTTP_PASSWORD | Enter the password for HTTP authentication. |
| AS2_HTTP_ENCTYPE | Set the encryption type. Valid values are:<br>■ AS2_HTTP_ENCTYPE_NONE - no encryption (default)<br>■ AS2_HTTP_ENCTYPE_SSL - use secure socket layer (SSL) |
| AS2_HTTP_TIMEOUT | Enter the time out allowance in milliseconds. Default is 60,000 (1 minute) |
| AS2_HTTP_PROXY_HOST | Required.<br>Enter the proxy server host name. |
| AS2_HTTP_PROXY_PORT | Enter the proxy server port number. Default is 80. |
| AS2_HTTP_PROXY_AUTHTYPE | ■ AS2_HTTP_AUTHTYPE_NONE - no authentication (Default)<br>■ AS2_HTTP_AUTHTYPE_BASIC - Use HTTP basic authentication.<br>■ AS2_HTTP_AUTHTYPE_DIGEST - user HTTP digest authentication. |

***Table 8–10 (Cont.) Properties for Delivering Documents over AS2***

| Property | Description |
|---|---|
| AS2_HTTP_PROXY_ USERNAME | Enter the username for proxy authentication. |
| AS2_HTTP_PROXY_ PASSWORD | Enter the password for HTTP proxy authentication. |

## 8.11.1 Delivery Receipt

The AS2 server always issues an AS2 delivery receipt for each AS2 request. Set the AS2_DELIVERY_RECEIPT_DIRECTORY property to specify the location to store the delivery receipts. If you do not specify this directory, delivery receipts will be ignored. Sample code for setting the delivery receipt directory is as follows:

***Example 8–20 Sample Code for Setting the Delivery Receipt Directory***

```
         :
         :
 // Set the delivery receipt directory
 req.addProperty(DeliveryPropertyDefinitions.AS2_DELIVERY_RECEIPT_DIRECTORY,
"/my/receipt/dir");
         :
         :
```

## 8.11.2 Synchrony

You can send either synchronous or asynchronous delivery requests to the AS2 servers. By default, the request is synchronous so that you can see the Message Disposition Notification (MDN) immediately in the DeliveryResponse.

If you set the AS2_ASYNC_ADDRESS to your request, the request will be asynchronous. You can specify either an HTTP URL or an e-mail address where the delivery receipt will be delivered after processing. You must set up the HTTP server or e-mail address to receive the delivery receipts.

The Delivery API can track down the asynchronous request if you specify the e-mail address for the AS2_ASYNC_ADDRESS. If you provide the e-mail account information to the Delivery API, the Delivery API will periodically check the e-mail account to obtain the delivery receipt. Sample code for this is as follows:

***Example 8–21 Sample Code for Sending Asynchronous Delivery Requests***

```
         :
         :
 // Set the email address - async request
 req.addProperty(DeliveryPropertyDefinitions.AS2_ASYNC_ADDRESS, "async_
target@acme.com");

 // Set the delivery receipt directory
 req.addProperty(DeliveryPropertyDefinitions.AS2_DELIVERY_RECEIPT_DIRECTORY,
"/my/receipt/dir");

 // Set the email server information where the delivery receipt will be delivered
to.
 req.addProperty(
     DeliveryPropertyDefinitions.AS2_ASYNC_EMAIL_SERVER_HOST, "mail.acme.com");
 req.addProperty(
     DeliveryPropertyDefinitions.AS2_ASYNC_EMAIL_SERVER_USERNAME, "async_target");
 req.addProperty(
     DeliveryPropertyDefinitions.AS2_ASYNC_EMAIL_SERVER_PASSWORD, "mypassword");
 req.addProperty(
     DeliveryPropertyDefinitions.AS2_ASYNC_EMAIL_SERVER_FOLDER_NAME, "inbox");

 // set the document
 req.setDocument("/document/myEDIdoc");

 // submit the request with the DeliveryResponseListener
 req.submit(myDeliveryListener);
         :
         :
```

Note that as shown in the preceding code, you must use the Delivery APIs asynchronous delivery request mechanism to track down the asynchronous requests. See Section 8.15, "Asynchronous Delivery Requests."

### 8.11.3 Document Signing

The Delivery API enables you to sign a document for the secure transaction. This is based on the public key architecture, so you must set up the following:

- Sender side: sender's public/private keys

  Sender must have sender's public/private keys in a PKCS12 standard file. The file extension is .p12. Place that file in your local system where you want to run the Delivery API.

- Receiver side (AS2 server side): sender's public key certificate

  The receiver must have the sender's public key certificate. Installing certificates on the AS2 server can vary depending on your server. Generally, you must copy the .der or .cer certificates to a particular location. Consult your AS2 server manual for the procedure.

Once you have completed the setup, you can sign your document by setting properties in the delivery request. Sample code for this is as follows:

***Example 8–22 Sample Code for Signing Documents***

```
         :
         :
 // Signing the document
 req.addProperty(DeliveryPropertyDefinitions.AS2_MESSAGE_SIGNATURE, "true");
 req.addProperty(DeliveryPropertyDefinitions.AS2_SENDER_PKCS12_FILE,
"/path/to/mykey.p12");
 req.addProperty(DeliveryPropertyDefinitions.AS2_SENDER_PKCS12_PASSWORD,
"welcome");
         :
         :
```

### 8.11.4 Document Encryption

The Delivery API enables you to encrypt documents for the secure transaction. This is based on the public key architecture, so you need to set up the following:

- Sender's side: Receiver's public key certificate

The sender side must have the receiver's public key certificate file. The file extension is .der or .cer. Place that file in your local system where you want to run the Delivery API. Please consult the manual of your AS2 server for the procedure to obtain the AS2 server's public key certificate.

- Receiver's side (AS2 server side): Receiver's public/private keys

  The receiver side (AS2 Server) must have the receiver's public/private keys. Please consult the manual of your AS2 server for the procedure to set up keys.

Once set up, you can encrypt your document by setting properties in the delivery request. The sample code is as follows:

***Example 8–23 Sample Code for Encrypting Documents***

```
        :
        :
 // Encrypting the document
 req.addProperty(DeliveryPropertyDefinitions.AS2_MESSAGE_ENCRYPTION, "true");
 req.addProperty(DeliveryPropertyDefinitions.AS2_RECEIVER_CERTIFICATES_FILE,
"/path/to/server-certificate.der");
        :
        :
```

## 8.12 Delivering Documents Using an External Command

The Delivery API supports the use of external, operating system (OS) native commands to deliver documents.

Specify your OS native command with the `{file}` placeholder. At run time, this placeholder will be replaced with the document file name.

The delivery status is determined by the exit value of the OS command. If the value is '0', the request is marked successful.

Sample code is as follows:

***Example 8–24 Sample Code for Delivering Documents Using External Commands***

```
// create delivery manager instance
DeliveryManager dm = new DeliveryManager();
// create a delivery request
DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_EXTERNAL);
// set the OS native command for delivery
req.addProperty(DeliveryPropertyDefinitions.EXTERNAL_DELIVERY_COMMAND,
"/usr/bin/lp -d myprinter {file}");
// set the document
req.setDocument("/document/test.pdf");

// submit the request
req.submit();
// close the request
req.close();
```

The following property is supported and defined in DeliveryPropertyDefinitions:

***Table 8–11 Properties for Delivering Documents Using External Commands***

| Property | Description |
|---|---|
| EXTERNAL_DELIVERY_COMMAND | Required.<br>Enter the OS native command for delivery. |

## 8.13 Delivering Documents to the Local File System

The Delivery API supports the delivery of documents to the local file system where the Delivery API runs. The command copies the file to the location you specify.

The following sample code copies the file /document/test.pdf to /destination/document.pdf:

***Example 8–25 Sample Code for Delivering Documents to Local File Systems***

```
     // create delivery manager instance
     DeliveryManager dm = new DeliveryManager();
     // create a delivery request
     DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_LOCAL);
     // set the document destination in the local filesystem.
     req.addProperty(DeliveryPropertyDefinitions.LOCAL_DESTINATION,
"/destination/document.pdf");
     // set the document to deliver.
     req.setDocument("/document/test.pdf");

     // submit the request
     req.submit();
     // close the request
     req.close();
```

The following property is supported and defined in DeliveryPropertyDefinitons:

***Table 8–12 Properties for Delivering Documents to Local File Systems***

| Property | Description |
|---|---|
| LOCAL_DESTINATION | Required.<br>Full path to the destination file name in the local file system. |

## 8.14 Direct and Buffering Modes

The delivery system supports two modes: direct mode and buffering mode. Buffering mode is the default mode.

### 8.14.1 Direct Mode

Direct Mode offers full, streamlined delivery processing. Documents are delivered to the connection streams that are directly connected to the destinations. This mode is fast, and uses less memory and disk space. It is recommended for online interactive processing.

To set the direct mode, set the BUFFERING_MODE property to "false". Following is a code sample:

***Example 8–26 Sample Code for Setting Direct Mode***

```
// create delivery manager instance
     DeliveryManager dm = new DeliveryManager();
     // create a delivery request
     DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_IPP_PRINTER);

     // set the direct mode
     req.addProperty(DeliveryPropertyDefinitions.BUFFERING_MODE, "false");
           :
           :
           :
```

This mode does not offer document redelivery. For redelivery requirements, use the buffering mode.

### 8.14.2 Buffering Mode

The buffering mode enables you to redeliver documents as many times as you want. The delivery system uses temporary files to buffer documents, if you specify a temporary directory (`ds-temp-dir`) in the delivery server configuration file. If you do not specify a temporary directory, the delivery system uses the temporary memory buffer. It is recommended that you define a temporary directory. For more information about the configuration file, see Section 8.21, "Configuration File Support."

You can explicitly clear the temporary file or buffer by calling `DeliveryRequest.close()` after finishing your delivery request.

***Example 8–27 Sample Code for Setting Buffering Mode***

```
// create delivery manager instance
     DeliveryManager dm = new DeliveryManager();

     // create a delivery request
     DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_IPP_PRINTER);

     // set buffering mode
     req.addProperty(DeliveryPropertyDefinitions.BUFFERING_MODE, "true");
     req.addProperty(DeliveryPropertyDefinitions.TEMP_DIR, "/tmp");
           :
           :
           :
     // submit request
     req.submit();
           :
           :
     // submit request again
     req.submit();
           :
           :
     // close the request
     req.close();
```

## 8.15 Asynchronous Delivery Requests

The Delivery API provides the ability to run the delivery requests asynchronously by registering the callback functions.

You can create your own callback logic by implementing the DeliveryResponseListener interface. You must implement the responseReceived()

method. You can implement your logic in this method so that it will be called when the delivery request is finished. Sample code is as follows:

***Example 8–28 Sample Code for Implementing Callback Logic***

```
import oracle.apps.xdo.delivery.DeliveryResponseListener;

class MyListener implements DeliveryResponseListener
{

  public void responseReceived(DeliveryResponse pResponse)
  {
    // Show the status to the System.out
    System.out.println("Request done!");
    System.out.println("Request status id  : " + pResponse.getStatus());
    System.out.println("Request status msg : " + pResponse.getStatusMessage());
  }

}
```

Once you implement the callback, you can pass your callback when you call the submit() method of your DeliveryRequest. If you call the submit() with the callback, the delivery process will start in the background and the submit() method will immediately return the control. Sample code follows:

***Example 8–29 Sample Code for Submitting Callback Logic***

```
  // create delivery manager instance
  DeliveryManager dm = new DeliveryManager();

  // create a delivery request
  DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_IPP_PRINTER);
        :
        :
  // submit request with the callback logic
  req.submit(new MyListener());
        :
        :
```

## 8.16 Document Filter Support

The Delivery API supports the document filter functionality for all the supported protocols. This functionality enables you to call the native operating system (OS) command to transform the document before each delivery request. To specify the filter, pass the native OS command string with the two placeholders for the input and output filename: `{infile}` and `{outfile}`. You can set your filter in your delivery request as a delivery property. Following are two samples:

***Example 8–30 Sample Code for Setting Document Filter as Delivery Property***

```
// The easiest filter, just copy the file :)
req.addProperty(DeliveryPropertyDefinitions.FILTER, "cp {infile} {outfile}");


// Call "pdftops" utility to transform the PDF document into Postscript format
 req.addProperty(DeliveryPropertyDefinitions.FILTER, "pdftops {infile}
{outfile}");
```

Alternatively, you can also specify the filter for each server in the configuration file (see Section 8.21, "Configuration File Support"). In this case, the server will always use this filter for the requests to this server:

***Example 8–31 Sample Code for Setting Document Filter in Configuration File***

```
         :
         :

<server name="printer1" type="ipp_printer" default="true">
<uri>ipp://myserver:80/printers/MyPrinter1/.printer</uri>
<filter>pdftops {infile} {outfile}</filter>
</server>
         :
         :
```

This is useful especially if you are trying to call IPP printers directly or IPP printers on Microsoft Internet Information Service (IIS) because those printers usually do not accept PDF documents, but only limited document formats. With this functionality, you can call any of the native operating system (OS) commands to transform the document to the format that the target printer can understand. For example, if you need to call the HP LaserJet printer setup on the Microsoft IIS from Linux, you can set Ghostscript as a filter to transform the PDF document into the format that the HP LaserJet can understand.

***Example 8–32 Sample Code for Setting Document Filter through OS Commands***

```
// specify filter
req.addProperty(DeliveryPropertyDefinitions.FILTER,
"gs -q -dNOPAUSE -dBATCH -sDEVICE=laserjet -sOutputFile={outfile}
 {infile}");
```

Note that to use this functionality you must set the buffering mode must be enabled and a temporary directory must be specified. See Section 8.21, "Configuration File Support."

### 8.16.1 PDF-to-PostScript Conversion Filter

In addition, BI Publisher provides a PDF-to-Postscript Level 2 conversion filter. You do not need to set {infile} and {outfile} place holders to use this internal filter, instead, directly specify the filter class as shown below:

***Example 8–33 Sample for Setting the PDF-to-Postscript Level 2 Conversion Filter***

```
req.addProperty(DeliveryPropertyDefinitions.FILTER,
"oracle.xdo.delivery.filter.PDF2PSFilterImpl");
<server name="printer1" type="ipp_printer" default="true">
ipp://myserver:80/printers/MyPrinter1/.printer
<filter>oracle.xdo.delivery.filter.PDF2PSFilterImpl</filter>
</server>
```

## 8.17 Date Expression Support

BI Publisher provides properties that support date expressions. Use date expressions if you want to name a file by the date, and have the date automatically set at run time.

The following properties support date expressions:

- SMTP_CONTENT_FILENAME
- FTP_REMOTE_FILENAME
- WEBDAV_REMOTE_FILENAME

The supported date expressions are:

- %y : 4 digit year (ex, 1972, 2005)
- %m : 2 digit month (00 - 12)
- %d : 2 digit date (00 - 31)
- %H : 24h based 2 digit hour (00 - 24)
- %M : 2 digit minute (00 - 59)
- %S : 2 digit sec (00 - 59)
- %l : 3 digit millisec (000 - 999)

For example, if you specify `my_file_%y%m%d.txt` for the filename, the actual filename will would be `my_file_20051108.txt` for November 8, 2005. All undefined expressions will be translated into 0 length string, for example, if you specify `my_file_%a%b%c.txt`, it would generate `my_file_.txt`. You can escape the '%' character by passing '%%'.

## 8.18 Internationalization Support

The Delivery Server API supports following internationalization features for the listed delivery channels:

### 8.18.1 SMTP

- Specify character encoding for the main document with SMTP_CONTENT_TYPE.
- Specify character encoding for the attachments by passing content type when you call addAttachment() method.
- Specify the character encoding for email To/From/CC/Subject with SMTP_CHARACTER_ENCODING property. The default value is "UTF-8".

### 8.18.2 IPP

- Specify character encoding for the IPP attributes by using IPP_ATTRIBUTE_CHARSET property. The default value is "UTF-8".
- Specify IPP_URL_CHARACTER_ENCODING property for encoding non-ASCII letters in a URL.

### 8.18.3 WebDAV

- Specify WEBDAV_URL_CHARACTER_ENCODING property for encoding non-ASCII letters in a URL.

### 8.18.4 FTP

- The FTP delivery channel automatically detects the internationalization support in the target FTP server. You can specify a non-ASCII directory name and file name only if the FTP server supports internationalization (see RFC 2640 for more detail). In that case, the UTF-8 encoding will be used automatically. If the server does not support internationalization and you specify a non-ASCII value, an exception will be thrown during the delivery process.

### 8.18.5 HTTP

- You can specify HTTP_CHARACTER_ENCODING property for encoding non-ASCII letters in a URL.

## 8.19 Setting Global Properties

You can define the global properties to the DeliveryManager so that all the delivery requests inherit the global properties automatically.

The following global properties are supported:

***Table 8–13 Global Properties Supported by the DeliveryManager API***

| Property | Description |
|---|---|
| BUFFERING_MODE | Valid values are "true" (default) and "false". See Section 8.14, "Direct and Buffering Modes." |
| TEMP_DIR | Define the location of the temporary directory. |
| CA_CERT_FILE | Define the location of the CA Certificate file generated by Oracle Wallet Manager. This is used for SSL connection with the Oracle SSL library. If not specified, the default CA Certificates are used. |

***Example 8–34 Sample Code for Setting Global Properties***

```
// create delivery manager instance
     DeliveryManager dm = new DeliveryManager();

     // set global properties
     dm.addProperty(DeliveryPropertyDefinitions.TEMP_DIR, "/tmp");
     dm.addProperty(DeliveryPropertyDefinitions.BUFFERING_MODE, "true");

     // create delivery requests
     DeliveryRequest req1 = dm.createRequest(DeliveryManager.TYPE_IPP_PRINTER);
     DeliveryRequest req2 = dm.createRequest(DeliveryManager.TYPE_IPP_FAX);
     DeliveryRequest req3 = dm.createRequest(DeliveryManager.TYPE_SMTP_EMAIL);
           :
           :
```

## 8.20 Adding a Custom Delivery Channel

You can add custom delivery channels to the system by following the steps below:

1. Define the delivery properties
2. Implement the DeliveryRequest interface
3. Implement the DeliveryRequestHandler interface
4. Implement the DeliveryRequestFactory interface
5. Register your custom DeliveryRequestFactory to the DeliveryManager

The following sections detail how to create a custom delivery channel by creating a sample called "File delivery channel" that delivers documents to the local file system.

### 8.20.1 Define Delivery Properties

The first step to adding a custom delivery channel is to define the properties. These will vary depending on what you want your channel to do. You can define constants for your properties. Our example, a file delivery channel requires only one property, which is the destination.

Sample code is:

***Example 8–35 Sample Code for Defining Delivery Channel Properties***

```
package oracle.apps.xdo.delivery.file;

public interface FilePropertyDefinitions
  {
     /** Destination property definition.  */
     public static final String FILE_DESTINATION = "FILE_DESTINATION:String";

   }
```

The value of each constant can be anything, if it is a String. It is recommend that you define the value in `[property name]:[property value type]` format so that the delivery system automatically validates the property value at run time. In the example, the `FILE_DESTINATION` property is defined to have a String value.

### 8.20.2 Implement DeliveryRequest Interface

DeliveryRequest represents a delivery request that includes document information and delivery metadata, such as destination and other properties. To implement oracle.apps.xdo.delvery.DeliveryRequest you can extend the class oracle.apps.xdo.delivery.AbstractDeliveryRequest.

For example, to create a custom delivery channel to deliver documents to the local file system, the DeliveryRequest implementation will be as follows:

***Example 8–36 Sample Code for Delivering Documents to a Local File System through a Custom Delivery Channel***

```
package oracle.apps.xdo.delivery.file;
import oracle.apps.xdo.delivery.AbstractDeliveryRequest;

public class FileDeliveryRequest extends AbstractDeliveryRequest
implements FilePropertyDefinitions
{
  private static final String[] MANDATORY_PROPS = {FILE_DESTINATION};

  /**
   * Returns mandatory property names
   */
  public String[] getMandatoryProperties()
  {
    return MANDATORY_PROPS;
  }
  /**
   * Returns optional property names
   */
  public String[] getOptionalProperties()
  {
    return null;
  }
}
```

### 8.20.3 Implement DeliveryRequestHandler Interface

DeliveryRequestHandler includes the logic for handling the delivery requests. A sample implementation of oracle.apps.xdo.delivery.DeliveryRequestHandler for the file delivery channel is as follows:

***Example 8–37 Sample Code for Implementing the DeliveryRequestHandler Interface***

```
package oracle.apps.xdo.delivery.file;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;

import oracle.apps.xdo.delivery.DeliveryException;
import oracle.apps.xdo.delivery.DeliveryRequest;
import oracle.apps.xdo.delivery.DeliveryRequestHandler;
import oracle.apps.xdo.delivery.DeliveryStatusDefinitions;

public class FileDeliveryRequestHandler implements DeliveryRequestHandler
{

  private FileDeliveryRequest mRequest;
  private boolean mIsOpen = false;
  private OutputStream mOut;

  /**
   * default constructor.
   */
  public FileDeliveryRequestHandler()
  {
  }

  /**
   * sets the request.
   */
  public void setRequest(DeliveryRequest pRequest)
  {
    mRequest = (FileDeliveryRequest) pRequest;
  }

  /**
   * returns the request.
   */
  public DeliveryRequest getRequest()
  {
    return mRequest;
  }

  /**
   * opens the output stream to the destination.
   */
  public OutputStream openRequest() throws DeliveryException
  {
    try
    {
      String filename =
        (String) mRequest.getProperty(FileDeliveryRequest.FILE_DESTINATION);
      mOut = new BufferedOutputStream(new FileOutputStream(filename));

      mIsOpen = true;
      // set request status to open
      mRequest.setStatus(DeliveryStatusDefinitions.STATUS_OPEN);
      return mOut;

    }
    catch (IOException e)
    {
      closeRequest();
      throw new DeliveryException(e);
    }

  }

  /**
   * flushes and closes the output stream to submit the request.
   */
  public void submitRequest() throws DeliveryException
  {
    try
    {
      // flush and close
      mOut.flush();
      mOut.close();
      // set request status
      mRequest.setStatus(DeliveryStatusDefinitions.STATUS_SUCCESSFUL);
      mIsOpen = false;
    }
    catch (IOException e)
    {
      closeRequest();
      throw new DeliveryException(e);
    }
  }

  /**
   * checks the delivery status.
   */
  public void updateRequestStatus() throws DeliveryException
  {

    // check if the file is successfully delivered
    String filename =
      (String) mRequest.getProperty(FileDeliveryRequest.FILE_DESTINATION);
    File f = new File(filename);

    // set request status
    if (f.exists())
      mRequest.setStatus(DeliveryStatusDefinitions.STATUS_SUCCESSFUL);
    else
      mRequest.setStatus(DeliveryStatusDefinitions.STATUS_FAILED_IO_ERROR);

  }
  /**
   * returns the request status.
   */
  public boolean isRequestOpen()
  {
    return mIsOpen;
  }

  /**
   * closes the request, frees all resources.
   */
  public void closeRequest()
  {
    mIsOpen = false;
    try
    {
      if (mOut != null)
      {
        mOut.flush();
        mOut.close();
      }
    }
    catch (IOException e)
    {
    }
    finally
    {
      mOut = null;
    }
  }

}
```

## 8.20.4 Implement DeliveryRequestFactory Interface

Implement the DeliveryRequestFactory interface to register your custom delivery channel to the delivery system.

A sample implementation of oracle.apps.xdo.delivery.DeliveryRequestFactory is as follows:

***Example 8–38 Sample Code for Implementing the DeliveryRequestFactory Interface***

```
package oracle.apps.xdo.delivery.file;

import oracle.apps.xdo.delivery.DeliveryRequest;
import oracle.apps.xdo.delivery.DeliveryRequestFactory;
import oracle.apps.xdo.delivery.DeliveryRequestHandler;

public class FileDeliveryRequestFactory
implements DeliveryRequestFactory
{
  /**
   * default constructor.
   */
  public FileDeliveryRequestFactory()
  {
  }
  /**
   * returns delivery request.
   */
  public DeliveryRequest createRequest()
  {
    return new FileDeliveryRequest();
  }
  /**
   * returns delivery request handler.
   */
  public DeliveryRequestHandler createRequestHandler()
  {
    return new FileDeliveryRequestHandler();
  }
  /**
   * returns this
   */
  public DeliveryRequestFactory getFactory()
  {
    return this;
  }
}
```

## 8.20.5 Register your custom DeliveryRequestFactory to DeliveryManager

The final step is to register your custom delivery channel to the delivery system. You can register your delivery channel in two ways:

- Static method

  Use this method to register your delivery channel to the whole delivery system by specifying it in the configuration file. See Section 8.21, "Configuration File Support."

- Dynamic method

  Register the delivery channel to the Java VM instance by calling the Register API programmatically.

  Sample code to register the file delivery channel using the dynamic method and call the file delivery channel is as follows:

***Example 8–39 Sample Code for Registering and Calling File Delivery Channel Using the Dynamic Method***

```
package oracle.apps.xdo.delivery.file;

import oracle.apps.xdo.delivery.DeliveryManager;
import oracle.apps.xdo.delivery.DeliveryRequest;

public class FileDeliverySample
{
  public static void main(String[] args) throws Exception
  {
    // register the file delivery channel
    DeliveryManager.addRequestFactory("file",
"oracle.apps.xdo.delivery.file.FileDeliveryRequestFactory");

    // create delivery manager instance
    DeliveryManager dm = new DeliveryManager();
    // create a delivery request
    DeliveryRequest req = dm.createRequest("file");

    // set the destination
```

```
        req.addProperty(
          FileDeliveryRequest.FILE_DESTINATION,
          "d:/Temp/testDocument_delivered.pdf");
        // set the document to deliver
        req.setDocument("D:/Temp/testDocument.pdf");

        // submit the request
        req.submit();
        // close the request
        req.close();
      }
    }
```

## 8.21 Configuration File Support

The delivery systems supports a configuration file to set default servers, default properties, and custom delivery channels. The location of the configuration file is

`{XDO_TOP}/resource/xdodelivery.cfg`

where `{XDO_TOP}` is a Java system property that points to the physical directory.

This system property can be set in two ways:

- Pass `-DXDO_TOP=/path/to/xdotop` to the Java startup parameter
- Use a Java API in your code, such as `java.lang.System.getProperties().put("XDO_TOP", "/path/to/xdotop")`

The system property must be defined before constructing a DeliveryManager object.

Following is a sample configuration file:

***Example 8–40 Sample Configuration File***

```
<?xml version='1.0' encoding='UTF-8'?>
 <config xmlns="http://xmlns.oracle.com/oxp/delivery/config">
   <! -  ========================================================  - >
   <! -     servers section                                          - >
   <! -     List your pre-defined servers here.                      - >

   <! -  ========================================================  - >
   <servers>
     <server name="myprinter1" type="ipp_printer" default="true">
       <uri>ipp://myprinter1.oracle.com:631/printers/myprinter1</uri>

     </server>
     <server name="myprinter2" type="ipp_printer" >
       <host>myprinter2.oracle.com</host>
       <port>631</port>

       <uri>ipp://myprinter2.oracle.com:631/printers/myprinter2</uri>
       <authType>basic</authType>
       <username>xdo</username>
       <password>xdo</password>

     </server>
     <server name="myfax1" type="ipp_fax" default="true" >
       <host>myfax1.oracle.com</host>
```

```
        <port>631</port>
        <uri>ipp://myfax1.oracle.com:631/printers/myfax1</uri>
      </server>
      <server name="mysmtp1" type="smtp_email" default="true">

        <host>myprinter1.oracle.com</host>
        <port>25</port>
      </server>
      <server name="mysmtp2" type="smtp_email" >

        <host>mysmtp12.oracle.com</host>
        <port>25</port>
        <username>xdo</username>
        <password>xdo</password>

      </server>
    </servers>
    <! -  ========================================================  - >
    <! -     properties section                                   - >
    <! -     List the system properties here.                     - >
    <! -  ========================================================  - >
    <properties>

      <property name="ds-temp-dir">/tmp</property>
      <property name="ds-buffering">true</property>
    </properties>
    <! -  ========================================================  - >
    <! -      channels section                                    - >

    <! -      List the custom delivery channels here.             - >
    <! -  ========================================================  - >
    <channels>
      <channel
name="file">oracle.apps.xdo.delivery.file.FileDeliveryRequestFactory</channel>
    </channels>

  </config>
```

## 8.21.1 Defining Multiple Servers for a Delivery Channel

You can define multiple server entries for each delivery channel. For example, the preceding sample configuration file has two server entries for the "ipp_printer" delivery channel ("myprinter1" and "myprinter2").

Load a server entry for a delivery request by calling DeliveryRequest.setServer() method. Following is an example:

***Example 8–41 Sample Code for Defining Multiple Servers for a Delivery Channel***

```
// create delivery manager instance
     DeliveryManager dm = new DeliveryManager();
     // create a delivery request
     DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_IPP_PRINTER);

     // load myprinter1 setting
     req.setServer("myprinter1");
```

### 8.21.2 Specifying a Default Server for a Delivery Channel

To define a default server for a delivery channel, specify default="true". In the configuration file example above, "myprinter1" is defined as the default sever for the "ipp_printer" delivery channel. If a user does not specify the server properties for "ipp_printer" delivery, the server properties under the default server will be used.

### 8.21.3 Supported Configuration File Properties and Elements

The following properties are supported in the `<properties>` section:

- `ds-temp-dir`: temporary directory location.
- `ds-buffering`: specify true or false for buffering mode.
- `ds-ca-cert-file`: specify the SSL certification file location.

The following elements are supported for `<server type="ipp_printer">` and `<server type="ipp_fax">`

- <host>
- <port>
- <printerName>
- <uri>
- <username>
- <password>
- <authType>
- <encType>
- <proxyHost>
- <proxyPort>
- <proxyUsername>
- <proxyPassword>
- <proxyAuthType>
- <filter>
- <filterOutputContentType>

The following elements are supported for `<server type="smtp_email">`

- <secureConnection>
- <host>
- <port>
- <username>
- <password>
- <authType>
- <filter>

The following elements are supported for `<server type="rightfax">`

- <host>
- <port>

- <uri>
- <username>
- <password>
- <authType>
- <encType>
- <proxyHost>
- <proxyPort>
- <proxyUsername>
- <proxyPassword>
- <proxyAuthType>
- <filter>
- <filterOutputContentType>

The following elements are supported for `<server type="printer">`

- <host>
- <filter>
- <filterOutputContentType>

The following elements are supported for `<server type="webdav">`

- <host>
- <port>
- <uri>
- <username>
- <password>
- <authType>
- <encType>
- <proxyHost>
- <proxyPort>
- <proxyUsername>
- <proxyPassword>
- <proxyAuthType>
- <filter>

The following elements are supported for `<server type="ftp">`

- <host>
- <port>
- <username>
- <password>
- <filter>
- <passiveMode>

The following elements are supported for `<server type="sftp">`

- <host>
- <port>
- <username>
- <password>
- <filter>
- <authType>

The following elements are supported for `<server type="http">`

- <host>
- <port>
- <uri>
- <username>
- <password>
- <authType>
- <encType>
- <proxyHost>
- <proxyPort>
- <proxyUsername>
- <proxyPassword>
- <proxyAuthType>

The following elements are supported for `<server type="as2">`

- <host>
- <port>
- <uri>
- <username>
- <password>
- <authType>
- <encType>
- <proxyHost>
- <proxyPort>
- <proxyUsername>
- <proxyPassword>
- <proxyAuthType>

The following elements are supported for `<server type="external">`

- <command>
- <filter>

# Part III

# Other Topics

This part contains the following chapters:

9

# Making a View Object Available to BI Publisher as a Data Source

This chapter describes the steps required to use a view object as a data source in BI Publisher.

It includes the following sections:

---

**Note:** This chapter assumes familiarity with Oracle Application Development Framework (ADF) and Oracle JDeveloper. For more information about these see:

- *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*
- *Oracle JDeveloper 11g Online Help*

---

## 9.1 Prerequisites

Oracle BI Publisher provides a mechanism to extract data from a remote server using Web service calls to a view object. Applications developers can define data sources as view objects in their application and then create a data model in BI Publisher to retrieve the data to use in their reports.

Following are the prerequisites for using the information in this chapter:

- BI Publisher is deployed to the WebLogic Server where the application will be running. The library "oracle.xdo.webapp" is required. The Oracle BI Platform Installer deploys this library.
- In JDeveloper, you have created the entity-based view object.
- In JDeveloper, the name of the database connection data source must be "ApplicationDB".

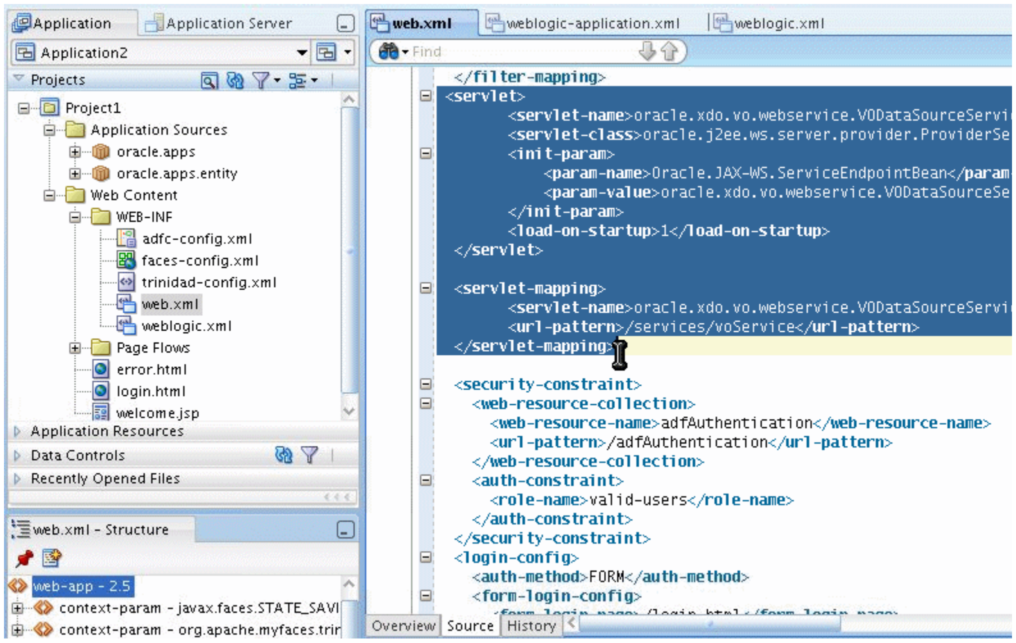
## 9.2 Configuring the Application Module

Using Oracle JDeveloper, configure the application module that contains the view object by following the steps in the remaining sections of this chapter.

## 9.3 Updating web.xml

1. In JDeveloper, navigate to the web.xml file under your Project > Web Content > WEB-INF folder.

***Figure 9–1 Updating the web.xml File in JDeveloper***



```
</filter-mapping>
<servlet>
        <servlet-name>oracle.xdo.vo.webservice.VODataSourceServi
        <servlet-class>oracle.j2ee.ws.server.provider.ProviderSe
        <init-param>
            <param-name>Oracle.JAX-WS.ServiceEndpointBean</param
            <param-value>oracle.xdo.vo.webservice.VODataSourceSe
        </init-param>
        <load-on-startup>1</load-on-startup>
 </servlet>

 <servlet-mapping>
        <servlet-name>oracle.xdo.vo.webservice.VODataSourceServi
        <url-pattern>/services/voService</url-pattern>
 </servlet-mapping>

 <security-constraint>
   <web-resource-collection>
     <web-resource-name>adfAuthentication</web-resource-name>
     <url-pattern>/adfAuthentication</url-pattern>
   </web-resource-collection>
   <auth-constraint>
     <role-name>valid-users</role-name>
   </auth-constraint>
 </security-constraint>
 <login-config>
   <auth-method>FORM</auth-method>
   <form-login-config>
```

2. In the Source view, update the web.xml file with the following:

```
<filter-mapping>
        <filter-name>adfBindings</filter-name>

<servlet-name>oracle.xdo.vo.webservice.VODataSourceService</servlet-name>
        <dispatcher>FORWARD</dispatcher>
        <dispatcher>REQUEST</dispatcher>
  </filter-mapping>

  <servlet>

<servlet-name>oracle.xdo.vo.webservice.VODataSourceService</servlet-name>

<servlet-class>oracle.j2ee.ws.server.provider.ProviderServlet</servlet-class>
        <init-param>
            <param-name>Oracle.JAX-WS.ServiceEndpointBean</param-name>

<param-value>oracle.xdo.vo.webservice.VODataSourceService</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
```

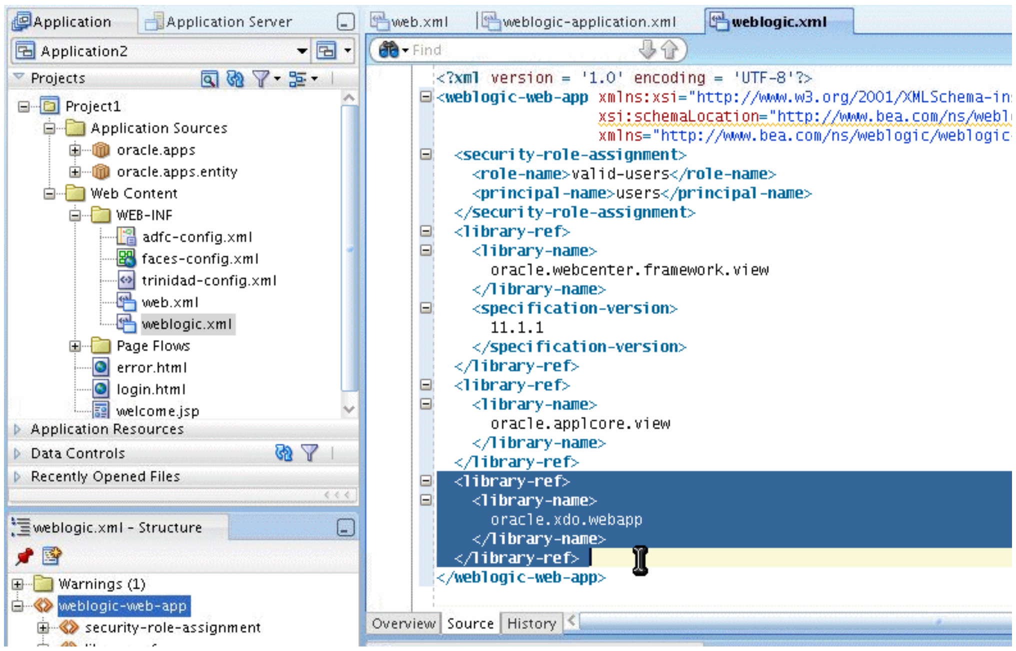
```
<servlet-name>oracle.xdo.vo.webservice.VODataSourceService</servlet-name>
        <url-pattern>/services/voService</url-pattern>
  </servlet-mapping>
```

## 9.4 Updating weblogic.xml

1. In JDeveloper, navigate to the weblogic.xml file under your Project > Web Content > WEB-INF folder.
2. Update the weblogic.xml file located in the WEB-INF directory with the following library reference:

```
<library-ref>
  <library-name>oracle.xdo.webapp</library-name>
</library-ref>
```

***Figure 9–2   Updating the weblogic.xml File in JDeveloper***



```
<?xml version = '1.0' encoding = 'UTF-8'?>
<weblogic-web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-in
                  xsi:schemaLocation="http://www.bea.com/ns/webl
                  xmlns="http://www.bea.com/ns/weblogic/weblogic
  <security-role-assignment>
    <role-name>valid-users</role-name>
    <principal-name>users</principal-name>
  </security-role-assignment>
  <library-ref>
    <library-name>
      oracle.webcenter.framework.view
    </library-name>
    <specification-version>
      11.1.1
    </specification-version>
  </library-ref>
  <library-ref>
    <library-name>
      oracle.applcore.view
    </library-name>
  </library-ref>
  <library-ref>
    <library-name>
      oracle.xdo.webapp
    </library-name>
  </library-ref>
</weblogic-web-app>
```

## 9.5 Deploying the Application Module

Deploy the application module to the WebLogic Server where BI Publisher is installed. Note the application context path.

## 9.6 Updating the providers.xml File

1. In your BI Publisher installation, navigate to the `providers.xml` file. The `providers.xml` file is located in `${xdo.server.config.dir}/repository/Admin/Configuration`.
2. Update the `providers.xml` file by providing a `name` for this data source and supplying the application context path in the `nonSSOUri` attribute as shown:

```
<provider name="MyWSVOTest" uri =""
nonSSOUri="http://example.com:7101/Application-VOTestWS-ViewController-context-
root"/>
```

3. Save the providers.xml file.
4. Restart the BI Publisher application.

   The view object data source will now be available from the data model editor.

   For instructions on how to create a data model for this data source, see the topic "Defining a View Object as a Data Set Type" in the *Oracle Fusion Middleware Report Designer's Guide for Oracle Business Intelligence Publisher*.

# 10 Setting Up After-Report Triggers

This chapter describes how to set up an after-report trigger using an HTTP servlet.

It includes the following sections:

- Section 10.1, "Overview of After-Report Triggers"
- Section 10.2, "Setting Up After-Report Triggers"

## 10.1 Overview of After-Report Triggers

BI Publisher enables you to set up an HTTP notification that will execute after-report generation as an after-report trigger. This enables you to integrate BI Publisher with other Oracle and third-party applications such as a BPEL process, Content Management applications, or other workflow applications.

BI Publisher supports Event triggers (Before Data and After Data triggers) in the Data Model Definition, which you can use to trigger programs during data generation. HTTP notification will trigger after the report is generated.

### 10.1.1 Limitations

Note that immediately upon the generation of the report in BI Publisher, the notification will execute. There is currently no ability to call back or introduce a listener or process between the report generation and the HTTP notification to your servlet.

### 10.1.2 Process Overview for Adding After-Report Triggers to Reports

The following tasks are required to complete the setup of an after-report trigger for your report:

1. Create your servlet or third-party application, as described in this chapter.
2. Register your servlet URL as an HTTP delivery server in the BI Publisher Administration page. See Section 10.2.1, "Registering the HTTP Servlet."

   The servlet has to be made available bypassing security, therefore, the servlet mapping is required in web.xml (under WEB-INF folder).
3. Create a schedule for the report, choosing HTTP Notification.

## 10.2 Setting Up After-Report Triggers

When the report generation has completed BI Publisher will call the HTTP notification as a post-process and submit the URL (that you registered as an HTTP server) with the following additional parameters:

- jobid
- report_url
- status

  Values for status are "S" for success and "F" for failure.

Your remote application can then access these parameters using BI Publisher's APIs and Web services to access the job details, including report output and XML data as shown in the following code sample:

***Example 10–1 Sample Code for Setting Up After-Report Triggers***

```
String id = request.getParameter("jobid");
      String report_url = request.getParameter("report_url");
      String status = request.getParameter("status");

      try
      {
       Scheduler sch =new SchedulerImpl();
       JobHistoryInfo[] jobs= sch.getJobHistoryInfo(id);
       for (int i = 0; i<jobs.length; i++){
       JobHistoryInfo outinfo = jobs[i];
       FileOutputStream fos = new FileOutputStream(targetDir+id+".pdf");
       byte[] buf = new byte[256];
       int read = 0;
       InputStream in  = outinfo.getDocumentOutput();

       while ((read =in.read(buf)) > 0) {
                        fos.write(buf, 0, read);
     }
                        in.close();
                        fos.close();
       }
      } catch (Exception e) {
         Logger.log(e);
      }
```

### 10.2.1 Registering the HTTP Servlet

Note that if the HTTP servlet is running inside the BI Publisher application on the same server, you must register it in web.xml (located in the WEB-INF folder). Update the web.xml file as follows:

```
<servlet>
<servlet-name>HttpNotificationTest</servlet-name>
<servlet-class>oracle.xdo.service.scheduling.HttpNotificationTest</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>HttpNotificationTest</servlet-name>
<url-pattern>/services/HttpNotificationTest</url-pattern>
</servlet-mapping>
```

Alternatively, you can create a JSP page instead of a HTTP Servlet to handle this HTTP notification. With JSP, you do not need to modify web.xml.

### 10.2.2 Sample Program

Following is a sample HTTP servlet that is called as an HTTP Notification. In this example, the servlet is deployed on the same server as the BI Publisher application. If your servlet is deployed on a remote server, use the BI Publisher Web service APIs to access the report details. For more information about the BI Publisher Web service APIs, see *Oracle Fusion Middleware Java API Reference for Oracle Business Intelligence Publisher 11g.*

In this sample, the servlet uses the information provided by the HTTP request as input to the BI Publisher Web services to retrieve the report output. This could then be used to insert in an approval workflow.

***Example 10–2 Sample Program Code***

```
package oracle.xdo.service.scheduling;

import java.io.FileOutputStream;


import java.io.IOException;
import java.io.InputStream;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import oracle.xdo.common.log.Logger;
import oracle.xdo.server.JobHistoryInfo;
import oracle.xdo.server.Scheduler;
import oracle.xdo.server.impl.SchedulerImpl;

public class HttpNotificationTest extends HttpServlet
{
 public String targetDir =
"c://scratch/example/apphome/xmlpserver/xmlpserver/output/";
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response) throws ServletException,
IOException
  {
    doPost(request, response);
  }
 public void doPost(HttpServletRequest request,
                     HttpServletResponse response) throws ServletException,
IOException
  {


      String id=    request.getParameter("jobid");
      String report_url = request.getParameter("report_url");
      String status = request.getParameter("status");

      try
      {
       Scheduler sch =new SchedulerImpl();
       JobHistoryInfo[] jobs= sch.getJobHistoryInfo(id);
       for (int i = 0; i<jobs.length; i++){
```

```
       JobHistoryInfo outinfo = jobs[i];
       FileOutputStream fos = new
FileOutputStream(targetDir+id+"."+getFileExtension(outinfo.getDocumentDataContentT
ype()));
       byte[] buf = new byte[256];
       int read = 0;
       InputStream in  = outinfo.getDocumentOutput();

       while ((read =in.read(buf)) > 0) {
                       fos.write(buf, 0, read);
     }
                       in.close();
                       fos.close();
       }
      } catch (Exception e) {
         Logger.log(e);
      }
      }


       public static String getFileExtension(String contentType)
         {
          String ext="pdf";

          if (contentType == "application/pdf")
          ext="pdf" ;

          else if (contentType == "text/html; charset=UTF-8")
          ext="html";
          return ext;
         }
```

11

# Adding Extensions to the Layout Editor

This chapter describes how to extend the layout editor functionality using custom plug-ins to integrate content generated by other applications into BI Publisher reports. The custom content can be viewed in the interactive viewer.

This chapter includes the following sections:

- Section 11.1, "Using Layout Editor Plug-in Extensions"
- Section 11.2, "Implementing Plug-ins"
- Section 11.3, "Coding the Custom Plug-in"
- Section 11.4, "Property Support"
- Section 11.5, "Samples"
- Section 11.6, "Creating a Data Field Plug-in"

## 11.1 Using Layout Editor Plug-in Extensions

BI Publisher supports adding JavaScript plug-in extensions to the layout editor to add custom components to your reports. The custom components are included in your report when you view the report in interactive mode. During design time, when you add the plug-in to the layout editor, the icon that you define for it displays in the layout editor's **Insert** menu. You can then drag and drop the custom component to the report layout as you do any other component in the layout editor.

Use this functionality to insert static components to your reports, such as text, images, and video or data-driven components.

This feature also supports defining custom properties for your plug-in to enable report designers to add properties specific to the component plug-in you create.

## 11.2 Implementing Plug-ins

To implement a plug-in:

1. Code the JavaScript plug-in using the guidelines described in Section 11.3, "Coding the Custom Plug-in."
2. Place the JavaScript (.js) file in the following location `<BI Publisher repository>\Admin\Plugins`.
3. Reload the Layout Editor. Your plug-in icon appears in the layout editor **Insert** menu.

## 11.3 Coding the Custom Plug-in

This section provides the specification of the plug-in structure and describes the APIs provided for use with the plug-in. It contains the following topics:

- Section 11.3.1, "Plug-in Structure"
- Section 11.3.2, "JavaScript APIs That Can Be Used in Custom Plug-ins"

### 11.3.1 Plug-in Structure

The plug-in module file is a simple JavaScript file. Call a single function with the plug-in definition object to enable the plug-in.

The plug-in definition JavaScript object has the following fields:

**id**
The `id` is an identification string. Oracle recommends using the reverse domain name to avoid any naming conflicts, for example: "com.example.helloworld".

**component**
The following fields comprise the `component` object:

**name**
The name of the component. Example: "Hello World"

**icon**
The icon is the image that displays in the layout editor Insert menu to represent the plug-in. This field takes a URL that points to the icon image. Example: "http://www.example.com/img/smile.gif"

**tooltip**
The tooltip message to display for the icon image. Example: "Hello World Plugin".

**events**
(Optional) Array of the click event definition object.

**id**
(string) The event identification, for example: "filter" or "showselection".

**source**
A true|false flag, when set to true, the click action against the component triggers the filtering of the other components.

**target**
A true|false flag, when set to true, this plugin component receives the click event.

**cssClass**
(Optional) Component CSS class selector to identify the plug-in components.

**render**
The render function renders the plug-in contents. The render function passes the following parameters:

**context**
Object which contains following information:

**id**
The `id` is an assigned instantiated component ID string. The system assures this ID is unique in the same template. Oracle recommends using this ID as a prefix or suffix to the HTML element that the plug-in code generates. This practice prevents ID conflicts.

**reportLocale**
The locale assigned to the template.

**containerElem**
The container HTML element. The contents must be set to this element

**rows**
The data rows array, each item contains another array for the columns.

**fields**
The assigned xml path.

**props**
Current properties. See Section 11.4, "Property Support."

## 11.3.2 JavaScript APIs That Can Be Used in Custom Plug-ins

The following JavaScript APIs are available to use in custom plug-ins:

- handleClickEvent Method

  Captures clicked (selected) field information to send to the system.

- getPixelValue Method

  Returns the pixel value from the length string value. The system uses 96 dots per inch (dpi), which is the same as most browsers.

### 11.3.2.1 handleClickEvent Method

This method captures the clicked (or selected) field information to send to the system.

**Signature**
xdo.api.handleClickEvent(info)

This method takes the following parameter:

**info**
Clicked field information object.

The structure of the object is:

```
Object Structure
{
  id: [component id],
  [
    {
      field: [xpath to the element],
      value: [filter value]
    },
    {
      field: [xpath to the element],
      value: [filter value]
    },
  ]
```

#### 11.3.2.2 getPixelValue Method

This method returns the pixel value from the length string value. The system uses 96 dpi, which is the same as most browsers.

**Signature**

```
xdo.api.getPixelValue(lengthString)
```

The method takes one parameter:

**lengthString**
A string value that specifies the length. Supported units are "px", "pt", "in", and "cm".

## 11.4 Property Support

To add custom properties, a properties field is available. Array of property definition object can be set to this field. Construct the property definition object from the following values.

**Key**
A string value that specifies the property key. This value must be unique.

**label**
A string value that specifies the label displayed for this property in the layout editor's Properties pane.

**type**
A string value that specifies the property type. The layout editor uses this value to open the appropriate editor to edit the property. The following values are supported for type:

- string - creates a text entry box to enter string data.
- number - creates a text entry box to enter numeric data.
- bool -  creates a True/False (boolean) choice option.
- length - creates text entry box to enter length data and select units in px, in, cm, or pt.
- color - displays a color-chooser for color selection.
- font - displays the list of supported fonts for selection.
- fontsize - displays the font size selector.
- lov - creates a list of values. See options for creating the name-value pairs.

**value**
The initial value of the property. The value must follow the format of the type specified.

**options**
This parameter is valid only when the property **type** is "lov". The options parameter contains label-value pairs to define the list of values.

- **label** - the label for each list item
- **value** - the value for the label

### 11.4.1 Predefined Properties

The layout editor sets the following property settings by default:

- **width**: 400px
- **height**: 200px
- **padding**: 0px 0px 0px 0px
- **margin**: 0px 0px 0px 0px
- **border-top**: 0px none #000000
- **border-left**: 0px none #000000
- **border-right**: 0px none #000000
- **border-bottom**: 0px none #000000

## 11.5 Samples

Following are plug-in examples:

- Section 11.5.1, "Example of Static Plug-in: Company Logo"
- Section 11.5.2, "Example Plug-in to Insert YouTube Video"

### 11.5.1 Example of Static Plug-in: Company Logo

This example demonstrates how to implement the simplest type of plug-in. This plug-in adds an icon to the layout editor toolbar that when selected will insert a company logo into the layout at the insertion point. The JavasSript file for this plug-in defines everything in a single object.

1. Write the JavaScript for the plug-in.

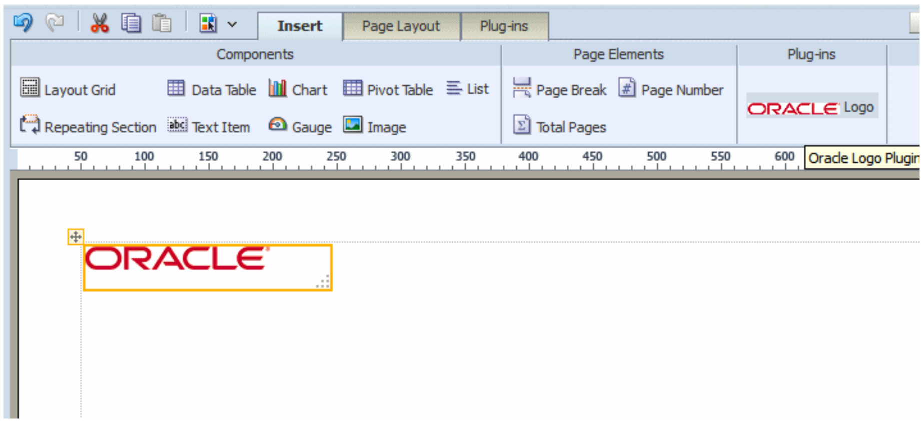
   The plug-in definition for this example is:

```
{
  id: "com.oracle.xdo.logo",
  component:
  {
    name: "Logo",
    icon: "http://localhost:7001/xmlpserver/logo_icon.gif",
    tooltip: "Oracle Logo Plugin"
  },
  render: function(context, containerElem, rows, fields, props)
  {
    containerElem.innerHTML = '<img
src="http://localhost:7001/xmlpserver/oracle.gif" />';
  }
}
```

2. Save the file with the ".js" extension and place it under the `<REPOSITORY ROOT>/Admin/Plugins` directory.
3. Reload the layout editor application. The logo plug-in displays in the ribbon.

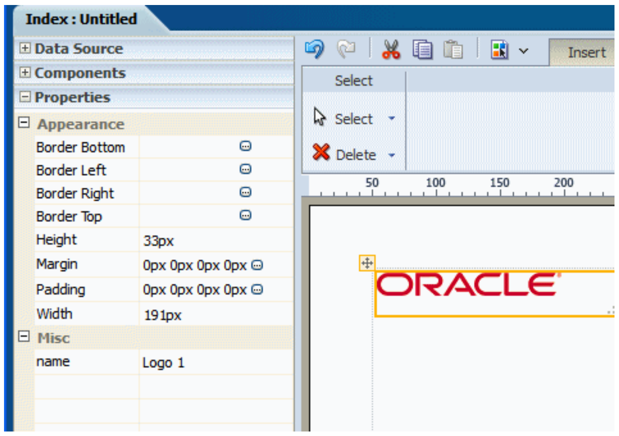
   Click the "Oracle" icon or drag and drop the item to insert the logo into your layout. This is shown in Figure 11–1.

***Figure 11–1 Logo Plug-in Shown in the Layout Editor***



With the inserted plug-in component selected, click the **Properties** pane to view or edit the default properties, as shown in Figure 11–2.

***Figure 11–2 Properties for the Logo Plug-in Component***



| | |
|---|---|
| Appearance | |
| Border Bottom | |
| Border Left | |
| Border Right | |
| Border Top | |
| Height | 33px |
| Margin | 0px 0px 0px 0px |
| Padding | 0px 0px 0px 0px |
| Width | 191px |
| Misc | |
| name | Logo 1 |

## 11.5.2 Example Plug-in to Insert YouTube Video

This example creates a plug-in to enable users to embed YouTube videos in a BI Publisher report. When viewed as interactive output, the user can view the YouTube video using the embedded controls.

### 1. Define the id and component functions of the plug-in.

YouTube provides simple HTML code in each of its video pages to facilitate third-party embedding. This code is revealed when you click the "Share" button and then the "Embed" button shown on the YouTube video page. An example of this code is:

```
<iframe width="425" height="349" src="http://www.youtube.com/embed/TK7KYaCEGZU"
frameborder="0" allowfullscreen></iframe>
```

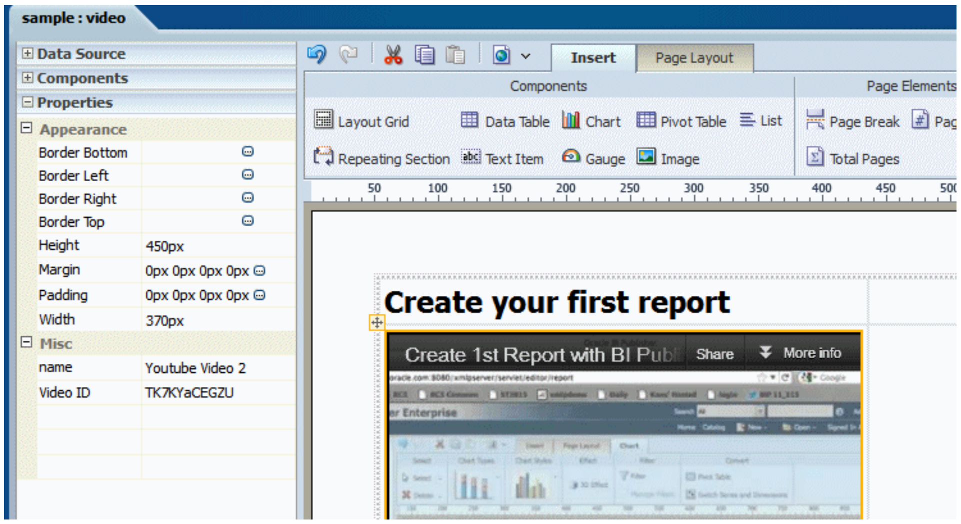
When embedding the video, the iframe requires a width, height, and video identifier (called in this example "videoid"). Define these as properties in your plug-in. Although width and height already have default values, you can define them here to override the defaults. Because the base URL for YouTube videos is constant, defining the videoid as a property should produce the expected results. To give report designers more control, you can add additional properties such as "allowfullscreen" as a boolean property or "privacy-enhanced" boolean property.

Following is the sample code for the property definitions:

```
{
   id: "com.oracle.xdo.youtube",
   component: {
     name: "Youtube Video",
     cssClass: "youtube",
     icon: "http://hostname.com/youtube_32x32.png",
     tooltip: "Insert Youtube Video"
   },
   properties: [
                 {key: "width", label:"Width", type:"length", value:"450px"},
                 {key: "height", label:"Height", type:"length", value:"370px"},
                 {key: "videoid", label:"Video ID", type:"string",
value:"TK7KYaCEGZU"}
                ]
   .......
}
```

Figure 11–3 shows how the properties appear in the Properties pane of the layout editor.

***Figure 11–3 Properties Display in the Layout Editor for the YouTube Plug-in***



| | |
|---|---|
| Appearance | |
| Border Bottom | |
| Border Left | |
| Border Right | |
| Border Top | |
| Height | 450px |
| Margin | 0px 0px 0px 0px |
| Padding | 0px 0px 0px 0px |
| Width | 370px |
| Misc | |
| name | Youtube Video 2 |
| Video ID | TK7KYaCEGZU |

## 2. Define the render function

In the render function, retrieve the property values using the property key; call the `getPixelValue` system function to get the pixel value for width and height; and create html code for embedding the YouTube video.

YouTube videos use swf (Flash) for playback. Embedded Flash video players interrupt the click event required to play or pause the video. YouTube provides the workaround for this. Add `&wmode=transparent` after the embedding `src` attribute.

The code for the render function is as follows:

```
render: function(context, containerElem, rows, fields, props) {
    var videoid = props["videoid"];
    var width = props["width"];
```

```
     var height = props["height"];

     var widthPx = xdo.api.getPixelValue(width);
     var heightPx = xdo.api.getPixelValue(height);

     var iframe = '<iframe id="'+context.id+'_iframe" style="z-index: -1"
src="http://www.youtube.com/embed/'+videoid+'?wmode=transparent"
width="'+(widthPx)+'" height="'+(heightPx)+'"> </iframe>';
     xdo.dom.DOMElement.set(containerElem, iframe);
   }
```

If you are embedding a flash component from another source that does not provide a similar workaround, then you must implement one yourself. You can either modify the flash component to propagate a click event to HTML or create extra "clickable" space around the flash component so that the user can click the space to select the component.

The complete code sample for the YouTube plug-in is:

```
{
   id: "com.oracle.xdo.youtube",
   component: {
     name: "Youtube Video",
     cssClass: "youtube",
     icon: "http://hostname.com/youtube_32x32.png",
     tooltip: "Insert Youtube Video"
   },
   properties: [
                {key: "width", label:"Width", type:"length", value:"450px"},
                {key: "height", label:"Height", type:"length", value:"370px"},
                {key: "videoid", label:"Video ID", type:"string",
value:"TK7KYaCEGZU"}
               ],
   render: function(context, containerElem, rows, fields, props) {
     var videoid = props["videoid"];

     var width = props["width"];
     var height = props["height"];
     var widthPx = xdo.api.getPixelValue(width);
     var heightPx = xdo.api.getPixelValue(height);

     var iframe = '<iframe id="'+context.id+'_iframe" style="z-index: -1"
src="http://www.youtube.com/embed/'+videoid+'?wmode=transparent"
width="'+(widthPx)+'" height="'+(heightPx)+'"> </iframe>';
     xdo.dom.DOMElement.set(containerElem, iframe);
   }
}
```

## 11.6 Creating a Data Field Plug-in

You can define a plug-in to have a data field that is defined in the data model. To do this, specify field information in the **fields** component of the plug-in structure. Specify the label and the measure.

```
{
  id: "com.oracle.xdo...",
  component: {
    name: "Field Test"
  }
  fields:
  [
    {name: "labelField", caption: "Drop Label Field Here", fieldType:"label",
dataType:"string"},
    {name: "dataField", caption: "Drop Data Field Here", fieldType:"measure",
dataType: "number", formula: "summation"}
  ],
}
```

For each field define the following:

- **name**
- **caption** - text that the layout editor displays to the user for the field. For example: "Drop Label Field Here".
- **fieldType** - valid values for fieldType are "label" and "measure". The layout editor displays the caption (such as "Drop Label Field Here") defined for the field. The layout editor user can drag and drop the data field from the data source tree structure to the plug-in component.
- **dataType** - the following data types are supported:
  - string (text string, default)
  - number (number, including integer and float)
  - data (XML date format)

  The data type of the element that you drag and drop from the data model structure in the layout editor must match the dataType defined here.
- **formula** - when the fieldType is "measure" supports aggregating values by specifying the function name. The following function names are supported:
  - count
  - count-distinct
  - summation
  - average
  - maximum
  - minimum

### Accessing Data

At runtime, calculated data is stored in the `rows` variable of the `render` function. The `rows` variable is an array type and each `rows` element has another array for keeping column information. The following render function implementation displays data in HTML:

```
render: function(context, containerElem, rows, fields, props) {
  // setup column
  var html = '<table>';
  for (var i=0, rowCount=rows.length; i<rowCount; i++)
  {
    html += "<tr>";
    var col = rows[i];
    for (var j=0, colCount=col.length; j<colCount; j++)
    {
      html += "<td>";
      html += col[j];
      html += "</td>";
    }
    html += "</tr>";
```

```
   }
  html += '</table>';
  containerElem.innerHTML = html;
 }
```

### Field label

Users can define a field label string by using the layout editor's property editor. This information can then be accessed by the `fields` variable in the `render` function arguments, as well as information defined in the plug-in definition.

- field - the field path
- fieldFormula - the field formula; must be null if this field is fieldType="label"
- fieldType - the field type: "label" or "measure"
- dataType - the data type: "string", "number", or "data"
- label - the user-specified label string

# Index