Oracle® Fusion Middleware
Developer’s Guide for Oracle Business Intelligence Publisher
(Oracle Fusion Applications Edition)
11g Release 1 (11.1.1)
E26385-03

June 2013
Explains how to incorporate Oracle Business Intelligence Publisher functionality into custom applications using the Java and Web services application programming interfaces.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audience</td>
<td>xiii</td>
</tr>
<tr>
<td>Documentation Accessibility</td>
<td>xiii</td>
</tr>
<tr>
<td>Related Documentation and Other Resources</td>
<td>xiv</td>
</tr>
<tr>
<td>Related Documentation and Other Resources</td>
<td>xiv</td>
</tr>
<tr>
<td>Conventions</td>
<td>xiv</td>
</tr>
<tr>
<td><strong>Part I Oracle BI Publisher Web Services</strong></td>
<td></td>
</tr>
<tr>
<td>1 Introduction to the BI Publisher Web Services</td>
<td></td>
</tr>
<tr>
<td>1.1 About BI Publisher Web Services</td>
<td>1-1</td>
</tr>
<tr>
<td>1.2 Accessing the WSDLs</td>
<td>1-2</td>
</tr>
<tr>
<td>1.3 About In-Session Methods</td>
<td>1-2</td>
</tr>
<tr>
<td>1.4 Debugging Web Service Applications</td>
<td>1-2</td>
</tr>
<tr>
<td>2 Data Types in Oracle BI Publisher Web Services</td>
<td></td>
</tr>
<tr>
<td>2.1 Base Data Types</td>
<td>2-1</td>
</tr>
<tr>
<td>2.2 XML-to-Java Data Type Mappings</td>
<td>2-1</td>
</tr>
<tr>
<td>2.3 Complex Data Types</td>
<td>2-2</td>
</tr>
<tr>
<td>2.3.1 ArrayOf_xsd_string</td>
<td>2-3</td>
</tr>
<tr>
<td>2.3.2 ArrayOfEMailDeliveryOption</td>
<td>2-3</td>
</tr>
<tr>
<td>2.3.3 ArrayOfFaxDeliveryOption</td>
<td>2-3</td>
</tr>
<tr>
<td>2.3.4 ArrayOfFTPDeliveryOption</td>
<td>2-4</td>
</tr>
<tr>
<td>2.3.5 ArrayOfItemData</td>
<td>2-4</td>
</tr>
<tr>
<td>2.3.6 ArrayOfJobInfo</td>
<td>2-4</td>
</tr>
<tr>
<td>2.3.7 ArrayOfJobOutput</td>
<td>2-4</td>
</tr>
<tr>
<td>2.3.8 ArrayOfJobOutputDelivery</td>
<td>2-4</td>
</tr>
<tr>
<td>2.3.9 ArrayOfLocalDeliveryOption</td>
<td>2-4</td>
</tr>
<tr>
<td>2.3.10 ArrayOfMetaData</td>
<td>2-5</td>
</tr>
<tr>
<td>2.3.11 ArrayOfParamNameValue</td>
<td>2-5</td>
</tr>
<tr>
<td>2.3.12 ArrayOfPrintDeliveryOption</td>
<td>2-5</td>
</tr>
<tr>
<td>2.3.13 ArrayOfString</td>
<td>2-5</td>
</tr>
<tr>
<td>2.3.14 ArrayOfTemplateFormatLabelValue</td>
<td>2-5</td>
</tr>
<tr>
<td>2.3.15 ArrayOfTemplateFormatLabelValues</td>
<td>2-6</td>
</tr>
<tr>
<td>2.3.16 ArrayOfWebDAVDeliveryOption</td>
<td>2-6</td>
</tr>
<tr>
<td>2.3.17 BIPDataSource</td>
<td>2-6</td>
</tr>
</tbody>
</table>
3 ScheduleService

3.1 cancelSchedule() Method .................................................................................. 3-2
3.2 cancelScheduleInSession() Method .................................................................... 3-2
3.3 deleteJobHistory() Method .................................................................................. 3-3
3.4 deleteJobHistoryInSession() Method .................................................................... 3-3
3.5 deleteSchedule() Method ..................................................................................... 3-3
3.6 deleteScheduleInSession() Method ...................................................................... 3-4
3.7 deliveryService() Method .................................................................................... 3-4
3.8 deliveryServiceInSession() Method ...................................................................... 3-4
3.9 downloadDocumentData() Method ....................................................................... 3-5
3.10 downloadDocumentDataInSession() Method ....................................................... 3-5
3.11 downloadXMLData() Method ............................................................................... 3-6
4 ReportService

4.1 createReport() Method ................................................................. 4-2
4.2 createReportInSession() Method .................................................. 4-3
4.3 downloadReportDataChunk() Method .......................................... 4-4
4.4 downloadReportDataChunkInSession() Method ......................... 4-4
4.5 getReportDefinition() Method .................................................... 4-5
4.6 getReportDefinitionInSession() Method ..................................... 4-5
4.7 getReportParameters() Method .................................................. 4-6
4.8 getReportParametersInSession() Method ................................. 4-6
4.9 getReportSampleData() Method ................................................ 4-7
4.10 getReportSampleDataInSession() Method ............................... 4-7
4.11 getTemplate() Method ............................................................. 4-7
4.12 getTemplateInSession() Method ................................................ 4-8
4.13 getTemplateParameters() Method ............................................. 4-8
4.14 getTemplateParameterInSession() Method ............................. 4-9
4.15 getXDOSchema() Method .......................................................... 4-9
4.16 getXDOSchemaInSession() Method ............................................ 4-10
4.17 removeTemplateForReport() Method ..................................... 4-10
4.18 removeTemplateForReportInSession() Method ..................... 4-11
5 SecurityService

5.1 assignRolesToUser() Method ................................................................. 5-2
5.2 createRole() Method .............................................................................. 5-2
5.3 createUser() Method ............................................................................. 5-2
5.4 deleteRole() Method ............................................................................. 5-3
5.5 deleteUser() Method ........................................................................... 5-3
5.6 getBIPHTTPSessionInterval() Method ................................................ 5-3
5.7 getObjectSecurityXML() Method ......................................................... 5-4
5.8 getSecurityModel() Method .................................................................. 5-4
5.9 hasObjectAccess() Method ................................................................. 5-4
5.10 hasObjectAccessInSession() Method ................................................ 5-4
5.11 impersonate() Method ....................................................................... 5-4
5.12 isUserExists() Method ....................................................................... 5-5
5.13 login() Method .................................................................................. 5-6
5.14 logout() Method ................................................................................ 5-6
5.15 notifyBIEEPreferencesUpdated() Method ....................................... 5-6
5.16 notifyBIEEPreferencesUpdatedWithString() Method ..................... 5-6
5.17 removeRolesFromUser() Method ......................................................... 5-7
5.18 updateRole() Method ....................................................................... 5-7
5.19 updateUser() Method ....................................................................... 5-7
5.20 validateLogin() Method .................................................................... 5-8

6 CatalogService

6.1 copyObject() Method .......................................................................... 6-2
6.2 copyObjectInSession() Method ............................................................. 6-2
6.3 createFolder() Method ........................................................................ 6-3
6.4 createFolderInSession() Method .......................................................... 6-3
6.5 createObject() Method ....................................................................... 6-3
6.6 createObjectInSession() Method .......................................................... 6-4
6.7 deleteObject() Method ...................................................................... 6-5
6.8 deleteObjectInSession() Method ........................................................ 6-5
6.9 downloadObject() Method .................................................................. 6-5
Part II Oracle BI Publisher Java APIs

7 Using the BI Publisher Java APIs

7.1 BI Publisher Core APIs ......................................................................................... 7-1
7.2 Prerequisites ........................................................................................................... 7-2
7.3 Obtaining the Libraries .......................................................................................... 7-3
7.4 PDF Form Processing Engine .................................................................................. 7-4
  7.4.1 Overview of the PDF Form Processing Engine .................................................... 7-4
  7.4.2 Merging a PDF Template with XML Data ........................................................... 7-4
  7.4.3 Merging XML Data with a PDF Template Using Input/Output File Name .......... 7-4
  7.4.4 Merging XML Data with a PDF Template Using Input/Output Streams .......... 7-5
  7.4.5 Merging an XML Data String with a PDF Template ........................................ 7-5
  7.4.6 Retrieving a List of Field Names ....................................................................... 7-6
  7.4.7 Generating XFDF Data ...................................................................................... 7-6
  7.4.8 Converting XML Data into XFDF Format Using XSLT ...................................... 7-7
7.5 RTF Processor Engine ............................................................................................ 7-8
  7.5.1 Pairing with XLIFF File ...................................................................................... 7-9
  7.5.2 Generating XSL .................................................................................................. 7-9
    7.5.2.1 Generating XSL with Input/Output File Names .............................................. 7-9
    7.5.2.2 Generating XSL with Input/Output Stream .................................................... 7-10
7.6 FO Processor Engine ............................................................................................ 7-10
  7.6.1 Major Features of the FO Processor ................................................................. 7-10
    7.6.1.1 Bidirectional Text ....................................................................................... 7-10
    7.6.1.2 Font Fallback Mechanism .......................................................................... 7-11
    7.6.1.3 Variable Header and Footer ....................................................................... 7-11
    7.6.1.4 Horizontal Table Break .............................................................................. 7-11
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.6.2</td>
<td>Generating Output from an XML File and an XSL File</td>
<td>7-12</td>
</tr>
<tr>
<td>7.6.3</td>
<td>Generating Output Using File Names</td>
<td>7-13</td>
</tr>
<tr>
<td>7.6.4</td>
<td>Generating Output Using Streams</td>
<td>7-14</td>
</tr>
<tr>
<td>7.6.5</td>
<td>Generating Output from an Array of XSL Templates and XML Data</td>
<td>7-14</td>
</tr>
<tr>
<td>7.6.6</td>
<td>Using the XSL-FO Utility</td>
<td>7-15</td>
</tr>
<tr>
<td>7.6.6.1</td>
<td>Creating XSL-FO from an XML File and an XSL File</td>
<td>7-15</td>
</tr>
<tr>
<td>7.6.6.2</td>
<td>Creating XSL-FO from Two XML Files and Two XSL files</td>
<td>7-16</td>
</tr>
<tr>
<td>7.6.6.3</td>
<td>Merging Two XSL-FO Files</td>
<td>7-17</td>
</tr>
<tr>
<td>7.6.6.4</td>
<td>Generating Output from an FO File</td>
<td>7-17</td>
</tr>
<tr>
<td>7.6.6.5</td>
<td>Generating Output Using File Names</td>
<td>7-17</td>
</tr>
<tr>
<td>7.6.6.6</td>
<td>Generating Output Using Streams</td>
<td>7-18</td>
</tr>
<tr>
<td>7.6.6.7</td>
<td>Generating Output with an Array of FO Data</td>
<td>7-19</td>
</tr>
<tr>
<td>7.7</td>
<td>PDF Document Merger</td>
<td>7-19</td>
</tr>
<tr>
<td>7.7.1</td>
<td>Merging PDF Documents</td>
<td>7-20</td>
</tr>
<tr>
<td>7.7.1.1</td>
<td>Merging PDF Documents with Input/Output File Names</td>
<td>7-20</td>
</tr>
<tr>
<td>7.7.1.2</td>
<td>Merging PDF Documents with Input/Output Streams</td>
<td>7-21</td>
</tr>
<tr>
<td>7.7.1.3</td>
<td>Merging with Background to Place Page Numbering</td>
<td>7-21</td>
</tr>
<tr>
<td>7.7.1.4</td>
<td>Adding Page Numbers to Merged Documents</td>
<td>7-22</td>
</tr>
<tr>
<td>7.7.2</td>
<td>Setting a Text or Image Watermark</td>
<td>7-23</td>
</tr>
<tr>
<td>7.7.2.1</td>
<td>Setting a Text Watermark</td>
<td>7-23</td>
</tr>
<tr>
<td>7.7.2.2</td>
<td>Setting Image Watermark</td>
<td>7-25</td>
</tr>
<tr>
<td>7.8</td>
<td>PDF Bookbinder Processor</td>
<td>7-26</td>
</tr>
<tr>
<td>7.8.1</td>
<td>Usage</td>
<td>7-26</td>
</tr>
<tr>
<td>7.8.2</td>
<td>XML Control File</td>
<td>7-26</td>
</tr>
<tr>
<td>7.8.3</td>
<td>Command Line Options</td>
<td>7-27</td>
</tr>
<tr>
<td>7.8.4</td>
<td>API Method Call</td>
<td>7-28</td>
</tr>
<tr>
<td>7.9</td>
<td>PDF Digital Signature Engine</td>
<td>7-28</td>
</tr>
<tr>
<td>7.9.1</td>
<td>Overview of the PDF Digital Signature Engine</td>
<td>7-28</td>
</tr>
<tr>
<td>7.9.2</td>
<td>Signing PDF Documents</td>
<td>7-28</td>
</tr>
<tr>
<td>7.9.3</td>
<td>Delivering Signed PDF Documents</td>
<td>7-29</td>
</tr>
<tr>
<td>7.9.4</td>
<td>Verifying Signed PDF Documents</td>
<td>7-29</td>
</tr>
<tr>
<td>7.10</td>
<td>eText Processor</td>
<td>7-30</td>
</tr>
<tr>
<td>7.10.1</td>
<td>Converting RTF eText Templates to XSL</td>
<td>7-30</td>
</tr>
<tr>
<td>7.10.2</td>
<td>Producing Text Output for EDI and EFT Transmissions</td>
<td>7-30</td>
</tr>
<tr>
<td>7.10.3</td>
<td>Getting Sequence Numbers</td>
<td>7-31</td>
</tr>
<tr>
<td>7.11</td>
<td>Document Processor Engine</td>
<td>7-32</td>
</tr>
<tr>
<td>7.11.1</td>
<td>Hierarchy and Elements of the Document Processor XML File</td>
<td>7-32</td>
</tr>
<tr>
<td>7.11.2</td>
<td>XML File Samples</td>
<td>7-35</td>
</tr>
<tr>
<td>7.11.2.1</td>
<td>Defining two data sets</td>
<td>7-35</td>
</tr>
<tr>
<td>7.11.2.2</td>
<td>Defining multiple templates and data</td>
<td>7-36</td>
</tr>
<tr>
<td>7.11.2.3</td>
<td>Retrieving templates over HTTP</td>
<td>7-36</td>
</tr>
<tr>
<td>7.11.2.4</td>
<td>Retrieving data over HTTP</td>
<td>7-37</td>
</tr>
<tr>
<td>7.11.2.5</td>
<td>Generating more than one output</td>
<td>7-37</td>
</tr>
<tr>
<td>7.11.2.6</td>
<td>Defining page numbers</td>
<td>7-38</td>
</tr>
<tr>
<td>7.11.3</td>
<td>Invoke Processors</td>
<td>7-39</td>
</tr>
<tr>
<td>7.11.3.1</td>
<td>Invoking Processors with Input File Name</td>
<td>7-39</td>
</tr>
<tr>
<td>7.11.3.2</td>
<td>Invoking Processors with InputStream</td>
<td>7-40</td>
</tr>
</tbody>
</table>
8 Using the Delivery Manager Java APIs

8.1 Using the Delivery Manager .............................................................................. 8-1
8.2 Delivering Documents by E-Mail ......................................................................... 8-2
8.2.1 Defining Multiple Recipients ................................................................. 8-4
8.2.2 Attaching Multiple Documents to One Request ................................... 8-4
8.2.3 Attaching HTML Documents ................................................................. 8-5
8.2.4 Displaying Attachments at the Top of E-Mail ....................................... 8-5
8.2.5 Using a String Object as the E-Mail Body ............................................ 8-6
8.2.6 Using an HTML Document as the E-Mail Body .................................... 8-7
8.2.7 Providing User Name and Password for Authentication .................... 8-7
8.3 Delivering Your Document to a Printer ...................................................... 8-8
8.3.1 Printing over an HTTP Proxy Server .................................................... 8-11
8.4 Delivering Your Document to a Local Printer ............................................ 8-11
8.5 Delivering Your Documents to a Fax Server .............................................. 8-13
8.6 Delivering Your Documents to a RightFax Server ..................................... 8-13
8.7 Delivering Your Documents to a WebDAV Server ...................................... 8-16
8.8 Delivering Your Document over the File Transfer Protocol (FTP) ............ 8-18
8.9 Delivering Your Documents over Secure FTP .............................................. 8-19
8.9.1 Authentication Modes ............................................................................. 8-20
8.10 Delivering Your Documents over Hypertext Transfer Protocol (HTTP) .... 8-21
8.11 Delivering Documents over A52 ................................................................. 8-23
8.11.1 Delivery Receipt .................................................................................... 8-26
8.11.2 Synchrony ........................................................................................... 8-26
8.11.3 Document Signing ............................................................................... 8-27
8.11.4 Document Encryption ......................................................................... 8-27
8.12 Delivering Documents Using an External Command .............................. 8-28
8.13 Delivering Documents to the Local File System ....................................... 8-29
8.14 Direct and Buffering Modes ........................................................................... 8-29
8.14.1 Direct Mode ........................................................................................ 8-29
8.14.2 Buffering Mode .................................................................................... 8-30
8.15 Asynchronous Delivery Requests ............................................................... 8-30
8.16 Document Filter Support ............................................................................. 8-31
8.16.1 PDF-to-PostScript Conversion Filter ................................................. 8-32
8.17 Date Expression Support ............................................................................. 8-32
8.18 Internationalization Support ........................................................................ 8-33
8.18.1 SMTP .................................................................................................. 8-33
8.18.2 IPP ...................................................................................................... 8-33
8.18.3 WebDAV ............................................................................................ 8-33
8.18.4 FTP .................................................................................................... 8-33
8.18.5 HTTP ................................................................................................ 8-34
# Part III Other Topics

9 Making a View Object Available to BI Publisher as a Data Source

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1</td>
<td>Prerequisites</td>
</tr>
<tr>
<td>9.2</td>
<td>Configuring the Application Module</td>
</tr>
<tr>
<td>9.3</td>
<td>Updating web.xml</td>
</tr>
<tr>
<td>9.4</td>
<td>Updating weblogic.xml</td>
</tr>
<tr>
<td>9.5</td>
<td>Deploying the Application Module</td>
</tr>
<tr>
<td>9.6</td>
<td>Updating the providers.xml File</td>
</tr>
</tbody>
</table>

10 Using BI Publisher with Oracle JDeveloper

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.1</td>
<td>Including Oracle BI Publisher Reports in ADF Applications</td>
</tr>
<tr>
<td>10.2</td>
<td>Installing the BI Publisher Extension</td>
</tr>
<tr>
<td>10.3</td>
<td>Adding BI Publisher Content to an ADF Project</td>
</tr>
<tr>
<td>10.3.1</td>
<td>Adding the BI Publisher Technology Scope</td>
</tr>
<tr>
<td>10.3.2</td>
<td>Adding a BI Publisher Region to the JSF Page</td>
</tr>
<tr>
<td>10.3.3</td>
<td>Configuring Connection to the BI Publisher Server</td>
</tr>
<tr>
<td>10.3.4</td>
<td>Setting Properties for the BI Publisher Region</td>
</tr>
<tr>
<td>10.4</td>
<td>Configuring a Security Policy</td>
</tr>
<tr>
<td>10.4.1</td>
<td>Viewing and Configuring the Server Policy</td>
</tr>
<tr>
<td>10.4.2</td>
<td>Configuring the Client-Side Service Policy</td>
</tr>
<tr>
<td>10.5</td>
<td>Deploying and Running the Application</td>
</tr>
<tr>
<td>10.6</td>
<td>Passing Parameters from the Application Page</td>
</tr>
<tr>
<td>10.7</td>
<td>Using a Push Data Model</td>
</tr>
<tr>
<td>10.8</td>
<td>Conditionally Required Settings</td>
</tr>
<tr>
<td>10.8.1</td>
<td>Setting the MIME Types for Your Report Output Type</td>
</tr>
</tbody>
</table>

11 Setting Up After-Report Triggers

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.1</td>
<td>Overview of After-Report Triggers</td>
</tr>
<tr>
<td>11.1.1</td>
<td>Limitations</td>
</tr>
<tr>
<td>11.1.2</td>
<td>Process Overview for Adding After-Report Triggers to Reports</td>
</tr>
<tr>
<td>11.2</td>
<td>Setting Up After-Report Triggers</td>
</tr>
<tr>
<td>11.2.1</td>
<td>Registering the HTTP Servlet</td>
</tr>
<tr>
<td>11.2.2</td>
<td>Sample Program</td>
</tr>
</tbody>
</table>
12 Adding Extensions to the Layout Editor

12.1 Using Layout Editor Plug-in Extensions ................................................................. 12-1
12.2 Implementing Plug-ins ......................................................................................... 12-1
12.3 Coding the Custom Plug-in .................................................................................. 12-2
12.3.1 Plug-in Structure .............................................................................................. 12-2
12.3.2 JavaScript APIs That Can Be Used in Custom Plug-ins .................................... 12-3
12.3.2.1 handleClickEvent Method .......................................................................... 12-3
12.3.2.2 getPixelValue Method ................................................................................ 12-4
12.4 Property Support ................................................................................................. 12-5
12.4.1 Predefined Properties ..................................................................................... 12-5
12.5 Samples ............................................................................................................... 12-5
12.5.1 Example of Static Plug-in: Company Logo ..................................................... 12-5
12.5.2 Example Plug-in to Insert YouTube Video ..................................................... 12-6
12.6 Creating a Data Field Plug-in .............................................................................. 12-8

Index
Preface

Oracle Business Intelligence Publisher (BI Publisher) is a comprehensive set of enterprise business intelligence 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.

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:


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


Conventions

The following text conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td>monospace</td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
This part explains how to use the Oracle BI Publisher Web services. It includes the following chapters:

- Chapter 1, "Introduction to the BI Publisher Web Services"
- Chapter 2, "Data Types in Oracle BI Publisher Web Services"
- Chapter 3, "ScheduleService"
- Chapter 4, "ReportService"
- Chapter 5, "SecurityService"
- Chapter 6, "CatalogService"
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
- SecurityService - v2/SecurityService
- 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:
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.com.

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

## 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>
<thead>
<tr>
<th>Base Type</th>
<th>Oracle JDeveloper</th>
</tr>
</thead>
<tbody>
<tr>
<td>xsd:boolean</td>
<td>java.lang.Boolean</td>
</tr>
<tr>
<td>xsd:dateTime</td>
<td>java.util.Date</td>
</tr>
<tr>
<td>xsd:int</td>
<td>java.lang.Integer</td>
</tr>
<tr>
<td>xsd:string</td>
<td>java.lang.String</td>
</tr>
</tbody>
</table>
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`
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>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String[] item</td>
<td>An array of strings.</td>
</tr>
</tbody>
</table>

2.3.2 ArrayOfEMailDeliveryOption

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMailDeliveryOption[] item</td>
<td>See Section 2.3.23, &quot;EMailDeliveryOption.&quot;</td>
</tr>
</tbody>
</table>

2.3.3 ArrayOfFaxDeliveryOption

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

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTPDeliveryOption[] item</td>
<td>See Section 2.3.26, &quot;FTPDeliveryOption.&quot;</td>
</tr>
</tbody>
</table>

2.3.5 **ArrayOfItemData**

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemData[] item</td>
<td>See Section 2.3.27, &quot;ItemData.&quot;</td>
</tr>
</tbody>
</table>

2.3.6 **ArrayOfJobInfo**

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JobInfo[] item</td>
<td>See Section 2.3.31, &quot;JobInfo.&quot;</td>
</tr>
</tbody>
</table>

2.3.7 **ArrayOfJobOutput**

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JobOutput[] item</td>
<td>See Section 2.3.33, &quot;JobOutput.&quot;</td>
</tr>
</tbody>
</table>

2.3.8 **ArrayOfJobOutputDelivery**

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JobOutputDelivery[] item</td>
<td>See Section 2.3.34, &quot;JobOutputDelivery.&quot;</td>
</tr>
</tbody>
</table>

2.3.9 **ArrayOfLocalDeliveryOption**

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

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

Table 2–12 Fields Provided by ArrayOfMetaData

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MetaData[] item</td>
<td>See Section 2.3.39, &quot;MetaData.&quot;</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParamNameValue[] item</td>
<td>See Section 2.3.41, &quot;ParamNameValue.&quot;</td>
</tr>
</tbody>
</table>

2.3.12 ArrayOfPrintDeliveryOption

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

Table 2–14 Fields Provided by ArrayOfPrintDeliveryOption

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrintDeliveryOption[] item</td>
<td>See Section 2.3.43, &quot;PrintDeliveryOption.&quot;</td>
</tr>
</tbody>
</table>

2.3.13 ArrayOfString

Use this data type to hold an array of strings.

Table 2–15 Fields Provided by ArrayOfString

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String[] item</td>
<td>An array of strings.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TemplateFormatLabelValue[] item</td>
<td>See Section 2.3.49, &quot;TemplateFormatLabelValue.&quot;</td>
</tr>
</tbody>
</table>
2.3.15 ArrayOfTemplateFormatLabelValues

Use this data type to hold an array of TemplateFormatsLabelValues objects. ArrayOfTemplateFormatLabelValues 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 ArrayOfTemplateFormatLabelValues

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TemplateFormatsLabelValues[] item</td>
<td>See Section 2.3.50, &quot;TemplateFormatsLabelValues.&quot;</td>
</tr>
</tbody>
</table>

2.3.16 ArrayOfWebDAVDeliveryOption

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

Table 2–18 Fields Provided by ArrayOfWebDAVDeliveryOption

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebDAVDeliveryOption[]</td>
<td>See Section 2.3.51, &quot;WebDAVDeliveryOption.&quot;</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDBCDataSource</td>
<td>Contains the elements to specify a JDBC data source. See Section 2.3.28, &quot;JDBCDataSource.&quot;</td>
</tr>
<tr>
<td>FileDataSource</td>
<td>Contains the elements to specify a file data source. See Section 2.3.25, &quot;FileDataSource.&quot;</td>
</tr>
</tbody>
</table>

2.3.18 CatalogContents

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

Table 2–20 Fields Provided by CatalogContents

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayOfItemData catalogContents</td>
<td>See Section 2.3.5, &quot;ArrayOfItemData.&quot;</td>
</tr>
</tbody>
</table>

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
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>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayOfEMailDeliveryOption</td>
<td>See Section 2.3.2, &quot;ArrayOfEMailDeliveryOption.&quot;</td>
</tr>
<tr>
<td>ArrayOfFaxDeliveryOption</td>
<td>See Section 2.3.3, &quot;ArrayOfFaxDeliveryOption.&quot;</td>
</tr>
<tr>
<td>ArrayOfFTPDeliveryOption</td>
<td>See Section 2.3.4, &quot;ArrayOfFTPDeliveryOption.&quot;</td>
</tr>
<tr>
<td>ArrayOfLocalDeliveryOption</td>
<td>See Section 2.3.9, &quot;ArrayOfLocalDeliveryOption.&quot;</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String contentType</td>
<td>The content type of the generated document. Possible values are: &quot;text/html;charset=UTF-8&quot; &quot;text/plain;charset=UTF-8&quot; &quot;application/pdf&quot; &quot;application/vnd.ms-powerpoint&quot; &quot;application/vnd.ms-powerpoint&quot; &quot;application/vnd.ms-excel&quot; &quot;application/msword&quot; &quot;application/x-shockwave-flash&quot; &quot;text/xml&quot; &quot;message/rfc822&quot;</td>
</tr>
<tr>
<td>DeliveryChannels</td>
<td>See Section 2.3.20, &quot;DeliveryChannels.”</td>
</tr>
<tr>
<td>byte[] documentData</td>
<td>The output document.</td>
</tr>
<tr>
<td>BIPDataSource</td>
<td>See Section 2.3.17, &quot;BIPDataSource.”</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayOf_xsd_String EMailServerNames</td>
<td>The list of e-mail server names returned in the <code>ArrayOf_xsd_string</code> data type. See Section 2.3.1, &quot;ArrayOf_xsd_string.&quot;</td>
</tr>
<tr>
<td>ArrayOf_xsd_String FTPServerNames</td>
<td>The list of FTP server names returned in the <code>ArrayOf_xsd_string</code> data type. See Section 2.3.1, &quot;ArrayOf_xsd_string.&quot;</td>
</tr>
<tr>
<td>ArrayOf_xsd_String HTTPServerNames</td>
<td>The list of HTTP server names returned in the <code>ArrayOf_xsd_string</code> data type. See Section 2.3.1, &quot;ArrayOf_xsd_string.”</td>
</tr>
<tr>
<td>ArrayOf_xsd_String SFTPServerNames</td>
<td>The list of SFTP server names returned in the <code>ArrayOf_xsd_string</code> data type. See Section 2.3.1, &quot;ArrayOf_xsd_string.”</td>
</tr>
<tr>
<td>ArrayOf_xsd_String defaultServerNames</td>
<td>The list of the default server names for each defined type, returned in the <code>ArrayOf_xsd_string</code> data type. See Section 2.3.1, &quot;ArrayOf_xsd_string.”</td>
</tr>
<tr>
<td>ArrayOf_xsd_String faxServerNames</td>
<td>The list of fax server names returned in the <code>ArrayOf_xsd_string</code> data type. See Section 2.3.1, &quot;ArrayOf_xsd_string.”</td>
</tr>
</tbody>
</table>
Complex Data Types

Data Types in Oracle BI Publisher Web Services

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

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

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String faxNumber</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>Fax Number is not specified.</td>
</tr>
<tr>
<td>String faxServer</td>
<td>Required. The fax server (defined on the BI Publisher server) 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:</td>
</tr>
<tr>
<td></td>
<td>Fax server is not specified.</td>
</tr>
</tbody>
</table>
Complex Data Types

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>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String dynamicDataSourcePath</td>
<td>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 &quot;false&quot;. For example: &quot;D:\BI\OracleBI\xmlp\XMLP\DemoFiles\Balance.xml&quot;)&quot; If the file is located in the system temporary directory, set temporaryDataSource to true, and specify the file name here. For example: &quot;Balance.xml&quot;.</td>
</tr>
<tr>
<td>boolean temporaryDataSource</td>
<td>Set to &quot;true&quot; when the file data source is in the system temporary directory. Set to &quot;false&quot; when dynamicDataSourcePath specifies the full path.</td>
</tr>
</tbody>
</table>

2.3.26 **FTPDeliveryOption**

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

This type is used in the ArrayOfFTPDeliveryOption complex data type.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String ftpServerName</td>
<td>Required. The FTP server name (for example, myftpserver.com.example). If this field is empty, a SOAP fault is thrown with the following message: FTP server is not specified.</td>
</tr>
<tr>
<td>String ftpUserName</td>
<td>A user name for the FTP server.</td>
</tr>
<tr>
<td>String ftpUserPassword</td>
<td>The password for the user entered.</td>
</tr>
<tr>
<td>String remoteFile</td>
<td>The name to assign the file on the server. For example: report.pdf.</td>
</tr>
<tr>
<td>boolean sftpOption</td>
<td>The value true indicates to use the secure FTP option. If ftpUserPassword is specified, sftpOption must be set to true.</td>
</tr>
</tbody>
</table>

2.3.27 **ItemData**

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String absolutePath</td>
<td>The path to the report object. For example: /HR Manager/ HR Reports/Employee Listing.xdo</td>
</tr>
<tr>
<td>dateTime creationDate</td>
<td>The creation date of the report object.</td>
</tr>
<tr>
<td>String displayName</td>
<td>The display name for the report object. For example: Employee Listing</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String JDBCDriverClass</td>
<td>The JDBC driver class for the data source (for example, oracle.jdbc.OracleDriver).</td>
</tr>
<tr>
<td>String JDBCDriverType</td>
<td>The driver type as defined in the BI Publisher data source definition page (for example, Oracle 9i/10g/11g).</td>
</tr>
<tr>
<td>String JDBCPassword</td>
<td>The password for the data source as defined in the BI Publisher data source definition page.</td>
</tr>
<tr>
<td>String JDBCURL</td>
<td>The connection string for the data source (for example, jdbc:oracle:thin:@mydatabase.com.example:1521: orcl).</td>
</tr>
<tr>
<td>String JDBCUserName</td>
<td>The user name for the data source as defined in the BI Publisher data source definition page.</td>
</tr>
<tr>
<td>String dataSourceName</td>
<td>The Data Source Name assigned to the data source in the BI Publisher data source definition page (for example, Oracle).</td>
</tr>
</tbody>
</table>

2.3.29 JobDetail

The collection of information about a job request.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean bursting</td>
<td>The value true indicates the bursting option is enabled for the job.</td>
</tr>
<tr>
<td>String burstingParameters</td>
<td>The parameters for the bursting engine.</td>
</tr>
<tr>
<td>dateTime created</td>
<td>The date the job was created.</td>
</tr>
<tr>
<td>String dataLocator</td>
<td>When the storageType is DB, dataLocator is the primary key for retrieving the data from database.</td>
</tr>
<tr>
<td>boolean deleted</td>
<td>The value true indicates the job was deleted.</td>
</tr>
</tbody>
</table>
### Table 2-31 (Cont.) Fields Provided by JobDetail

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String deliveryDescription</td>
<td>The description for the job delivery.</td>
</tr>
<tr>
<td>String deliveryParameters</td>
<td>The parameters for the delivery channels.</td>
</tr>
<tr>
<td>dateTime endDate</td>
<td>The date the job is scheduled to end.</td>
</tr>
<tr>
<td>int instanceId</td>
<td>The numeric identification for the job instance.</td>
</tr>
<tr>
<td>String issuer</td>
<td>The issuer of the job.</td>
</tr>
<tr>
<td>String jobGroup</td>
<td>The group to which the job belongs.</td>
</tr>
<tr>
<td>int jobId</td>
<td>The numeric identification for the job instance.</td>
</tr>
<tr>
<td>int jobSetId</td>
<td>Inactive. Do not use.</td>
</tr>
<tr>
<td>String jobType</td>
<td>The type of job.</td>
</tr>
<tr>
<td>dateTime lastUpdated</td>
<td>The date and time the job was last updated.</td>
</tr>
<tr>
<td>String locale</td>
<td>The locale to which the job belongs.</td>
</tr>
<tr>
<td>String notificationParameters</td>
<td>The notification parameters for the job.</td>
</tr>
<tr>
<td>String owner</td>
<td>The name of the job owner.</td>
</tr>
<tr>
<td>int parentJobId</td>
<td>The numeric identifier for the parent of the job.</td>
</tr>
<tr>
<td>boolean public</td>
<td>Whether the job is viewable by other users (true) or not (false).</td>
</tr>
<tr>
<td>String reportParameters</td>
<td>The parameters for the report.</td>
</tr>
<tr>
<td>String reportUrl</td>
<td>The URL to the job output.</td>
</tr>
<tr>
<td>String runType</td>
<td>The type of job (either Single or Recurring).</td>
</tr>
<tr>
<td>String scheduleContext</td>
<td>The context for the job schedule when external applications submit the report.</td>
</tr>
<tr>
<td>String scheduleDescription</td>
<td>The description of the job schedule.</td>
</tr>
<tr>
<td>String scheduleParameters</td>
<td>The parameters of the job schedule.</td>
</tr>
<tr>
<td>String scheduleSource</td>
<td>The source of the job schedule, which is used with the scheduleContext to provide information on external applications.</td>
</tr>
<tr>
<td>dateTime startDate</td>
<td>The date the job is scheduled to start or started.</td>
</tr>
<tr>
<td>String status</td>
<td>The status of the job.</td>
</tr>
<tr>
<td>String statusDetail</td>
<td>The details of the job status.</td>
</tr>
<tr>
<td>String storageType</td>
<td>The storage type for the job. Supported value is DB.</td>
</tr>
<tr>
<td>String userDescription</td>
<td>A user-assigned description for the job.</td>
</tr>
<tr>
<td>String userJobName</td>
<td>The user-assigned named for the job.</td>
</tr>
<tr>
<td>boolean xmlDataAvailable</td>
<td>Whether XML data for the job is available (true) or not (false).</td>
</tr>
<tr>
<td>boolean xmlDataCompressed</td>
<td>A value of true indicates the XML data for the job is or will be compressed.</td>
</tr>
<tr>
<td>String xmlDataContentType</td>
<td>The content type of the XML data.</td>
</tr>
<tr>
<td>String xschurl</td>
<td>The URL of the report.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>endTime</code></td>
<td>The time the job is scheduled to end or ended.</td>
</tr>
<tr>
<td><code>endTimeOperator</code></td>
<td>The operator for <code>endTime</code>. Valid values are &quot;Equals or Earlier Than&quot;, &quot;Equals&quot; or &quot;Earlier Than&quot;.</td>
</tr>
<tr>
<td><code>jobID</code></td>
<td>The numeric identification assigned by BI Publisher for the job request.</td>
</tr>
<tr>
<td><code>jobName</code></td>
<td>The user-assigned job name. Valid values are &quot;Contains&quot; or &quot;Equals&quot;.</td>
</tr>
<tr>
<td><code>jobNameOperator</code></td>
<td>The operator of the <code>jobName</code>.</td>
</tr>
<tr>
<td><code>owner</code></td>
<td>The name of the job's owner.</td>
</tr>
<tr>
<td><code>ownerOperator</code></td>
<td>The operator for <code>owner</code>. Valid values are &quot;Contains&quot; or &quot;Equals&quot;.</td>
</tr>
<tr>
<td><code>reportName</code></td>
<td>The user-assigned named for the job.</td>
</tr>
<tr>
<td><code>reportNameOperator</code></td>
<td>The operator for <code>reportName</code>. Valid values are &quot;Contains&quot; or &quot;Equals&quot;.</td>
</tr>
<tr>
<td><code>scope</code></td>
<td>The scope of the job. Valid values are &quot;All&quot;, &quot;Private&quot; or &quot;Public&quot;.</td>
</tr>
<tr>
<td><code>startTime</code></td>
<td>The time the job is scheduled to start or started.</td>
</tr>
<tr>
<td><code>startTimeOperator</code></td>
<td>The operator for <code>startTime</code>. Valid values are &quot;Equals or Later Than&quot;, &quot;Equals&quot; or &quot;Later Than&quot;.</td>
</tr>
<tr>
<td><code>status</code></td>
<td>The current status for the job.</td>
</tr>
</tbody>
</table>

2.3.31 JobInfo

The collection of information about a job request.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>burstingJob</code></td>
<td>A value of <code>true</code> indicates the bursting option is enabled.</td>
</tr>
<tr>
<td><code>created</code></td>
<td>The date the job was created.</td>
</tr>
<tr>
<td><code>deleted</code></td>
<td>The value of <code>true</code> indicates the job was deleted.</td>
</tr>
<tr>
<td><code>endDate</code></td>
<td>The date the job is scheduled to end or ended.</td>
</tr>
<tr>
<td><code>instanceId</code></td>
<td>The numeric identification for the scheduled job request.</td>
</tr>
<tr>
<td><code>jobID</code></td>
<td>The numeric identification assigned by BI Publisher to the job request.</td>
</tr>
</tbody>
</table>
### 2.3.32 JobInfoList

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

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayOfJobInfo jobInfoList</td>
<td>An array of JobInfo objects. See Section 2.3.6, “ArrayOfJobInfo.”</td>
</tr>
</tbody>
</table>

### 2.3.33 JobOutput

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

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String burstKey</td>
<td>The key used to split the data for each bursted job.</td>
</tr>
<tr>
<td>dateTime created</td>
<td>The date the report job was created.</td>
</tr>
<tr>
<td>boolean deleted</td>
<td>The value &quot;true&quot; indicates the job output was deleted.</td>
</tr>
<tr>
<td>boolean documentDataAvailable</td>
<td>True indicates that the user selected the &quot;Save Output&quot; option when the report was scheduled.</td>
</tr>
<tr>
<td>boolean documentDataCompressed</td>
<td>True indicates that the document data is compressed.</td>
</tr>
</tbody>
</table>
Table 2–35 (Cont.) Fields Provided by JobOutput

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

2.3.34 JobOutputDelivery

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

Table 2–36 Fields Provided by JobOutputDelivery

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

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayOfJobOutputDelivery</td>
<td>See Section 2.3.8, “ArrayOfJobOutputDelivery.”</td>
</tr>
</tbody>
</table>

2.3.36 JobOutputsList

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayOfJobOutput</td>
<td>See Section 2.3.7, “ArrayOfJobOutput.”</td>
</tr>
</tbody>
</table>

2.3.37 JobStatus

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String jobID</td>
<td>The numeric identification assigned by BI Publisher to the job request.</td>
</tr>
<tr>
<td>String jobStatus</td>
<td>The current status of the job.</td>
</tr>
<tr>
<td>String message</td>
<td>Details on the job status.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String metaDataName</td>
<td>The name of the metadata.</td>
</tr>
<tr>
<td>String metaDataValue</td>
<td>The metadata content.</td>
</tr>
</tbody>
</table>
2.3.40 MetaDataList

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayOfMetaData</td>
<td>See Section 2.3.10, &quot;ArrayOfMetaData.&quot;</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListOfParamNameValues</td>
<td>See Section 2.3.11, &quot;ArrayOfParamNameValue.&quot;</td>
</tr>
</tbody>
</table>

### 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>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrintNumberOfCopy</td>
<td>The number of copies to print.</td>
</tr>
<tr>
<td>PrintOrientation</td>
<td>Valid values are &quot;portrait&quot; or &quot;landscape&quot;.</td>
</tr>
<tr>
<td>PrintRange</td>
<td>A range of pages to print. Separate multiple ranges with a comma.</td>
</tr>
<tr>
<td>PrintSide</td>
<td>Valid values are &quot;Single sided&quot;, &quot;Double Sided Long Edge (Duplex)&quot;, and &quot;Double Sided Short Edge (Tumble)&quot;.</td>
</tr>
<tr>
<td>PrintTray</td>
<td>Valid values are &quot;Default&quot;, &quot;Tray 1&quot;, &quot;Tray 2&quot;, and &quot;Tray 3&quot;.</td>
</tr>
<tr>
<td>PrinterName</td>
<td>Required. The name of the printer to which to send the report. If empty, a SOAP fault is thrown with the following message: Print server is not specified.</td>
</tr>
</tbody>
</table>

### 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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte[] reportDataChunk</td>
<td>Byte[] array representing binary report data transported between the BI Publisher client and server.</td>
</tr>
<tr>
<td>String reportDataFileID</td>
<td>The identifier for the data file of the report on the BI Publisher server.</td>
</tr>
<tr>
<td>long reportDataOffset</td>
<td>The offset value for the location of the previously downloaded report data file.</td>
</tr>
</tbody>
</table>

#### 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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean autoRun</td>
<td>True indicates that the report property Auto Run is turned on.</td>
</tr>
<tr>
<td>boolean cacheDocument</td>
<td>True indicates that the report property Enable document cache is turned on.</td>
</tr>
<tr>
<td>boolean controledByExtApp</td>
<td>Whether the report definition is controlled by an external application (true) or not (false).</td>
</tr>
<tr>
<td>String dataModelURL</td>
<td>The .xdm location from where to get the Data Model definition.</td>
</tr>
<tr>
<td>String defaultOutputFormat</td>
<td>The default output format. Valid values include:</td>
</tr>
<tr>
<td></td>
<td>■ csv (CSV)</td>
</tr>
<tr>
<td></td>
<td>■ data (Data)</td>
</tr>
<tr>
<td></td>
<td>■ eText (eText template)</td>
</tr>
<tr>
<td></td>
<td>■ excel (Microsoft Excel)</td>
</tr>
<tr>
<td></td>
<td>■ excel2000 (Microsoft Excel 2000)</td>
</tr>
<tr>
<td></td>
<td>■ flash (Adobe Flash)</td>
</tr>
<tr>
<td></td>
<td>■ html (HTML)</td>
</tr>
<tr>
<td></td>
<td>■ mhtml (MIME HTML)</td>
</tr>
<tr>
<td></td>
<td>■ pdf (Adobe PDF)</td>
</tr>
<tr>
<td></td>
<td>■ ppt (Microsoft PowerPoint)</td>
</tr>
<tr>
<td></td>
<td>■ rtf (Rich Text Format)</td>
</tr>
<tr>
<td>String defaultTemplateId</td>
<td>The default template identified for the report.</td>
</tr>
<tr>
<td>boolean diagnostics</td>
<td>True indicates that diagnostics have been turned on for the report.</td>
</tr>
<tr>
<td>String ESSJobName</td>
<td>The ESS job name.</td>
</tr>
<tr>
<td>String ESSPackageName</td>
<td>The ESS package name. Used in conjunction with ESSJobName.</td>
</tr>
<tr>
<td>ArrayOfTemplateFormatsLabelValues listOfTemplateFormatsLabelValues</td>
<td>Passes the list of template format labels through the ArrayOfTemplateFormatsLabelValues data type. See Section 2.3.15, &quot;ArrayOfTemplateFormatLabelValues.&quot;</td>
</tr>
<tr>
<td>boolean onLine</td>
<td>True indicates the property &quot;Run report online&quot; is turned on for the report.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String attributeCalendar</td>
<td>The formatting calendar to use for the report request. Valid values are: &quot;Gregorian&quot;, &quot;Arabic Hijrah&quot;, &quot;English Hijrah&quot;, &quot;Japanese Imperial&quot;, &quot;Thai Buddha&quot;, and &quot;ROC Official&quot;.</td>
</tr>
<tr>
<td>String attributeFormat</td>
<td>The output format of the requested report. Valid values are: pdf, rtf, html, excel, excel2000, mhtml, csv, data, flash, and powerpoint.</td>
</tr>
<tr>
<td>String attributeLocale</td>
<td>The locale selection for the report. Example: fr-FR</td>
</tr>
<tr>
<td>String attributeTemplate</td>
<td>The template to apply to the report. For example: Employeelisting.rtf.</td>
</tr>
<tr>
<td>String attributeTimeZone</td>
<td>Specifies the time zone to use for the request, using a supported Java time zone ID. For example: &quot;America/Los_Angeles&quot;.</td>
</tr>
<tr>
<td>boolean byPassCache</td>
<td>True indicates to bypass document cache.</td>
</tr>
<tr>
<td>BIPDataSource</td>
<td>If the data source for the report is not defined, you can dynamically define it. See Section 2.3.17, &quot;BIPDataSource.&quot;</td>
</tr>
<tr>
<td>boolean flattenXML</td>
<td>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.</td>
</tr>
<tr>
<td>ParamNameValues</td>
<td>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, &quot;ParamNameValues.&quot;</td>
</tr>
</tbody>
</table>
2.3.47 ReportResponse

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 2–49 Fields Provided by ReportResponse

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>The absolute path to the report in the BI Publisher repository. For example: /HR Manager/HR Reports/Employee Listing.xdo.</td>
</tr>
<tr>
<td>byte[] reportData</td>
<td>If you are providing the data directly for the report use this element to pass the data.</td>
</tr>
<tr>
<td>String reportOutputPath</td>
<td>Specifies the output path for the generated report.</td>
</tr>
<tr>
<td>String reportRawData</td>
<td>If raw XML data is used for the report, this element contains the XML data.</td>
</tr>
<tr>
<td>integer sizeOfDataChunkDownload</td>
<td>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.</td>
</tr>
<tr>
<td>MetaDataList</td>
<td>See Section 2.3.40, &quot;MetaDataList.&quot;</td>
</tr>
<tr>
<td>XDOPROPERTYLIST</td>
<td></td>
</tr>
</tbody>
</table>

2.3.48 ScheduleRequest

The options to schedule a report.

Table 2–50 Fields Provided by ScheduleRequest

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean bookBindingOutputOption</td>
<td>Whether the book binding output is enabled (true) or not (false).</td>
</tr>
<tr>
<td>String dataModelUrl</td>
<td>The location of the .xdm file from which to obtain the Data Model definition.</td>
</tr>
</tbody>
</table>
### Table 2–50  (Cont.) Fields Provided by ScheduleRequest

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeliveryChannels</td>
<td>See Section 2.3.20, &quot;DeliveryChannels.&quot;</td>
</tr>
<tr>
<td>String endDate</td>
<td>The end date of the schedule.</td>
</tr>
<tr>
<td>String jobLocale</td>
<td>The locale to use for the scheduled requests. Example: fr-FR</td>
</tr>
<tr>
<td>String jobTZ</td>
<td>The time zone to use for the scheduled requests.</td>
</tr>
<tr>
<td>boolean mergeOutputOption</td>
<td>Whether the merge output option is enabled (true) or not (false).</td>
</tr>
<tr>
<td>String notificationPassword</td>
<td>The HTTP notification server password when scheduling notification through an HTTP server.</td>
</tr>
<tr>
<td>String notificationServer</td>
<td>The name of the HTTP server used for notification.</td>
</tr>
<tr>
<td>String notificationTo</td>
<td>E-mail addresses to which to send notifications.</td>
</tr>
<tr>
<td>String notificationUserName</td>
<td>The user name for the HTTP server used for notification.</td>
</tr>
<tr>
<td>boolean notifyHttpWhenFailed</td>
<td>True indicates to send a notification when the job request fails.</td>
</tr>
<tr>
<td>boolean notifyHttpWhenSuccess</td>
<td>True indicates to send a notification when the job request succeeds.</td>
</tr>
<tr>
<td>boolean notifyHttpWhenWarning</td>
<td>True indicates to send a notification when the job completes with a warning.</td>
</tr>
<tr>
<td>boolean notifyWhenFailed</td>
<td>True indicates to send a notification when the job request fails.</td>
</tr>
<tr>
<td>boolean notifyWhenSuccess</td>
<td>True indicates to send a notification when the job request succeeds.</td>
</tr>
<tr>
<td>boolean notifyWhenWarning</td>
<td>True indicates to send a notification when the job completes with a warning.</td>
</tr>
<tr>
<td>String recurrenceExpression</td>
<td>The expression that defines a recurring schedule.</td>
</tr>
<tr>
<td>String recurrenceExpressionType</td>
<td>The type of expression defined for a recurring schedule. Valid value is cron.</td>
</tr>
<tr>
<td>integer repeatCount</td>
<td>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.</td>
</tr>
<tr>
<td>integer repeatInterval</td>
<td>The interval between two scheduled jobs in seconds.</td>
</tr>
<tr>
<td>ReportRequest</td>
<td>Information about the request included through the ReportRequest data type. See Section 2.3.46, &quot;ReportRequest.&quot;</td>
</tr>
<tr>
<td>boolean saveDataOption</td>
<td>True indicates that the report data from the scheduled request run will be saved.</td>
</tr>
<tr>
<td>boolean scheduleBurstingOption</td>
<td>True indicates that the scheduled requests will be burst.</td>
</tr>
<tr>
<td>boolean schedulePublicOption</td>
<td>True indicates that the scheduled requests are to be made public.</td>
</tr>
<tr>
<td>String startDate</td>
<td>The date on which the schedule starts.</td>
</tr>
<tr>
<td>boolean useUTF8Option</td>
<td>True indicates that the Use UTF8 option is enabled.</td>
</tr>
<tr>
<td>String userJobDesc</td>
<td>The user-entered description for the scheduled job.</td>
</tr>
<tr>
<td>String userJobName</td>
<td>The user-entered name for the scheduled job.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String templateFormatLabel</td>
<td>The label that displays for a template format. Examples include: &quot;HTML&quot; &quot;PDF&quot; &quot;Excel&quot;</td>
</tr>
<tr>
<td>String templateFormatValue</td>
<td>The template format value that corresponds to the label. Examples include: &quot;html&quot; &quot;pdf&quot; &quot;excel&quot;</td>
</tr>
</tbody>
</table>

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

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 2–53  Fields Provided by WebDAVDeliveryOption

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String deliveryAuthType</td>
<td>Authentication type. Valid values are: None, Basic, Digest</td>
</tr>
<tr>
<td>String password</td>
<td>If a proxy server has been set up, the password required to access the proxy server.</td>
</tr>
<tr>
<td>String remoteFilePath</td>
<td>The path to directory on the remote server to which to deliver the report file.</td>
</tr>
<tr>
<td>String server</td>
<td>Required. The WebDAV server name (for example, myserver). If empty, a SOAP fault is thrown with the following message: WebDAV server is not specified.</td>
</tr>
<tr>
<td>String userName</td>
<td>If a proxy server has been set up, the user name required to access the proxy server.</td>
</tr>
</tbody>
</table>
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 3.1, "cancelSchedule() Method"
- Section 3.2, "cancelScheduleInSession() Method"
- Section 3.3, "deleteJobHistory() Method"
- Section 3.4, "deleteJobHistoryInSession() Method"
- Section 3.5, "deleteSchedule() Method"
- Section 3.6, "deleteScheduleInSession() Method"
- Section 3.7, "deliveryService() Method"
- Section 3.8, "deliveryServiceInSession() Method"
- Section 3.9, "downloadDocumentData() Method"
- Section 3.10, "downloadDocumentDataInSession() Method"
- Section 3.11, "downloadXMLData() Method"
- Section 3.12, "downloadXMLDataInSession() Method"
- Section 3.13, "getAllScheduledReport() Method"
- Section 3.14, "getAllScheduledReportHistory() Method"
- Section 3.15, "getAllScheduledReportHistoryInSession() Method"
- Section 3.16, "getAllScheduledReportInSession() Method"
- Section 3.17, "getDeliveryServiceDefinition() Method"
- Section 3.18, "getDeliveryServiceDefinitionInSession() Method"
- Section 3.19, "getDocumentData() Method"
- Section 3.20, "getDocumentDataInSession() Method"
- Section 3.21, "getScheduledJobInfo() Method"
- Section 3.22, "getScheduledJobInfoInSession() Method"
- Section 3.23, "getScheduledReportDeliveryInfo() Method"
- Section 3.24, "getScheduledReportDeliveryInfoInSession() Method"
- Section 3.25, "getScheduledReportOutputInfo() Method"
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String jobInstanceID</td>
<td>The ID assigned to the instance of the job to be canceled. The jobInstanceID is a string of integers.</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

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);
### 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.

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

**Signature**

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String instanceJobID</td>
<td>The ID assigned to the instance of the job that generated the output. The instanceJobID is a string of integers.</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

### 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."

```java
boolean deleteJobHistoryInSession(String jobInstanceID, String bipSessionToken);
```

**Signature**

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String jobInstanceID</td>
<td>The ID assigned to the instance of the job that generated the output. The jobInstanceID is a string of integers.</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 3.5 deleteSchedule() Method

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

```java
boolean deleteScheduleInSession(String jobInstanceID, String bipSessionToken);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>jobInstanceID</code></td>
<td>The ID assigned to the instance of the job that generated the output.</td>
</tr>
<tr>
<td><code>bipSessionToken</code></td>
<td>The proprietary token string generated for the user by the BI Publisher server.</td>
</tr>
</tbody>
</table>

3.7 `deliveryService()` Method

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

### Signature

```java
String deliveryService(DeliveryRequest deliveryRequest, String userID, String password);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>deliveryRequest</code></td>
<td>The <code>DeliveryRequest</code> object.</td>
</tr>
<tr>
<td><code>userID</code></td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td><code>password</code></td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

3.8 `deliveryServiceInSession()` Method

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

### Signature

```java
String deliveryServiceInSession(DeliveryRequest deliveryRequest, String jobInstanceID, String bipSessionToken);
```
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeliveryRequest deliveryRequest</td>
<td>The DeliveryRequest object. See Section 2.3.21, &quot;DeliveryRequest.&quot;</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String jobOutputID</td>
<td>The ID assigned to the output. The jobOutputID is a string of integers.</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String jobInstanceID</td>
<td>The ID assigned to the instance of the job that generated the output. The jobInstanceID is a string of integers.</td>
</tr>
</tbody>
</table>
### 3.11 `downloadXMLData()` Method

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

**Signature**

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String <code>jobInstanceID</code></td>
<td>The ID assigned to the XML data. The <code>jobInstanceID</code> is a string of integers.</td>
</tr>
<tr>
<td>String <code>userID</code></td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String <code>password</code></td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

### 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**

```java
String downloadXMLDataInSession(String jobInstanceID, String `bipSessionToken`);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String <code>jobInstanceID</code></td>
<td>The ID assigned to the instance of the job that generated the output. The <code>jobInstanceID</code> is a string of integers.</td>
</tr>
<tr>
<td>String <code>bipSessionToken</code></td>
<td>The proprietary token string generated for the user by the BI Publisher server. With the <code>bipSessionToken</code> 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.</td>
</tr>
</tbody>
</table>

### 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.
getAllScheduledReportHistoryInSession() Method

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

Table 3–13 Parameters for getAllScheduledReport() Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JobFilterProperties filter</td>
<td>The JobFilterProperties object specifies the specific criteria for the report jobs you want to return information about. See Section 2.3.30, &quot;JobFilterProperties.&quot;</td>
</tr>
<tr>
<td>int beginIdx</td>
<td>The starting point of the index (default is 1).</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JobFilterProperties filter</td>
<td>The JobFilterProperties object specifies the specific criteria for the report jobs you want to return information about. See Section 2.3.30, &quot;JobFilterProperties.&quot;</td>
</tr>
<tr>
<td>int beginIdx</td>
<td>The starting point of the index (default is 1).</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

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);
### 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**

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>JobFilterProperties filter</code></td>
<td>The <code>JobFilterProperties</code> object specifies the specific criteria for the report jobs you want to return information about. See Section 2.3.30, &quot;JobFilterProperties.&quot;</td>
</tr>
<tr>
<td><code>int beginIdx</code></td>
<td>The starting point of the index (default is 1).</td>
</tr>
<tr>
<td><code>String bipSessionToken</code></td>
<td>The proprietary token string generated for the user by the BI Publisher server. With the <code>bipSessionToken</code> 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.</td>
</tr>
</tbody>
</table>

### 3.17 `getDeliveryServiceDefinition()` Method

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

**Signature**

```java
DeliveryServiceDefinition getDeliveryServiceDefinition(String userID, String password);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>String userID</code></td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td><code>String password</code></td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

3.19 getDocumentData() Method

Use the getDocumentData() method to return the document generated by a BI Publisher scheduled job. You can use the JobOutputID returned from the scheduleReport() method to retrieve the generated report document.

Note, 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 quite large.

Signature

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

Table 3–19 Parameters for getDocumentData() Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String JobOutputID</td>
<td>Job output assigned to the output. The output ID is a string of integers.</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

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);
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**

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>jobInstanceID</code></td>
<td>The ID of the job for which to return job information.</td>
</tr>
<tr>
<td><code>userID</code></td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td><code>password</code></td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

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**

```java
JobDetail getScheduledJobInfoInSession(String jobInstanceID, String bipSessionToken);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>jobInstanceID</code></td>
<td>The ID assigned to the job instance. The <code>jobInstanceID</code> is a string of integers.</td>
</tr>
<tr>
<td><code>bipSessionToken</code></td>
<td>The proprietary token string generated for the user by the BI Publisher server. With the <code>bipSessionToken</code> 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.</td>
</tr>
</tbody>
</table>
3.23 get ScheduledReportDeliveryInfo() Method

Use the get ScheduledReportDeliveryInfo() method to retrieve information about the delivery of a scheduled job output. For each scheduled job, it could have multiple output IDs. For each output ID, there could be multiple delivery information. See Section 2.3.35, "JobOutputDeliverysList."

**Signature**

```java
JobOutputDeliverysList get ScheduledReportDeliveryInfo(String jobOutputID, String userID, String password);
```

**Table 3–23 Parameters for get ScheduledReportDeliveryInfo() Method**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String jobOutputID</td>
<td>The ID assigned to the output of the job for which you want information. The jobOutputID is a string of integers.</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

3.24 get ScheduledReportDeliveryInfoInSession() 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**

```java
JobOutputDeliverysList get ScheduledReportDeliveryInfoInSession(String jobOutputID, String bipSessionToken);
```

**Table 3–24 Parameters for get ScheduledReportDeliveryInfoInSession() Method**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String jobOutputID</td>
<td>The ID assigned to the job output. The jobOutputID is a string of integers.</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

3.25 get ScheduledReportOutputInfo() Method

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

**Signature**

```java
JobOutputsList get ScheduledReportOutputInfo(String jobInstanceID, String userID, String password);
```
3.26 *getScheduledReportOutputInfoInSession() Method*

Use the `getScheduledReportOutputInfoInSession()` 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**

```java
JobOutputsList getScheduledReportOutputInfoInSession(String jobInstanceID, String bipTokenSession);
```

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String jobInstanceID</td>
<td>The ID of the job for which to return job information.</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String jobInstanceID</td>
<td>The ID of the job for which to return job information.</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

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**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String JobInstanceID</td>
<td>The ID assigned to the instance of the job that generated the output. The JobInstanceID is a string of integers.</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String jobInstanceID</td>
<td>The ID assigned to the job instance. The jobInstanceID is a string of integers.</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String instanceJobID</td>
<td>The ID assigned to the instance of the job that generated the output. The instanceJobID is a string of integers.</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String instanceJobID</td>
<td>The ID assigned to the job instance. The instanceJobID is a string of integers.</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 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**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String outputJobID</td>
<td>The ID of the scheduled job to resend.</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

### 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**

```java
boolean resendScheduledReportInSession(String outputJobID, String bipSessionToken);
```

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String outputJobID</td>
<td>The ID of the scheduled job to resend.</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 3.33 resumeSchedule() Method

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

```java
boolean resumeScheduleInSession(String outputJobID, String bipSessionToken);
```

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String jobInstanceID</td>
<td>The ID assigned to the instance of the job to be resumed.</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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 delivery types that are set up in your BI Publisher Enterprise Server instance. The method will return a jobID of the scheduled job.

**Signature**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScheduleRequest scheduleRequest</td>
<td>Specifies a ScheduleRequest object for the report that you want to run. See Section 2.3.48, &quot;ScheduleRequest.&quot;</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>
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**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScheduleRequest scheduleRequest</td>
<td>Specifies a ScheduleRequest object for the report that you want to run. See Section 2.3.48, &quot;ScheduleRequest.&quot;.</td>
</tr>
<tr>
<td>DeliveryChannel deliveryChannel</td>
<td>Specifies the delivery channels through which the report will be delivered. See Section 2.3.20, &quot;DeliveryChannels.&quot;</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

3.37 suspendSchedule() Method

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

**Signature**

```java
boolean suspendSchedule(String jobInstanceID, String userID, String password);
```

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String jobInstanceID</td>
<td>The ID assigned to the instance of the job to be suspended. The jobInstanceID is a string of integers.</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

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**

```java
boolean suspendScheduleInSession(String jobInstanceID, String bipSessionToken);
```
### Table 3–38 Parameters for `suspendScheduleInSession()` Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String jobInstanceID</td>
<td>The ID assigned to the instance of the job to be suspended. The <code>jobInstanceID</code> is a string of integers.</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>The proprietary token string generated for the user by the BI Publisher server. With the <code>bipSessionToken</code> 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.</td>
</tr>
</tbody>
</table>
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 4.1, "createReport() Method"
- Section 4.2, "createReportInSession() Method"
- Section 4.3, "downloadReportDataChunk() Method"
- Section 4.4, "downloadReportDataChunkInSession() Method"
- Section 4.5, "getReportDefinition() Method"
- Section 4.6, "getReportDefinitionInSession() Method"
- Section 4.7, "getReportParameters() Method"
- Section 4.8, "getReportParametersInSession() Method"
- Section 4.9, "getReportSampleData() Method"
- Section 4.10, "getReportSampleDataInSession() Method"
- Section 4.11, "getTemplate() Method"
- Section 4.12, "getTemplateInSession() Method"
- Section 4.13, "getTemplateParameters() Method"
- Section 4.14, "getTemplateParameterInSession() Method"
- Section 4.15, "getXDOSchema() Method"
- Section 4.16, "getXDOSchemaInSession() Method"
- Section 4.17, "removeTemplateForReport() Method"
- Section 4.18, "removeTemplateForReportInSession() Method"
- Section 4.19, "runReport() Method"
- Section 4.20, "runReportInSession() Method"
- Section 4.21, "updateReportDefinition() Method"
- Section 4.22, "updateReportDefinitionInSession() Method"
- Section 4.23, "updateTemplateForReport() Method"
- Section 4.24, "updateTemplateForReportInSession() Method"
- Section 4.25, "updateXLIFFForReport() Method"
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportName</td>
<td>The report name to create with the suffix &quot;.xdo&quot;. For example, &quot;myreport.xdo&quot;.</td>
</tr>
<tr>
<td>String folderAbsolutePathURL</td>
<td>The path to the folder in which to place the created report. For example: xmlp/Reports/financials</td>
</tr>
<tr>
<td>String dataModelURL</td>
<td>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</td>
</tr>
<tr>
<td>String templateFileName</td>
<td>The file name of the template to add the report definition.</td>
</tr>
<tr>
<td>byte[] templateData</td>
<td>The template file.</td>
</tr>
<tr>
<td>String XLIFFFileName</td>
<td>The file name of the XLIFF file. You must append the locale to the XLIFF file name as follows: template_&lt;language code&gt;_&lt;country code&gt;.xlf where &lt;language code&gt; is the two-letter ISO 639 language code &lt;country code&gt; is the two-letter ISO 639 language code For example: template_en_us.xlf</td>
</tr>
<tr>
<td>byte[] XLIFFData</td>
<td>The XLIFF file.</td>
</tr>
<tr>
<td>boolean updateFlag</td>
<td>If true, overwrites existing report. If false, throws error if the report exists.</td>
</tr>
</tbody>
</table>
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**

```java
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportName</td>
<td>The report name to create with the suffix &quot;.xdo&quot;. For example, &quot;myreport.xdo&quot;.</td>
</tr>
<tr>
<td>String folderAbsolutePathURL</td>
<td>The path to the folder in which to place the created report. For example: xmlp/Reports/financials</td>
</tr>
<tr>
<td>String dataModelURL</td>
<td>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</td>
</tr>
<tr>
<td>String templateFileName</td>
<td>The file name of the template to add the report definition.</td>
</tr>
<tr>
<td>byte[] templateData,</td>
<td>The template file.</td>
</tr>
<tr>
<td>String XLIFFFileName</td>
<td>The file name of the XLIFF file. You must append the locale to the XLIFF file name as follows: template_&lt;language_code&gt;_&lt;country_code&gt;.xlf where &lt;language_code&gt; is the two-letter ISO 639 language code &lt;country_code&gt; is the two-letter ISO 639 language code For example: template_en_us.xlf</td>
</tr>
<tr>
<td>byte[] XLIFFData</td>
<td>The XLIFF file.</td>
</tr>
<tr>
<td>boolean updateFlag</td>
<td>If true, overwrites existing report. If false, throws error if the report exists.</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>
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.
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**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String fileID</td>
<td>fileID is returned inside <code>ReportResponse</code>, which is returned when calling <code>runReport()</code> Method.</td>
</tr>
<tr>
<td>int beginIdx</td>
<td>The starting point of the index (default is 1).</td>
</tr>
<tr>
<td>int size</td>
<td>The size of the file to download (in kilobytes).</td>
</tr>
</tbody>
</table>

4.4 `downloadReportDataChunkInSession()` Method

Use `downloadReportDataChunkInSession()` 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**

```java
ReportDataChunk downloadReportDataChunkInSession(String fileID, int beginIdx, int size, String bipSessionToken);
```
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>The path to the report for which to retrieve the report definition. For example: /HR Manager/Employee Reports/Employee Listing.xdo</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

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);
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**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>The path to the report for which to retrieve the report definition. For example: /HR Manager/Employee Reports/Employee Listing.xdo</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReportRequest reportRequest</td>
<td>See Section 2.3.46, &quot;ReportRequest.&quot;</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

4.8 **getReportParametersInSession() Method**

Use the `getReportParametersInSession()` 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**

```java
ParamNameValues getReportParametersInSession(ReportRequest reportRequest, String bipSessionToken);
```

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReportRequest reportRequest</td>
<td>See Section 2.3.46, &quot;ReportRequest.&quot;</td>
</tr>
</tbody>
</table>
4.9 getReportSampleData() Method

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

**Signature**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>The path to the report for which to retrieve the report data model sample data. For example: /HR Manager/Employee Reports/Employee Listing.xdo</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

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**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>The path to the report for which to retrieve the report data model sample data. For example: /HR Manager/Employee Reports/Employee Listing.xdo</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>Specifies 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.</td>
</tr>
</tbody>
</table>

4.11 getTemplate() Method

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

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>The path to the report to which the template is associated. For example: /HR Manager/Employee Reports/Employee Listing.xdo</td>
</tr>
<tr>
<td>String templateID</td>
<td>The ID of the template (for example, Chart Layout).</td>
</tr>
<tr>
<td>String locale</td>
<td>The locale of the template to retrieve (for example, en_US).</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

**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**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>The path to the report to which the template is associated. For example: /HR Manager/Employee Reports/Employee Listing.xdo</td>
</tr>
<tr>
<td>String templateID</td>
<td>The ID of the template (for example, Chart Layout).</td>
</tr>
<tr>
<td>String locale</td>
<td>The locale of the template to retrieve (for example, en_US).</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

**4.13 getTemplateParameters() Method**

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

See Section 2.3.41, "ParamNameValue."

**Signature**

```java
ParamNameValue[] getTemplateParameters(String reportAbsolutePath, String templateID, String userID, String password);
```
4.14 getTemplateParameterInSession() Method

Use the getTemplateParameterInSession() 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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>The path to the report for which to retrieve the report definition. For example: /HR Manager/Employee Reports/Employee Listing.xdo</td>
</tr>
<tr>
<td>String templateID</td>
<td>The ID assigned to the template, for example: &quot;Chart Layout&quot;.</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>The path to the report from which to retrieve the XDO schema. For example: /HR Manager/Employee Reports/Employee Listing.xdo</td>
</tr>
<tr>
<td>String locale</td>
<td>The locale of the template to retrieve (for example, en_US).</td>
</tr>
</tbody>
</table>

Table 4–13 Parameters for getTemplateParameters() Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>The path to the report for which to retrieve the report definition. For example: /HR Manager/Employee Reports/Employee Listing.xdo</td>
</tr>
<tr>
<td>String templateID</td>
<td>The ID assigned to the template, for example: &quot;Chart Layout&quot;.</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

Table 4–14 Parameters for getTemplateParameterInSession() Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>The path to the report for which to retrieve the report definition. For example: /HR Manager/Employee Reports/Employee Listing.xdo</td>
</tr>
<tr>
<td>String templateID</td>
<td>The ID assigned to the template, for example: &quot;Chart Layout&quot;.</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

Table 4–15 Parameters for getXDOSchema() Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>The path to the report from which to retrieve the XDO schema. For example: /HR Manager/Employee Reports/Employee Listing.xdo</td>
</tr>
<tr>
<td>String locale</td>
<td>The locale of the template to retrieve (for example, en_US).</td>
</tr>
</tbody>
</table>
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**

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>The path to the report from which to retrieve the XDO schema. For example: <code>/HR Manager/Employee Reports/Employee Listing.xdo</code></td>
</tr>
<tr>
<td>String locale</td>
<td>The locale of the template to retrieve (for example, <code>en_US</code>).</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

4.17 removeTemplateForReport() Method

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

**Signature**

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>The path to the report from which to remove the template. For example: <code>/HR Manager/Employee Reports/Employee Listing.xdo</code></td>
</tr>
<tr>
<td>String templateFileName</td>
<td>The file name of the template to remove.</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>
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**

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>The path to the report from which to remove the template. For example: /HR Manager/Employee Reports/Employee Listing.xdo</td>
</tr>
<tr>
<td>String templateFileName</td>
<td>The file name of the template to remove.</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

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**

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReportRequest reportRequest</td>
<td>See Section 2.3.46, &quot;ReportRequest.&quot;</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

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."
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsPath</td>
<td>The path to the report for which to update the report definition. For example: /HR Manager/Employee Reports/Employee Listing.xdo</td>
</tr>
<tr>
<td>ReportDefinition newReportDefn</td>
<td>See Section 2.3.45, &quot;ReportDefinition.&quot;</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsPath</td>
<td>The path to the report for which to update the report definition. For example: /HR Manager/Employee Reports/Employee Listing.xdo</td>
</tr>
<tr>
<td>ReportDefinition newReportDefn</td>
<td>See Section 2.3.45, &quot;ReportDefinition.&quot;</td>
</tr>
</tbody>
</table>
4.23 updateTemplateForReport() Method

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

**Signature**

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

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

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

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**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>The path to the report that contains the template to update. For example: /HR Manager/Employee Reports/Employee Listing.xdo</td>
</tr>
<tr>
<td>String templateName</td>
<td>The name of the template to update (for example, Chart Layout).</td>
</tr>
<tr>
<td>String locale</td>
<td>The locale of the template to update (for example, en_US).</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>The path to the report to that contains the XLIFF file to update. For example: /HR Manager/Employee Reports/Employee Listing.xdo</td>
</tr>
<tr>
<td>byte[] xliffData</td>
<td>The XLIFF file to upload.</td>
</tr>
<tr>
<td>String layoutFileName</td>
<td>The file name of the layout for which the XLIFF file is to be updated. For example: employee_listing.rtf.</td>
</tr>
<tr>
<td>String locale</td>
<td>The locale to assign to the XLIFF (for example, en_US).</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>The path to the report to that contains the XLIFF file to update. For example: /HR Manager/Employee Reports/Employee Listing.xdo</td>
</tr>
<tr>
<td>byte[] xliffData</td>
<td>The XLIFF file to upload.</td>
</tr>
</tbody>
</table>
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.
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**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String fileID</td>
<td>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.</td>
</tr>
<tr>
<td>byte[] reportDataChunk</td>
<td>The XML data to upload.</td>
</tr>
<tr>
<td>String reportRawDataChunk</td>
<td>String representation of XML data, presenting as reportRawDataChunk. This is an alternative to reportDataChunk byte[].</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
</tbody>
</table>
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**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte[] reportDataChunk</td>
<td>The XML data to upload.</td>
</tr>
<tr>
<td>String reportRawDataChunk</td>
<td>String representation of XML data, presenting as reportRawDataChunk. This is an alternative to reportDataChunk byte[].</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

4.29 uploadTemplateForReport() Method

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

**Signature**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>The path to the report to which to upload the template. For example: /HR Manager/Employee Reports/Employee Listing.xdo</td>
</tr>
<tr>
<td>String templateName</td>
<td>The file name of the template to upload.</td>
</tr>
</tbody>
</table>
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**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>The path to the report to which to upload the template. For example: /HR Manager /Employee Reports /Employee Listing.xdo</td>
</tr>
<tr>
<td>String templateFileName</td>
<td>The file name of the template to upload.</td>
</tr>
<tr>
<td>String templateName</td>
<td>The name of the template to upload.</td>
</tr>
<tr>
<td>String locale</td>
<td>The locale to assign to the template (for example, en_US).</td>
</tr>
<tr>
<td>byte[] templateData</td>
<td>The contents of the template file to upload.</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

**Table 4–31 Parameters for uploadTemplateForReport() Method (Cont.)**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String locale</td>
<td>The locale to assign to the template (for example, en_US).</td>
</tr>
<tr>
<td>byte[] templateData</td>
<td>The contents of the template file to upload.</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>
4.31 uploadXLIFFForReport() Method

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

**Signature**

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

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

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

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**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>The path to the report to which to upload the XLIFF. For example: /HR Manager/Employee Reports/Employee Listing.xdo</td>
</tr>
<tr>
<td>byte[] xliffData</td>
<td>The XLIFF file to upload.</td>
</tr>
</tbody>
</table>
Table 4-32 (Cont.) Parameters for `uploadXLIFFForReportInSession()` Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String <code>layoutFileName</code></td>
<td>The file name of the layout to which to associate the XLIFF file. For example: <code>employee_listing.rtf</code>.</td>
</tr>
<tr>
<td>String <code>locale</code></td>
<td>The locale to assign to the XLIFF (for example, <code>en_US</code>).</td>
</tr>
<tr>
<td>String <code>bipSessionToken</code></td>
<td>The proprietary token string generated for the user by the BI Publisher server. With the <code>bipSessionToken</code> 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.</td>
</tr>
</tbody>
</table>
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:

- Section 5.1, "assignRolesToUser() Method"
- Section 5.2, "createRole() Method"
- Section 5.3, "createUser() Method"
- Section 5.4, "deleteRole() Method"
- Section 5.5, "deleteUser() Method"
- Section 5.6, "getBIPHTTPSessionInterval() Method"
- Section 5.7, "getObjectSecurityXML() Method"
- Section 5.8, "getSecurityModel() Method"
- Section 5.9, "hasObjectAccess() Method"
- Section 5.10, "hasObjectAccessInSession() Method"
- Section 5.11, "impersonate() Method"
- Section 5.12, "isUserExists() Method"
- Section 5.13, "login() Method"
- Section 5.14, "logout() Method"
- Section 5.15, "notifyBIEEPReferencesUpdated() Method"
- Section 5.16, "notifyBIEEPReferencesUpdatedWithString() Method"
- Section 5.17, "removeRolesFromUser() Method"
- Section 5.18, "updateRole() Method"
- Section 5.19, "updateUser() Method"
- Section 5.20, "validateLogin() Method"

**Note:** For information on debugging applications built with BI Publisher Web services, see Section 1.4, "Debugging Web Service Applications."
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String userName</td>
<td>The user to which to add the role or roles.</td>
</tr>
<tr>
<td>String[] roleNames</td>
<td>The name of the role to add to the user. For example, &quot;Financial Users&quot;.</td>
</tr>
<tr>
<td>String adminUser</td>
<td>Specifies a BI Publisher user name for a user with administration privileges.</td>
</tr>
<tr>
<td>String adminPassword</td>
<td>Specifies the password for the administration user name.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String roleName</td>
<td>The name of the role to create. For example, &quot;Financial Users&quot;.</td>
</tr>
<tr>
<td>String description</td>
<td>The description of the role.</td>
</tr>
<tr>
<td>String adminUser</td>
<td>Specifies a BI Publisher user name for a user with administration privileges.</td>
</tr>
<tr>
<td>String adminPassword</td>
<td>Specifies the password for the administration user name.</td>
</tr>
</tbody>
</table>

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);
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**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String roleName</td>
<td>The user name to delete.</td>
</tr>
<tr>
<td>String adminUser</td>
<td>Specifies a BI Publisher user name for a user with administration privileges.</td>
</tr>
<tr>
<td>String adminPassword</td>
<td>Specifies the password for the administration user name.</td>
</tr>
</tbody>
</table>

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**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String userName</td>
<td>The user name to delete.</td>
</tr>
<tr>
<td>String adminUser</td>
<td>Specifies a BI Publisher user name for a user with administration privileges.</td>
</tr>
<tr>
<td>String adminPassword</td>
<td>Specifies the password for the administration user name.</td>
</tr>
</tbody>
</table>

5.6 getBIPHTTPSessionInterval() Method

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

**Signature**

```java
int getBIPHTTPSessionInterval(void);
```
5.7 `getObjectSecurityXML()` Method

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

**Signature**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String adminUsername</td>
<td>The user name for a BI Publisher user with administrator privileges.</td>
</tr>
<tr>
<td>String adminPassword</td>
<td>The password associated with the adminUserName.</td>
</tr>
<tr>
<td>String objectAbsolutePath</td>
<td>The path to the catalog object for which to retrieve the permissions description.</td>
</tr>
<tr>
<td>boolean isRecursive</td>
<td>Whether or not objectAbsolutePath is recursive.</td>
</tr>
</tbody>
</table>

5.8 `getSecurityModel()` Method

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

**Signature**

```java
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**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>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</td>
</tr>
<tr>
<td>String roleName</td>
<td>For future use. Ignore this parameter as it is not yet functional.</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

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**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>The absolute path to the report object.</td>
</tr>
<tr>
<td>String roleName</td>
<td>The role associated with the given user.</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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 given username.

**Signature**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String adminUserName</td>
<td>Specifies a BI Publisher user name for a user with administration privileges</td>
</tr>
<tr>
<td>String adminPassword</td>
<td>Specifies the password for the administration user name.</td>
</tr>
<tr>
<td>String username</td>
<td>The username of the user account that will be granted administrator privileges.</td>
</tr>
</tbody>
</table>

**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**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String userName</td>
<td>The user name to test.</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String bipSessionToken</td>
<td>The BI Publisher session ID.</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String bieeSessionID</td>
<td>The session ID for Oracle Business Intelligence/BI Publisher integration.</td>
</tr>
</tbody>
</table>

5.16 notifyBIEEPreferencesUpdatedWithString() Method

This method is provided for BIEE user preference integration purpose only.
**updateUser() Method**

```java
boolean notifyBIEEPreferencesUpdatedWithString(String bieeSessionID, String userPrefesXML);
```

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String bieeSessionID</td>
<td>The session ID for Oracle Business Intelligence/BI Publisher integration.</td>
</tr>
<tr>
<td>String userPrefesXML</td>
<td>The XML data that contain user preferences.</td>
</tr>
</tbody>
</table>

**removeRolesFromUser() Method**

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

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String userName</td>
<td>The user from which to delete the role or roles.</td>
</tr>
<tr>
<td>String[] roleNames</td>
<td>The name of the role to delete from the user. For example, &quot;Financial Users&quot;.</td>
</tr>
<tr>
<td>String adminUser</td>
<td>Specifies a BI Publisher user name for a user with administration privileges.</td>
</tr>
<tr>
<td>String adminPassword</td>
<td>Specifies the password for the administration user name.</td>
</tr>
</tbody>
</table>

**updateRole() Method**

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

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String currentRoleName</td>
<td>The name of the role to update.</td>
</tr>
<tr>
<td>String newDescription</td>
<td>The updated description of the role to apply.</td>
</tr>
<tr>
<td>String adminUser</td>
<td>Specifies a BI Publisher user name for a user with administration privileges.</td>
</tr>
<tr>
<td>String adminPassword</td>
<td>Specifies the password for the administration user name.</td>
</tr>
</tbody>
</table>

**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.
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**

```java
boolean validateLogin(String userID, String password);
```

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

**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**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String currentUserName</td>
<td>The user name to update.</td>
</tr>
<tr>
<td>String newPassword</td>
<td>The new password to assign to the user name.</td>
</tr>
<tr>
<td>String adminUser</td>
<td>Specifies a BI Publisher user name for a user with administration privileges.</td>
</tr>
<tr>
<td>String adminPassword</td>
<td>Specifies the password for the administration user name.</td>
</tr>
</tbody>
</table>
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 6.1, "copyObject() Method"
- Section 6.2, "copyObjectInSession() Method"
- Section 6.3, "createFolder() Method"
- Section 6.4, "createFolderInSession() Method"
- Section 6.5, "createObject() Method"
- Section 6.6, "createObjectInSession() Method"
- Section 6.7, "deleteObject() Method"
- Section 6.8, "deleteObjectInSession() Method"
- Section 6.9, "downloadObject() Method"
- Section 6.10, "downloadObjectInSession() Method"
- Section 6.11, "downloadXLIFF() Method"
- Section 6.12, "downloadXLIFFInSession() Method"
- Section 6.13, "getFolderContents() Method"
- Section 6.14, "getFolderContentsInSession() Method"
- Section 6.15, "getObject() Method"
- Section 6.16, "getObjectInfo() Method"
- Section 6.17, "getObjectInfoInSession() Method"
- Section 6.18, "getObjectInSession() Method"
- Section 6.19, "objectExist() Method"
- Section 6.20, "objectExistInSession() Method"
- Section 6.21, "renameObject() Method"
- Section 6.22, "renameObjectInSession() Method"
- Section 6.23, "updateObject() Method"
- Section 6.24, "updateObjectInSession() Method"
6.1 copyObject() Method

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

**Signature**

```java
boolean copyObject(String srcObjectAbsolutePath, String destObjectAbsolutePath, String newName, String userID, String password);
```

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String srcObjectAbsolutePath</td>
<td>The path to the catalog object to copy.</td>
</tr>
<tr>
<td>String destObjectAbsolutePath</td>
<td>The path to the location in the catalog to which to copy the object.</td>
</tr>
<tr>
<td>String newName</td>
<td>The name to assign the new object.</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies a BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

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**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String srcObjectAbsolutePath</td>
<td>The path to the catalog object to copy.</td>
</tr>
<tr>
<td>String destObjectAbsolutePath</td>
<td>The path to the location in the catalog to which to copy the object.</td>
</tr>
<tr>
<td>String newName</td>
<td>The name to assign the new object.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String folderAbsolutePath</td>
<td>The path to the folder that you want to create. For example: /HR Manager/Employee Reports/</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies a BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String folderAbsolutePath</td>
<td>The path to the folder that you want to create. For example: /HR Manager/Employee Reports/</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

6.5 createObject() Method

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

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String folderAbsolutePathURL</td>
<td>The absolute path to the folder in the catalog in which to place the new object.</td>
</tr>
<tr>
<td>String objectName</td>
<td>The name of the new object.</td>
</tr>
</tbody>
</table>
| String objectType     | The type of catalog object. Valid values are:  
  - `xdm` (data model)  
  - `xdo` (report)  
  - `xsb` (sub-template)  
  - `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**

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

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

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

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

**Signature**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportObjectAbsolutePath</td>
<td>The path to the object in the catalog to delete.</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies a BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

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**

```java
boolean deleteObjectInSession(String objectAbsolutePath, String bipSessionToken);
```

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String objectAbsolutePath</td>
<td>The path to the object in the catalog to delete.</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

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**

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

```java
cbyte[] downloadObjectInSession(String reportAbsolutePath, String bipSessionToken);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportAbsolutePath</td>
<td>The path to the object in the catalog to download.</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

downloadXLIFF() Method

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

**Signature**

```java
cbyte[] downloadXLIFF(String objectAbsolutePath, String userID, String password);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String objectAbsolutePath</td>
<td>The path to the XLIFF object to download.</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String objectAbsolutePath</td>
<td>The path to the XLIFF object to download.</td>
</tr>
<tr>
<td>String locale</td>
<td>The locale of the XLIFF object (for example, en_US).</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String folderAbsolutePath</td>
<td>The path to the folder for which to retrieve the contents. For example: / HR Manager/Employee Reports/</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies a BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

### 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);
```
### 6.15 `getObject()` Method

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

**Signature**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>reportObjectAbsolutePath</code></td>
<td>The path to the object to download. For example: <code>/HR Manager/Employee Reports/</code></td>
</tr>
<tr>
<td><code>locale</code></td>
<td>The locale of the object to get.</td>
</tr>
<tr>
<td><code>userID</code></td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td><code>password</code></td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>reportObjectAbsolutePath</code></td>
<td>The path to the object to download. For example: <code>/HR Manager/Employee Reports/</code></td>
</tr>
<tr>
<td><code>locale</code></td>
<td>The locale of the object to get.</td>
</tr>
<tr>
<td><code>userID</code></td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td><code>password</code></td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

### 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**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>reportObjectAbsolutePath</code></td>
<td>The path to the report object about which to get information. For example: <code>/HR Manager/Employee Reports/</code></td>
</tr>
<tr>
<td><code>userID</code></td>
<td>Specifies a BI Publisher user name.</td>
</tr>
<tr>
<td><code>password</code></td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

### 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**

```java
CatalogObjectInfo getObjectInSession(String objectAbsolutePath, String bipSessionToken);
```

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportObjectAbsolutePath</td>
<td>The path to the report object about which to get information.</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

## 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**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportObjectAbsolutePath</td>
<td>The path to the object to retrieve.</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

## 6.19  objectExist() Method

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

**Signature**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportObjectAbsolutePath</td>
<td>The path to the object to test for in the catalog. For example: /HR Manager/Employee Reports/Employee Data Model.xdm</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies a BI Publisher user name.</td>
</tr>
</tbody>
</table>
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**

```java
boolean objectExistInSession(String reportObjectAbsolutePath, String bipSessionToken);
```

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportObjectAbsolutePath</td>
<td>The path to the object to test for in the catalog. For example: /HR Manager/Employee Reports/Employee Data Model.xdm</td>
</tr>
</tbody>
</table>
| 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**

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportObjectAbsolutePath</td>
<td>The path to the object in the catalog to rename.</td>
</tr>
<tr>
<td>String newName</td>
<td>The new name to assign the object.</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies a BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

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."
updateObjectInSession() Method

Signature

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

Table 6–22 Parameters for renameObjectInSession() Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportObjectAbsolutePath</td>
<td>The path to the object in the catalog to rename.</td>
</tr>
<tr>
<td>String newName</td>
<td>The new name for the object.</td>
</tr>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportObjectAbsolutePath</td>
<td>The path to the object to update in the catalog. For example: /HR Manager/Employee Reports/Employee Data Model.xdm</td>
</tr>
<tr>
<td>byte[] objectData</td>
<td>The data with which to update the object.</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies a BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String reportObjectAbsolutePath</td>
<td>The path to the object to update in the catalog. For example: /HR Manager/Employee Reports/Employee Data Model.xdm</td>
</tr>
<tr>
<td>byte[] objectData</td>
<td>The data with which to update the object.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reportObjectAbsolutePathURL</td>
<td>The path to the object in the catalog.</td>
</tr>
</tbody>
</table>
| objectType              | The type of object to upload. Valid values are: 
  xdm (data model) 
  xdo (report) 
  xsb (sub-template) 
  xss (style template) |
| objectZippedData        | The object to upload in zipped format. |
| userID                  | Specifies a BI Publisher user name. |
| password                | Specifies the password for the user name. |

6.26 uploadObjectInSession() Method

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

Table 6–26 Parameters for uploadObjectInSession() Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reportObjectAbsolutePathURL</td>
<td>The path to the object in the catalog.</td>
</tr>
</tbody>
</table>
**uploadXLIFFMethod**

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

**Signature**

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String objectAbsolutePath</td>
<td>The path to the XLIFF object to upload.</td>
</tr>
<tr>
<td>byte[] xliffData</td>
<td>The XLIFF fie to upload.</td>
</tr>
<tr>
<td>String locale</td>
<td>The locale to assign to the XLIFF (for example, en_US).</td>
</tr>
<tr>
<td>String userID</td>
<td>Specifies the BI Publisher user name.</td>
</tr>
<tr>
<td>String password</td>
<td>Specifies the password for the user name.</td>
</tr>
</tbody>
</table>

**uploadXLIFFInSessionMethod**

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**

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String objectAbsolutePath</td>
<td>The path to the XLIFF object to upload.</td>
</tr>
<tr>
<td>byte[] xliffData</td>
<td>The XLIFF fie to upload.</td>
</tr>
<tr>
<td>String locale</td>
<td>The locale to assign to the XLIFF (for example, en_US).</td>
</tr>
</tbody>
</table>
### Table 6–28 (Cont.) Parameters for uploadXLIFFInSession() Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String bipSessionToken</td>
<td>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.</td>
</tr>
</tbody>
</table>
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:

- Chapter 7, "Using the BI Publisher Java APIs"
- Chapter 8, "Using the Delivery Manager Java APIs"
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:

- Section 7.1, "BI Publisher Core APIs"
- Section 7.2, "Prerequisites"
- Section 7.3, "Obtaining the Libraries"
- Section 7.4, "PDF Form Processing Engine"
- Section 7.5, "RTF Processor Engine"
- Section 7.6, "FO Processor Engine"
- Section 7.7, "PDF Document Merger"
- Section 7.8, "PDF Bookbinder Processor"
- Section 7.9, "PDF Digital Signature Engine"
- Section 7.10, "eText Processor"
- Section 7.11, "Document Processor Engine"
- Section 7.12, "BI Publisher Properties"
- Section 7.13, "Advanced Barcode Formatting"

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
Prerequisites

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](image)

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

```java
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

```java
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

```java
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

```java
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
<?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>Supplier</value>
  </field>
  ...
</fields>
```

The following code example shows how you can use the API:

Example 7–6  Sample Code for Retrieving Internal XFDF Documents

```java
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:

```xml
<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:

```xml
<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

```xml
<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

```java
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**

```java
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**

```java
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**

```java
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**

```java
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.
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
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**

```java
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);  // set output format
    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**

```java
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.setOutputStream(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:
Example 7–15  Sample Code for Generating Output from XSL Template Arrays and XML Data

```java
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");
    processor.setOutputFormat(FOProcessor.FORMAT_PDF);
    processor.process();
    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

```java
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

```java
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

```java
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

```java
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

```java
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

```java
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

```java
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)

Output:

- PDF Document (OutputStream)
Example 7–24 Sample Code for Merging PDF Documents with Background to Place Page Numbering

```java
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

```java
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**

```java
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
cDocMerger.setTextWatermark("DRAFT", 200f, 200f); // set text and place
cDocMerger.setTextWatermarkAngle(80);              // set angle
cDocMerger.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:

```java
float[] rct = {LowerLeft X, LowerLeft Y, UpperRight X, UpperRight Y}
```

For example:

```java
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:

```java
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**

```java
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;
    }
}
```

---

**PDF Document Merger**

Using the BI Publisher Java APIs  7-25
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:
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**

```
```

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.
7.8.4 API Method Call

The following is an example of an API method call:

Example 7–29 Sample API Method Call

```java
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 (PKCS-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

```java
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

```java
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

```java
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>
<thead>
<tr>
<th>Element</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestset</td>
<td>xmlns</td>
<td>Root element must contain [xmlns:xapi=&quot;<a href="http://xmlns.oracle.com/oxp/xapi">http://xmlns.oracle.com/oxp/xapi</a>&quot;] block. The version is not required, but defaults to &quot;1.0&quot;.</td>
</tr>
<tr>
<td></td>
<td>version</td>
<td></td>
</tr>
<tr>
<td>request</td>
<td>N/A</td>
<td>Element that contains the data and template processing definitions.</td>
</tr>
<tr>
<td>delivery</td>
<td>N/A</td>
<td>Defines where the generated output is sent.</td>
</tr>
<tr>
<td>document</td>
<td>output-type</td>
<td>Specify one output that can have several template elements. The output-type attribute is optional. Valid values are: pdf (Default) rtf html excel text</td>
</tr>
<tr>
<td>filesystem</td>
<td>output</td>
<td>Specify this element to save the output to the file system. Define the directory path in the output attribute.</td>
</tr>
<tr>
<td>print</td>
<td>■ printer</td>
<td>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: &quot;ipp://myprintserver:631/printers/printername&quot;</td>
</tr>
<tr>
<td></td>
<td>■ server-alias</td>
<td></td>
</tr>
<tr>
<td>fax</td>
<td>■ server</td>
<td>Specify a URI in the server attribute, for example: &quot;ipp://myfaxserver1:631/printers/myfaxmachine&quot;</td>
</tr>
<tr>
<td></td>
<td>■ server-alias</td>
<td></td>
</tr>
<tr>
<td>number</td>
<td></td>
<td>The number element can occur multiple times to list multiple fax numbers. Each element occurrence must contain only one number.</td>
</tr>
<tr>
<td>email</td>
<td>■ server</td>
<td>Specify the outgoing mail server (SMTP) in the server attribute.</td>
</tr>
<tr>
<td></td>
<td>■ port</td>
<td>Specify the mail server port in the port attribute.</td>
</tr>
<tr>
<td></td>
<td>■ from</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ reply-to</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ server-alias</td>
<td></td>
</tr>
<tr>
<td>Element</td>
<td>Attributes</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>message</td>
<td>■ to ● cc ● bcc ● attachment ● subject</td>
<td>The message element can be placed several times under the email element. You can specify character data in the message element. You can specify multiple e-mail addresses in the to, cc, and bcc attributes separated by a comma. 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. The subject attribute is optional.</td>
</tr>
<tr>
<td>background</td>
<td>where</td>
<td>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.</td>
</tr>
<tr>
<td>text</td>
<td>■ title ● default</td>
<td>Specify the watermark text in the title value. A default value of &quot;yes&quot; automatically draws the watermark with forward slash type. The default value is yes.</td>
</tr>
<tr>
<td>pagonenumber</td>
<td>■ initial-page-index ● initial-value ● x-pos ● y-pos</td>
<td>The initial-page-index default value is 0. The initial-value default value is 1. &quot;Helvetica&quot; is used for the page number font. The x-pos provides lower left x position. The y-pos provides lower left y position.</td>
</tr>
<tr>
<td>template</td>
<td>■ locale ● location ● type</td>
<td>Contains template information. Valid values for the type attribute are pdf rtf xsl-fo etext The default value is &quot;pdf&quot;.</td>
</tr>
</tbody>
</table>
### 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
<?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>
```

---

**Table 7–1 (Cont.) Elements in Document Processor XML File Hierarchy**

<table>
<thead>
<tr>
<th>Element</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>location</td>
<td>Define the <code>location</code> attribute to specify the location of the data, or attach the actual XML data with sub-elements. The default value of <code>location</code> is &quot;inline&quot;. It the <code>location</code> points to either an XML file or a URL, then the data should contain an XML declaration with the proper encoding. If the <code>location</code> attribute is not specified, the <code>data</code> element should contain the sub-elements for the actual data. This must not include an XML declaration.</td>
</tr>
</tbody>
</table>

---
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
<?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
<?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 set of data</field1>
        </xapi:data>
      </xapi:template>

      <xapi:template type="pdf" location="http://your.server:9999/templates/template2.pdf">
        <xapi:data>
          <field1>The third set of 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
<?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
<?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
<?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: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>
```
7.11.3 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

```java
import oracle.xdo.batch.DocumentProcessor;
.
.
.
public static void main(String[] args)
{
.
.
.
    try
```
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

```java
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

```java
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-encryption-level", "0");
    processor.setConfig(prop);
    // Start processing
    try
```
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**

```java
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
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-encryption-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:

```java
/**
 * 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

```java
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);
    }
  } 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;
```java
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.
This chapter describes the BI Publisher delivery manager APIs. It includes the following sections:

- Section 8.1, "Using the Delivery Manager"
- Section 8.2, "Delivering Documents by E-Mail"
- Section 8.3, "Delivering Your Document to a Printer"
- Section 8.4, "Delivering Your Document to a Local Printer"
- Section 8.5, "Delivering Your Documents to a Fax Server"
- Section 8.6, "Delivering Your Documents to a RightFax Server"
- Section 8.7, "Delivering Your Documents to a WebDAV Server"
- Section 8.8, "Delivering Your Document over the File Transfer Protocol (FTP)"
- Section 8.9, "Delivering Your Documents over Secure FTP"
- Section 8.10, "Delivering Your Documents over Hypertext Transfer Protocol (HTTP)"
- Section 8.11, "Delivering Documents over AS2"
- Section 8.12, "Delivering Documents Using an External Command"
- Section 8.13, "Delivering Documents to the Local File System"
- Section 8.14, "Direct and Buffering Modes"
- Section 8.15, "Asynchronous Delivery Requests"
- Section 8.16, "Document Filter Support"
- Section 8.17, "Date Expression Support"
- Section 8.18, "Internationalization Support"
- Section 8.19, "Setting Global Properties"
- Section 8.20, "Adding a Custom Delivery Channel"
- Section 8.21, "Configuration File Support"

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**

```java
// 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(
```
Delivering Documents by E-Mail

Using the Delivery Manager Java APIs

```java
// set the sender email address
req.addProperty(DeliveryPropertyDefinitions.SMTP_FROM, "myname@example.com");
// set the destination email address
req.addProperty(DeliveryPropertyDefinitions.SMTP_TO_RECIPIENTS, "user1@example.com, user2@example.com");
// 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>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| SMTP_TO_RECIPIENTS              | Required
|                                 | Enter multiple recipients separated by a comma (example: user1@example.com, user2@example.com) |
| SMTP_CC_RECIPIENTS              | Optional
|                                 | Enter multiple recipients separated by a comma.                              |
| SMTP_BCC_RECIPIENTS             | Optional
|                                 | Enter multiple recipients separated by a comma.                              |
| SMTP_FROM                       | Required
|                                 | Enter the e-mail address of the sending party.                               |
| SMTP_REPLY_TO                   | Optional
|                                 | Enter the reply-to e-mail address.                                          |
| SMTP_SUBJECT                    | Required
|                                 | Enter the subject of the e-mail.                                            |
| SMTP_CHARACTER_ENCODING         | Optional
|                                 | Default is "UTF-8".                                                         |
| SMTP_ATTACHMENT                 | Optional
|                                 | If you are including an attachment, enter the attachment object name.       |
| SMTP_CONTENT_FILENAME           | Optional
|                                 | 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
|                                 | Enter the MIME type.                                                       |
### 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**

```java
// set the TO email addresses
req.addProperty(
    DeliveryPropertyDefinitions.SMTP_TO_RECIPIENTS,
    "user1@example.com, user2@example.com, user3@example.com";

// set the CC email addresses
req.addProperty(
    DeliveryPropertyDefinitions.SMTP_CC_RECIPIENTS,
    "user4@example.com, user5@example.com, user6@example.com";
```

### 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

```java
// create Attachment instance
Attachment m = new Attachment();

// add PDF attachment
m.addAttachment("/pdf_doc/invoice.pdf", "invoice.pdf", "application/pdf");

// add RTF attachment
m.addAttachment("/rtf_doc/product.rtf", "product.rtf", "application/rtf");

// add XML attachment
m.addAttachment("/xml_doc/data.xml", "data.xml", "text/xml");

// If you want to attach HTML documents, 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

```java
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@example.com");
// set the destination email address
req.addProperty(DeliveryPropertyDefinitions.SMTP_TO_RECIPIENTS, "user1@example.com,
user2@example.com" );
// 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", "invoice.pdf", "application/pdf");
// 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**

```java
// 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@example.com");
// set the destination email address
req.addProperty(DeliveryPropertyDefinitions.SMTP_TO_RECIPIENTS, "user1@example.com,
user2@example.com");

// 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**

```java
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/](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>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPP_HOST</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Enter the host name.</td>
</tr>
</tbody>
</table>
### Table 8–2 (Cont.) Properties for Delivering Documents to Printers

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPP_PORT</td>
<td>Optional&lt;br&gt;Default is 631.</td>
</tr>
<tr>
<td>IPP_PRINTER_NAME</td>
<td>Required&lt;br&gt;Enter the name of the printer that is to receive the output.</td>
</tr>
<tr>
<td></td>
<td>• If you use CUPS with the default setup, enter the printer name as /printers/&lt;printer-name&gt;</td>
</tr>
<tr>
<td></td>
<td>• If you use the Microsoft Internet Information Service (IIS) with the default setup, enter the printer name as /printers/&lt;printer-name&gt;/printer</td>
</tr>
<tr>
<td>IPP_AUTHTYPE</td>
<td>Optional&lt;br&gt;Valid values for authentication type are:</td>
</tr>
<tr>
<td></td>
<td>IPP_AUTHTYPE_NONE - no authentication (default)&lt;br&gt;IPP_AUTHTYPE_BASIC - use HTTP basic authentication&lt;br&gt;IPP_AUTHTYPE_DIGEST - use HTTP digest authentication</td>
</tr>
<tr>
<td>IPP_USERNAME</td>
<td>Optional&lt;br&gt;Enter the username for HTTP authentication.</td>
</tr>
<tr>
<td>IPP_PASSWORD</td>
<td>Optional&lt;br&gt;Enter the password for HTTP authentication.</td>
</tr>
<tr>
<td>IPP_ENCTYPE</td>
<td>Optional&lt;br&gt;The encryption type can be set to either of the following:&lt;br&gt;IPP_ENCTYPE_NONE - no encryption (default)&lt;br&gt;IPP_ENCTYPE_SSL - use Secure Socket Layer</td>
</tr>
<tr>
<td>IPP_USE_FULL_URL</td>
<td>Optional&lt;br&gt;Set to “true” to send the full URL for the HTTP request header.&lt;br&gt;Valid values are “true” or “false” (default).</td>
</tr>
<tr>
<td>IPP_USE_CHUNKED_BODY</td>
<td>Optional&lt;br&gt;Valid values are “true” (default) to use HTTP chunked transfer coding for the message body, or “false”.</td>
</tr>
<tr>
<td>IPP_ATTRIBUTE_CHARSET</td>
<td>Optional&lt;br&gt;Attribute character set of the IPP request. Default is &quot;UTF-8&quot;.</td>
</tr>
<tr>
<td>IPP_NATURAL_LANGUAGE</td>
<td>Optional&lt;br&gt;The natural language of the IPP request. Default is &quot;en&quot;.</td>
</tr>
<tr>
<td>IPP_JOB_NAME</td>
<td>Optional&lt;br&gt;Job name of the IPP request.</td>
</tr>
<tr>
<td>IPP_COPIES</td>
<td>Optional&lt;br&gt;Define the number of copies to print (example: &quot;1&quot;, &quot;5&quot;, &quot;10&quot;).&lt;br&gt;Default is 1.</td>
</tr>
</tbody>
</table>
Table 8–2  (Cont.) Properties for Delivering Documents to Printers

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPP_SIDES</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Enable two-sided printing. This setting will be ignored if the target printer does not support two-sided printing. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>■ IPP_SIDES_ONE_SIDED - default</td>
</tr>
<tr>
<td></td>
<td>■ IPP_SIDES_TWO_SIDED_LONG_EDGE - prints both sides of paper for binding long edge.</td>
</tr>
<tr>
<td></td>
<td>■ IPP_SIDES_TWO_SIDED_SHORT_EDGE - prints both sides of paper for binding short edge.</td>
</tr>
<tr>
<td></td>
<td>■ IPP_SIDES_DUPLEX: Same as IPP_SIDES_TWO_SIDED_LONG_EDGE.</td>
</tr>
<tr>
<td></td>
<td>■ IPP_SIDES_TUMBLE: Same as IPP_SIDES_TWO_SIDED_SHORT_EDGE.</td>
</tr>
<tr>
<td>IPP_ORIENTATIONS</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Sets the paper orientation. This setting will be ignored if the target printer does not support orientation settings. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>■ IPP_ORIENTATIONSPORTRAIT (default)</td>
</tr>
<tr>
<td></td>
<td>■ IPP_ORIENTATIONSLANDSCAPE</td>
</tr>
<tr>
<td>IPP_DOCUMENT_FORMAT</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>The target printer must support the specified format. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>■ IPP_DOCUMENT_FORMAT_POSTSCRIPT</td>
</tr>
<tr>
<td></td>
<td>■ IPP_DOCUMENT_FORMATPLAINTEXT</td>
</tr>
<tr>
<td></td>
<td>■ IPP_DOCUMENT_FORMATPDF</td>
</tr>
<tr>
<td></td>
<td>■ IPP_DOCUMENT_FORMATOCTETSTREAM (default)</td>
</tr>
<tr>
<td>IPP_MEDIA</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>■ IPP_MEDIA_TRAY1: Media on tray 1</td>
</tr>
<tr>
<td></td>
<td>■ IPP_MEDIA_TRAY2: Media on tray 2</td>
</tr>
<tr>
<td></td>
<td>■ IPP_MEDIA_TRAY3: Media on tray 3</td>
</tr>
<tr>
<td></td>
<td>■ IPP_MEDIA_A3: A3 Media</td>
</tr>
<tr>
<td></td>
<td>■ IPP_MEDIA_A4: A4 Media</td>
</tr>
<tr>
<td></td>
<td>■ IPP_MEDIA_A5: A5 Media</td>
</tr>
<tr>
<td></td>
<td>■ IPP_MEDIA_B4: B4 Media</td>
</tr>
<tr>
<td></td>
<td>■ IPP_MEDIA_B5: B5 Media</td>
</tr>
<tr>
<td>IPP_PAGE_RANGES</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Specify page ranges to print. By default, all pages are printed. Example valid values are:</td>
</tr>
<tr>
<td></td>
<td>■ &quot;3&quot;: prints only page 3.</td>
</tr>
<tr>
<td></td>
<td>■ &quot;2-5&quot;: prints pages 2-5.</td>
</tr>
<tr>
<td></td>
<td>■ &quot;1,3-5&quot;: print page 1 and 3-5.</td>
</tr>
</tbody>
</table>
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

```java
// 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 content 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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRINTER_CONTENT_TYPE</td>
<td>Optional&lt;br&gt;The document content type (example: &quot;application/pdf&quot;).</td>
</tr>
<tr>
<td>PRINTER_COPIES</td>
<td>Optional&lt;br&gt;Specify the number of copies to print (example: &quot;1&quot;, &quot;5&quot;, &quot;10&quot;). Default is 1.</td>
</tr>
<tr>
<td>PRINTER_HOST</td>
<td>Optional&lt;br&gt;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.</td>
</tr>
<tr>
<td>PRINTER_MEDIA</td>
<td>Optional&lt;br&gt;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.&lt;br&gt;Valid values are:&lt;br&gt;- PRINTER_MEDIA_TRAY1: Media on tray 1&lt;br&gt;- PRINTER_MEDIA_TRAY2: Media on tray 2&lt;br&gt;- PRINTER_MEDIA_TRAY3: Media on tray 3&lt;br&gt;- PRINTER_MEDIA_A3: A3 Media&lt;br&gt;- PRINTER_MEDIA_A4: A4 Media&lt;br&gt;- PRINTER_MEDIA_A5: A5 Media&lt;br&gt;- PRINTER_MEDIA_B4: B4 Media&lt;br&gt;- PRINTER_MEDIA_B5: B5 Media</td>
</tr>
<tr>
<td>PRINTER_ORIENTATIONS</td>
<td>Optional&lt;br&gt;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</td>
</tr>
<tr>
<td>PRINTER_PAGE_RANGES</td>
<td>Specify page ranges to print. By default, all pages are printed.&lt;br&gt;Example valid values are:&lt;br&gt;- &quot;3&quot;: prints only page 3.&lt;br&gt;- &quot;2-5&quot;: prints pages 2-5.&lt;br&gt;- &quot;1,3-5&quot;: print page 1 and 3-5.</td>
</tr>
<tr>
<td>PRINTER_SIDES</td>
<td>Optional&lt;br&gt;Enable two-sided printing. This setting will be ignored if the target printer does not support two-sided printing. Valid values are:&lt;br&gt;- PRINTER_SIDES_ONE_SIDED - default&lt;br&gt;- PRINTER_SIDES_TWO_SIDED_LONG_EDGE - prints both sides of paper for binding long edge.&lt;br&gt;- PRINTER_SIDES_TWO_SIDED_SHORT_EDGE - prints both sides of paper for binding short edge.&lt;br&gt;- PRINTER_SIDES_DUPLEX: Same as PRINTER_SIDES_TWO_SIDED_LONG_EDGE.&lt;br&gt;- PRINTER_SIDES_TUMBLE: Same as PRINTER_SIDES_TWO_SIDED_SHORT_EDGE.</td>
</tr>
</tbody>
</table>
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**

```java
// 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>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPP_PHONE_NUMBER</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Enter the fax number.</td>
</tr>
</tbody>
</table>

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**

```java
// 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");
// destination
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>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIGHTFAX_HTTP_HOST</td>
<td>Required HTTP host of the RightFax server</td>
</tr>
<tr>
<td>RIGHTFAX_HTTP_PORT</td>
<td>Optional HTTP port of the RightFax server. Default=80.</td>
</tr>
<tr>
<td>RIGHTFAX_HTTP_REMOTE_DIRECTORY</td>
<td>Optional Enter the remote directory name (example: /RFWebCon.dll) of the RightFax XML interface.</td>
</tr>
<tr>
<td>RIGHTFAX_HTTP_AUTHTYPE</td>
<td>Optional 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_HTTP_AUTHTYPE_NONE.</td>
</tr>
<tr>
<td>RIGHTFAX_HTTP_USERNAME</td>
<td>Optional HTTP username for the RightFax server url. Required when RIGHTFAX_HTTP_AUTH_TYPE is set to values other than RIGHTFAX_HTTP_AUTHTYPE_NONE.</td>
</tr>
<tr>
<td>RIGHTFAX_HTTP_PASSWORD</td>
<td>Optional HTTP password for the RightFax server url. Required when RIGHTFAX_HTTP_AUTH_TYPE is set to values other than RIGHTFAX_HTTP_AUTHTYPE_NONE.</td>
</tr>
<tr>
<td>RIGHTFAX_HTTP_ENCTYPE</td>
<td>Optional The encryption type can be set to either of the following: RIGHTFAX_HTTP_ENCTYPE_NONE - no encryption (default) RIGHTFAX_HTTP_ENCTYPE_SSL - use Secure Socket Layer</td>
</tr>
</tbody>
</table>
### Table 8–5 (Cont.) Properties for Delivering Documents to RightFax Servers

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| RIGHTFAX_HTTP_USE_FULL_URL       | Optional  
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  
Valid values are "true" (default) to use HTTP chunked transfer coding for the message body, or "false". |
| RIGHTFAX_HTTP_TIMEOUT            | Optional  
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  
Enter the proxy server host name. |
| RIGHTFAX_HTTP_PROXY_PORT         | Optional  
Enter the proxy server port number. Default=80. |
| RIGHTFAX_HTTP_PROXY_AUTHTYPE     | Optional  
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  
Enter the username for proxy authentication. |
| RIGHTFAX_HTTP_PROXY_PASSWORD     | Optional  
Enter the password for HTTP proxy authentication. |
| RIGHTFAX_SENDER_FROM_NAME        | Optional  
Enter the name of the sender. |
| RIGHTFAX_SENDER_EMP_ID           | Optional  
Enter the employee id of the sender. |
| RIGHTFAX_SENDER_FROM_COMPANY     | Optional  
Enter the name of the sender's company. |
| RIGHTFAX_SENDER_FROM_DEPARTMENT  | Optional  
Enter the name of the sender's department. |
| RIGHTFAX_SENDER_FROM_PHONE       | Optional  
Enter sender’s phone number. |
| RIGHTFAX_SENDER_RETURN_EMAIL     | Optional  
Enter sender’s return email address. |
| RIGHTFAX_SENDER_BILLINFO1        | Optional  
Enter the billing code of the fax owner. |
| RIGHTFAX_SENDER_BILLINFO2        | Optional  
Enter the secondary billing code of the fax owner. |
| RIGHTFAX_SENDER_RF_USER          | Required  
Enter the name of the sender's RightFax user name. |
The following is sample code for delivery to a Web-based Distributed Authoring and Versioning (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,
```

### Table 8–5  (Cont.) Properties for Delivering Documents to RightFax Servers

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIGHTFAX_FAX_TO_NUMBER</td>
<td>Required Enter the fax number where the document will be sent.</td>
</tr>
<tr>
<td>RIGHTFAX_FAX_TO_NAME</td>
<td>Optional Enter the recipient's name.</td>
</tr>
<tr>
<td>RIGHTFAX_FAX_TO_COMPANY</td>
<td>Optional Enter the recipient's company name.</td>
</tr>
<tr>
<td>RIGHTFAX_FAX_ALT_FAX_NUM</td>
<td>Optional Enter the alternative fax number.</td>
</tr>
<tr>
<td>RIGHTFAX_FAX_TO_CONTACTNUM</td>
<td>Optional Enter the contact phone number of the recipient.</td>
</tr>
<tr>
<td>RIGHTFAX_FAX_COVERSHEET</td>
<td>Optional 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.</td>
</tr>
<tr>
<td>RIGHTFAX_COVERTEXT</td>
<td>Optional Enter the text that should appear on the cover sheet.</td>
</tr>
<tr>
<td>RIGHTFAX_COVERTEXT_TYPE</td>
<td>Optional Enter the type of the cover sheet text. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>- TXT (default)</td>
</tr>
<tr>
<td></td>
<td>- RTF</td>
</tr>
<tr>
<td>RIGHTFAX_COVERTEXT_ENCODING</td>
<td>Optional Enter the encoding of the cover sheet text. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>- NONE (default)</td>
</tr>
<tr>
<td></td>
<td>- BASE64</td>
</tr>
<tr>
<td></td>
<td>- QUOTEDPRINTABLE</td>
</tr>
<tr>
<td>RIGHTFAX_DOCUMENT_FORMAT</td>
<td>Optional Valid values are:</td>
</tr>
<tr>
<td></td>
<td>- PDF (default)</td>
</tr>
<tr>
<td></td>
<td>- PS</td>
</tr>
<tr>
<td></td>
<td>- TEXT</td>
</tr>
</tbody>
</table>

### 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
```

```
// set document content type
req.addProperty(DeliveryPropertyDefinitions.WEBDAV_CONTENT_TYPE,
```
Delivering Your Documents to a WebDAV Server

*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>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WEBDAV_CONTENT_TYPE</td>
<td>Required. Enter the document content type (example: &quot;application/pdf&quot;).</td>
</tr>
<tr>
<td>WEBDAV_HOST</td>
<td>Required. Enter the server host name.</td>
</tr>
<tr>
<td>WEBDAV_PORT</td>
<td>Optional. Enter the server port number. Default is 80.</td>
</tr>
<tr>
<td>WEBDAV_REMOTE_DIRECTORY</td>
<td>Required. Enter the remote directory name (example: &quot;/myreports/&quot;.</td>
</tr>
<tr>
<td>WEBDAV_REMOTE_FILENAME</td>
<td>Required. Enter the remote file name.</td>
</tr>
<tr>
<td>WEBDAV_AUTHTYPE</td>
<td>Optional. Valid values for authentication type are:</td>
</tr>
<tr>
<td></td>
<td>WEBDAV_AUTHTYPE_NONE - no authentication (default)</td>
</tr>
<tr>
<td></td>
<td>WEBDAV_AUTHTYPE_BASIC - use HTTP basic authentication</td>
</tr>
<tr>
<td></td>
<td>WEBDAV_AUTHTYPE_DIGEST - use HTTP digest authentication</td>
</tr>
<tr>
<td>WEBDAV_USERNAME</td>
<td>Optional. Enter the username for HTTP authentication.</td>
</tr>
<tr>
<td>WEBDAV_PASSWORD</td>
<td>Optional. Enter the password for HTTP authentication.</td>
</tr>
</tbody>
</table>
**8.8 Delivering Your Document over the File Transfer Protocol (FTP)**

The following is sample code for delivery to an FTP server:

```java
// 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>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTP_HOST</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Enter the server host name.</td>
</tr>
</tbody>
</table>

**WEBDAV_ENCTYPE**

Optional

Valid values for encryption type are:

- WEBDAV_ENCTYPE_NONE - no encryption (default)
- WEBDAV_ENCTYPE_SSL - use Secure Socket Layer

**WEBDAV_USE_FULL_URL**

Optional

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

Valid values are "true" (default) to use HTTP chunked transfer coding for the message body, or "false".

**WEBDAV_URL_CHARACTERENCODING**

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.
Delivering Your Documents over Secure FTP

Using the Delivery Manager Java APIs

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 list the supported properties. A string value is required for each property unless otherwise noted.

Example 8–15 Sample Code for Delivering Documents over SFTP

```java
// 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 8–7 (Cont.) Properties for Delivering Documents over FTP

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTP_PORT</td>
<td>Optional&lt;br&gt;Enter the server port number. Default is 21.</td>
</tr>
<tr>
<td>FTP_USERNAME</td>
<td>Required&lt;br&gt;Enter the login user name to the FTP server.</td>
</tr>
<tr>
<td>FTP_PASSWORD</td>
<td>Required&lt;br&gt;Enter the login password to the FTP server.</td>
</tr>
<tr>
<td>FTP_REMOTE_DIRECTORY</td>
<td>Required&lt;br&gt;Enter the directory to which to deliver the document (example: /pub/). To deliver the document to the user’s home directory, enter &quot;.&quot;</td>
</tr>
<tr>
<td>FTP_REMOTE_FILENAME</td>
<td>Required&lt;br&gt;Enter the document file name for the remote server.</td>
</tr>
<tr>
<td>FTP_BINARY_MODE</td>
<td>Optional&lt;br&gt;Valid values are &quot;true&quot; (default) or &quot;false&quot;.</td>
</tr>
<tr>
<td>FTP_PASSIVE_MODE</td>
<td>Optional&lt;br&gt;Valid values are &quot;true&quot; or &quot;false&quot; (default).</td>
</tr>
</tbody>
</table>
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

```java
// 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");
```

---

**Table 8–8 Properties for Delivering Documents over SFTP**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFTP_HOST</td>
<td>Required Enter the target server host name.</td>
</tr>
<tr>
<td>SFTP_PORT</td>
<td>Optional Enter the target server SSH port number. Default is 22.</td>
</tr>
<tr>
<td>SFTP_USERNAME</td>
<td>Required Enter the login user name.</td>
</tr>
<tr>
<td>SFTP_PASSWORD</td>
<td>Required if you choose the SFTP_AUTH_TYPE_PASSWORD authentication type. Enter the login password.</td>
</tr>
<tr>
<td>SFTP_REMOTE_DIRECTORY</td>
<td>Required Enter the directory to which to deliver the document (example: <code>/pub/</code>). To deliver the document to the user’s home directory, enter <code>.</code>.</td>
</tr>
<tr>
<td>SFTP_REMOTE_FILENAME</td>
<td>Required Enter the document file name on the remote server.</td>
</tr>
<tr>
<td>SFTP_AUTH_TYPE</td>
<td>Set either of the following: SFTP_AUTH_TYPE_PASSWORD (Default) Requires providing password at login. SFTP_AUTH_TYPE_PUBLIC_KEY - public key authorization type.</td>
</tr>
<tr>
<td>SFTP_PRIVATE_KEY_FILE</td>
<td>Enter the client private key file. Required if you choose SFTP_AUTH_TYPE_PUBLIC_KEY.</td>
</tr>
<tr>
<td>SFTP_PRIVATE_KEY_PASSWORD</td>
<td>Enter the client private key password. Required if you choose SFTP_AUTH_TYPE_PUBLIC_KEY.</td>
</tr>
<tr>
<td>SFTP_FILE_PERMISSION</td>
<td>Enter the permissions to set for the file being created. Default is 0755.</td>
</tr>
</tbody>
</table>
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**

```java
// 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**

```java
// create delivery manager instance
DeliveryManager dm = new DeliveryManager();
// create a delivery request
DeliveryRequest req = dm.createRequest(DeliveryManager.TYPE_HTTP);

// set request method
req.addProperties(DeliveryPropertyDefinitions.HTTP_METHOD, 
                 DeliveryPropertyDefinitions.HTTP_METHOD_POST);
// set document content type
req.addProperties(DeliveryPropertyDefinitions.HTTP_CONTENT_TYPE, 
                 "application/pdf");
// set the HTTP server hostname
req.addProperties(DeliveryPropertyDefinitions.HTTP_HOST, "myhost");
// set the HTTP server port number
req.addProperties(DeliveryPropertyDefinitions.HTTP_PORT, "80");
// set the target remote directory
req.addProperties(DeliveryPropertyDefinitions.HTTP_REMOTE_DIRECTORY, 
                 "/servlet/");
// set the remote filename (servlet class)
req.addProperties(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>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_METHOD</td>
<td>Optional&lt;br&gt;Sets the HTTP request method. Valid values are:&lt;br&gt;HTTP_METHOD_POST (Default)&lt;br&gt;HTTP_METHOD_PUT</td>
</tr>
<tr>
<td>HTTP_CONTENT_TYPE</td>
<td>Optional&lt;br&gt;The document content type (example: &quot;application/pdf&quot;).</td>
</tr>
<tr>
<td>HTTP_HOST</td>
<td>Required&lt;br&gt;Enter the server host name.</td>
</tr>
<tr>
<td>HTTP_PORT</td>
<td>Optional&lt;br&gt;Enter the server port number. The default is 80.</td>
</tr>
<tr>
<td>HTTP_REMOTE_DIRECTORY</td>
<td>Required&lt;br&gt;Enter the remote directory name (example: &quot;/home/&quot;).</td>
</tr>
<tr>
<td>HTTP_REMOTE_FILENAME</td>
<td>Required&lt;br&gt;Enter the file name to save the document as in the remote directory.</td>
</tr>
<tr>
<td>HTTP_AUTHTYPE</td>
<td>Optional&lt;br&gt;Valid values for authentication type are:&lt;br&gt;HTTP_AUTHTYPE_NONE - no authentication (default)&lt;br&gt;HTTP_AUTHTYPE_BASIC - use basic HTTP authentication&lt;br&gt;HTTP_AUTHTYPE_DIGEST - use digest HTTP authentication</td>
</tr>
<tr>
<td>HTTP_USERNAME</td>
<td>Optional&lt;br&gt;If the server requires authentication, enter the username.</td>
</tr>
<tr>
<td>HTTP_PASSWORD</td>
<td>Optional&lt;br&gt;If the server requires authentication, enter the password for the username.</td>
</tr>
</tbody>
</table>
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**

```java
// 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");
```

### Properties for Delivering Documents over HTTP

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_ENCTYPE</td>
<td>Optional&lt;br&gt;Enter the encryption type:&lt;br&gt;HTTP_ENCTYPE_NONE - no encryption (default)&lt;br&gt;HTTP_ENCTYPE_SSL - use Secure Socket Layer</td>
</tr>
<tr>
<td>HTTP_USE_FULL_URL</td>
<td>Optional&lt;br&gt;Set to “true” to send the full URL for the HTTP request header. Valid values are “true” or “false” (default).</td>
</tr>
<tr>
<td>HTTP_USE_CHUNKED_BODY</td>
<td>Optional&lt;br&gt;Valid values are “true” (default) to use HTTP chunked transfer coding for the message body, or “false”.</td>
</tr>
<tr>
<td>HTTP_TIMEOUT</td>
<td>Optional&lt;br&gt;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).</td>
</tr>
<tr>
<td>HTTP_URL_CHARACTER_ENCODING</td>
<td>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.</td>
</tr>
</tbody>
</table>

---

### 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:
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**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS2_FROM</td>
<td>Required. Enter the AS2 message sender.</td>
</tr>
<tr>
<td>AS2_TO</td>
<td>Required. Enter the AS2 message recipient.</td>
</tr>
<tr>
<td>AS2_SUBJECT</td>
<td>Required. Enter the message subject.</td>
</tr>
<tr>
<td>AS2_MESSAGE_COMPRESSION</td>
<td>Default value is False. Enter True to compress the message.</td>
</tr>
<tr>
<td>AS2_MESSAGE_SIGNATURE</td>
<td>Default value is False. Enter True to sign the message.</td>
</tr>
<tr>
<td>AS2_MESSAGE_ENCRYPTION</td>
<td>Default value is False. Enter True to encrypt the message.</td>
</tr>
<tr>
<td>AS2_CONTENT_TYPE</td>
<td>Required. Enter the content type of the document. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>* application/EDIFACT</td>
</tr>
<tr>
<td></td>
<td>* application/xml</td>
</tr>
<tr>
<td>AS2_ENC_ALGO</td>
<td>The AS2 encryption algorithm. Set one of the following:</td>
</tr>
<tr>
<td></td>
<td>* AS2_ENC_ALGO_RC2_40</td>
</tr>
<tr>
<td></td>
<td>* AS2_ENC_ALGO_RC2_64</td>
</tr>
<tr>
<td></td>
<td>* AS2_ENC_ALGO_RC2_128</td>
</tr>
<tr>
<td></td>
<td>* AS2_ENC_ALGO_DES</td>
</tr>
<tr>
<td></td>
<td>* AS2_ENC_ALGO_DES_EDE3 (Default)</td>
</tr>
<tr>
<td></td>
<td>* AS2_ENC_ALGO_AES_128</td>
</tr>
<tr>
<td></td>
<td>* AS2_ENC_ALGO_AES_192</td>
</tr>
<tr>
<td></td>
<td>* AS2_ENC_ALGO_AES_256</td>
</tr>
<tr>
<td>AS2_DIGEST_ALGO</td>
<td>Enter the AS2 digest algorithm for signing the messages. Set either of the</td>
</tr>
<tr>
<td></td>
<td>following:</td>
</tr>
<tr>
<td></td>
<td>* AS2_DIGEST_ALGO_MD5 (Default)</td>
</tr>
<tr>
<td></td>
<td>* AS2_DIGEST_ALGO_SHA1</td>
</tr>
<tr>
<td>AS2_ASYNC_ADDRESS</td>
<td>Enter the asynchronous address to which MDN notifications should be set.</td>
</tr>
<tr>
<td>AS2_ASYNC_EMAIL_SERVER_HOST</td>
<td>Enter the email server host for asynchronous email MDN.</td>
</tr>
<tr>
<td>AS2_ASYNC_EMAIL_SERVER_PORT</td>
<td>Enter the email server port for asynchronous email MDN.</td>
</tr>
<tr>
<td>AS2_ASYNC_EMAIL_SERVER_USERNAME</td>
<td>Enter the email server USERNAEME for asynchronous email MDN.</td>
</tr>
</tbody>
</table>
Delivering Documents over AS2

Table 8–10 (Cont.) Properties for Delivering Documents over AS2

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS2_ASYNC_EMAIL_SERVER_PASSWORD</td>
<td>Enter the email server PASSWORD for asynchronous email MDN.</td>
</tr>
<tr>
<td>AS2_ASYNC_EMAIL_SERVER_FOLDER_NAME</td>
<td>Enter the IMAP folder name for asynchronous email MDN.</td>
</tr>
<tr>
<td>AS2_SENDER_PKCS12_FILE</td>
<td>Location of the sender’s PKCS12 (public/private key) file.</td>
</tr>
<tr>
<td>AS2_SENDER_PKCS12_PASSWORD</td>
<td>Password for the sender’s PKCS12 (public/private key).</td>
</tr>
<tr>
<td>AS2_RECEIVER_CERTIFICATES_FILE</td>
<td>Location of the receiver's certificates file.</td>
</tr>
<tr>
<td>AS2_DELIVERY_RECEIPT_DIRECTORY</td>
<td>Directory to store the delivery receipts. This directory must be specified if to receive delivery receipts.</td>
</tr>
<tr>
<td>AS2_HTTP_HOST</td>
<td>Required. Enter the server host name.</td>
</tr>
<tr>
<td>AS2_HTTP_PORT</td>
<td>Enter the server HTTP port number. The default is 80.</td>
</tr>
<tr>
<td>AS2_HTTP_REMOTE_DIRECTORY</td>
<td>Required. Enter the remote directory name. (Example: /home/)</td>
</tr>
<tr>
<td>AS2_HTTP_REMOTE_FILENAME</td>
<td>Required. Enter the remote file name.</td>
</tr>
<tr>
<td>AS2_HTTP_AUTHTYPE</td>
<td>Enter the HTTP authentication type. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>■ AS2_HTTP_AUTHTYPE_NONE - no authentication (Default)</td>
</tr>
<tr>
<td></td>
<td>■ AS2_HTTP_AUTHTYPE_BASIC - Use HTTP basic authentication.</td>
</tr>
<tr>
<td></td>
<td>■ AS2_HTTP_AUTHTYPE_DIGEST - user HTTP digest authentication.</td>
</tr>
<tr>
<td>AS2_HTTP_USERNAME</td>
<td>Enter the username for HTTP authentication.</td>
</tr>
<tr>
<td>AS2_HTTP_PASSWORD</td>
<td>Enter the password for HTTP authentication.</td>
</tr>
<tr>
<td>AS2_HTTP_ENCTYPE</td>
<td>Set the encryption type. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>■ AS2_HTTP_ENCTYPE_NONE - no encryption (default)</td>
</tr>
<tr>
<td></td>
<td>■ AS2_HTTP_ENCTYPE_SSL - use secure socket layer (SSL)</td>
</tr>
<tr>
<td>AS2_HTTP_TIMEOUT</td>
<td>Enter the time out allowance in milliseconds. Default is 60,000 (1 minute)</td>
</tr>
<tr>
<td>AS2_HTTP_PROXY_HOST</td>
<td>Required. Enter the proxy server host name.</td>
</tr>
<tr>
<td>AS2_HTTP_PROXY_PORT</td>
<td>Enter the proxy server port number. Default is 80.</td>
</tr>
<tr>
<td>AS2_HTTP_PROXY_AUTHTYPE</td>
<td>■ AS2_HTTP_AUTHTYPE_NONE - no authentication (Default)</td>
</tr>
<tr>
<td></td>
<td>■ AS2_HTTP_AUTHTYPE_BASIC - Use HTTP basic authentication.</td>
</tr>
<tr>
<td></td>
<td>■ AS2_HTTP_AUTHTYPE_DIGEST - user HTTP digest authentication.</td>
</tr>
</tbody>
</table>
### 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**

```java
// 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**

```java
// 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( 
```

---

Table 8–10 (Cont.) Properties for Delivering Documents over AS2

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS2_HTTP_PROXY_USERNAME</td>
<td>Enter the username for proxy authentication.</td>
</tr>
<tr>
<td>AS2_HTTP_PROXY_PASSWORD</td>
<td>Enter the password for HTTP proxy authentication.</td>
</tr>
</tbody>
</table>
Delivering Documents over AS2

Using the Delivery Manager Java APIs

```java
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**

```java
// 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:
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:

```java
// 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 DeliveryPropertyDefinition:

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:

```java
// create delivery manager instance
DeliveryManager dm = new DeliveryManager();
```
Asynchronous Delivery Requests

// 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**

```java
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**

```java
// 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**

```java
// 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**

```xml
<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**

```java
// 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**

```xml
<server name="printer1" type="ipp_printer" default="true">
  <uri>ipp://myserver:80/printers/MyPrinter1/.printer</uri>
  <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:
Internationalization Support

### SMTP

- **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 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**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFFERING_MODE</td>
<td>Valid values are &quot;true&quot; (default) and &quot;false&quot;. See Section 8.14, &quot;Direct and Buffering Modes.&quot;</td>
</tr>
<tr>
<td>TEMP_DIR</td>
<td>Define the location of the temporary directory.</td>
</tr>
<tr>
<td>CA_CERT_FILE</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Example 8–34 Sample Code for Setting Global Properties**

```java
// 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

```java
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;
    }
}
```

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.delivery.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

```java
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;
    }
}
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

```java
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**

```java
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()
    {
```
Adding a Custom Delivery Channel

Using the Delivery Manager Java APIs

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**

  ```java
  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
  ```
8.21 Configuration File Support

The delivery system 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
  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
<?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>
    </server>
  </servers>
</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

```java
// 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 server 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>
Configuration File Support

- `<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>`
- `<filterOutputContentType>`
- `<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>`
This part contains the following chapters:

- Chapter 9, "Making a View Object Available to BI Publisher as a Data Source"
- Section 10, "Using BI Publisher with Oracle JDeveloper"
- Chapter 11, "Setting Up After-Report Triggers"
- Section 12, "Adding Extensions to the Layout Editor"
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:

- Section 9.1, "Prerequisites"
- Section 9.2, "Configuring the Application Module"
- Section 9.3, "Updating web.xml"
- Section 9.4, "Updating weblogic.xml"
- Section 9.5, "Deploying the Application Module"
- Section 9.6, "Updating the providers.xml File"

---

**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**

2. In the Source view, update the web.xml file with the following:

   ```xml
   <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>
     <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>

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:
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.
This chapter describes how to add a BI Publisher report to an Oracle Application Development Framework (ADF) application using Oracle JDeveloper. It includes the following sections:

- Section 10.1, "Including Oracle BI Publisher Reports in ADF Applications"
- Section 10.2, "Installing the BI Publisher Extension"
- Section 10.3, "Adding BI Publisher Content to an ADF Project"
- Section 10.4, "Configuring a Security Policy"
- Section 10.5, "Deploying and Running the Application"
- Section 10.6, "Passing Parameters from the Application Page"
- Section 10.7, "Using a Push Data Model"

**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

### 10.1 Including Oracle BI Publisher Reports in ADF Applications

BI Publisher provides an integration with Oracle Application Development Framework (ADF) that enables you to embed a BI Publisher report in an application (JSF) page. At runtime, your application sends a web service request to the BI Publisher server to run a report in the BI Publisher catalog and retrieve the output to display in your application page. The data for your report can be generated by the BI Publisher data engine, or you can push data from another source to the BI Publisher report formatting engine. The integration also supports passing parameters from your application page back to the BI Publisher server.

To facilitate development of your application, BI Publisher provides an extension to JDeveloper. The extension enables you to drag and drop a report region to your page and establish the connection between this region and the BI Publisher server. The extension also enables you to define aspects of the report format and to set properties for the region.

This integration is illustrated in Figure 10–1.
As shown in the figure, during design time, the BI Publisher extension establishes the connection between your page and the BI Publisher server and enables you to set properties for your report region. After deploying your application, at run time, the ADF application uses a web service to run and retrieve the BI Publisher report back to your page.

To enable communication between your application and the BI Publisher web service, define a policy using Oracle Web Services Policy Manager (OWSM).

**Prerequisites**

Prerequisites of this integration include:

- Oracle JDeveloper 11.1.1.6 or 11.1.1.7.
- BI Publisher and your application server are configured to use Oracle Fusion Middleware Security with Oracle Web Services Policy Manager.
- BI Publisher and your application server are configured to use the same single sign-on server.

**Limitations**

Limitations of this integration include:

- BI Publisher’s interactive output type is not supported.
- Oracle Fusion Middleware Security is the only security model supported.
- BI Publisher Trial Edition does not support this integration.

### 10.2 Installing the BI Publisher Extension

Download the BI Publisher extension for JDeveloper, **BI Publisher ADF Components**, from one of the following locations:

- Oracle Extensions for Oracle JDeveloper Update Center
  

- Oracle BI Publisher Downloads page on the Oracle Technology Network:
  
Save the zip file to a local directory. Use the JDeveloper Check for Updates wizard available from the Help menu. For the update Source, choose Install From Local File and select the extension file from the saved location.

10.3 Adding BI Publisher Content to an ADF Project

After you install the BI Publisher extension you can create a project that includes a BI Publisher report region. The following procedures describe how to add BI Publisher content to an ADF project:

- Section 10.3.1, "Adding the BI Publisher Technology Scope"
- Section 10.3.2, "Adding a BI Publisher Region to the JSF Page"
- Section 10.3.3, "Configuring Connection to the BI Publisher Server"
- Section 10.3.4, "Setting Properties for the BI Publisher Region"

10.3.1 Adding the BI Publisher Technology Scope

To display the BI Publisher options, add the BI Publisher technology scope to your project.

1. In Oracle JDeveloper, go to the Projects Pane and right-click the project to which you want to add the technology scopes and select Project Properties.
2. Select Technology Scope.
3. In the Available Technologies list, select BI Publisher as shown in Figure 10–2:

   Figure 10–2 Selecting the BI Publisher Technology Scope

   ![Figure 10–2 Selecting the BI Publisher Technology Scope](image)

4. Click the right shuttle button to add BI Publisher and its dependent technologies to your project. Dependent technologies are moved together. Figure 10–3 shows the results of this action.
5. Click Finish.

10.3.2 Adding a BI Publisher Region to the JSF Page

The BI Publisher region must reside on a JSF page. To add a JSF page to your project:

1. Right-click your project and select New.
2. From the Gallery dialog, on the Categories pane, under Web Tier, select JSF.
3. From the items presented in the right pane, select JSF Page, as shown in Figure 10–4.

**Figure 10–5  Create JSF Page Dialog**

![Create JSF Page Dialog](image)

5. From the Component Palette list, select BI Publisher, as shown in Figure 10–6.

**Figure 10–6  Component Palette List**

![Component Palette List](image)

6. Drag and drop the BI Publisher Region component from the palette to the page.

7. Once the BI Publisher Region is dropped to the JSPX page, the **Insert BI Publisher Region** dialog, shown in Figure 10–7, prompts you to enter a region ID for the BI Publisher Region. The region ID must be unique and must not include the underscore "_" character.
Click OK. When you insert the BI Publisher region notice the following:

- A placeholder image displays in the BI Publisher region of your page.
- The BI Publisher configuration file, /WEB-INF/xmlp-client-config.xml is created. Click Refresh on the Projects pane if it does not display immediately.
- The Property Inspector presents the BI Publisher region properties when the BI Publisher region is selected.

Figure 10–8 shows these items:

8. In the Property Inspector, under the Appearance region, enter the report region Width and Height in px to specify the size of the report region in your application page. For example 1000 px and 800 px. No defaults are assigned for height and width, so ensure that you set these fields.

10.3.3 Configuring Connection to the BI Publisher Server

To establish a connection to the BI Publisher server, update the xmlp-client-config.xml file under Project > Web Content > WEB-INF with the connection information as follows:

1. Double-click xmlp-client-config.xml to open the file for editing.

Figure 10–9 shows the sample xmlp-client-config.xml file.
2. Update the properties (keys) as shown in the following example. Add the properties not found in the default file.

The sample `xmlp-client-config.xml` file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <comment>BIP Server Information</comment>
    <entry key="bipuri">http://example.com:7001/xmlpserver</entry>
    <entry key="sso.bipuri">http://example.com:7001/xmlpserver</entry>
    <entry key="username">username</entry>
    <entry key="policy">oracle/wss_username_token_client_policy</entry>
</properties>
```

- **bipuri** - enter the URL for the BI Publisher server. This may be behind a firewall. For example: `http://hostname.com:7001/xmlpserver`
- **sso.bipuri** - enter the external SSO-enabled BI Publisher URL, which is accessible by a client browser. In Oracle Fusion Applications Business Intelligence domains this property is already configured. For other implementations, you must enter this property.
- **username** - enter the secure user name to connect to BI Publisher. This user is the authenticated user used to connect from your application to BI Publisher.
- **policy** - enter the OWSM web service security policy to use when connecting to the BI Publisher web service. See Section 10.4, "Configuring a Security Policy" for more information.

### 10.3.4 Setting Properties for the BI Publisher Region

The following properties are available in the Property Inspector:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Id</strong></td>
<td>The unique ID for the BI Publisher region. You assign this ID when you insert the BI Publisher region to your page. The ID must not contain spaces.</td>
</tr>
<tr>
<td><strong>Report Path</strong></td>
<td>The path to the report in the BI Publisher catalog. Reports have the extension &quot;.xdo&quot;. Begin the path from the first level beneath &quot;Shared Folders,&quot; but do not include &quot;Shared Folders.&quot; For example: &quot;/Samples/1.+Overview/Balance+Letter+Report.xdo&quot;</td>
</tr>
</tbody>
</table>
Table 10–1  (Cont.) Common Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rendered</td>
<td>The control to show or hide the BI Publisher region when the page is rendered. The default is true (show).</td>
</tr>
</tbody>
</table>

Table 10–2  Advanced Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layout Name</td>
<td>The layout from the report definition to apply to the report data. For example, &quot;My Layout&quot;. If you do not specify a value here the default layout from the report definition is used.</td>
</tr>
<tr>
<td>OutputFormat</td>
<td>Sets the default output format. See Table 10–4 for the list of valid values. If you do not specify a value here the default output format from the report definition is used.</td>
</tr>
</tbody>
</table>
| Parameters      | To pass parameters from your application page to the BI Publisher report, use this property to specify the backing bean that captures the parameter values. Use the Expression Builder to populate this field. For example: 

```
#{UIBackingBean.parameters}
```

See Section 10.6, "Passing Parameters from the Application Page" for information. |
| ReportData      | If your report uses a push data model, use this property to specify the backing bean that contains the method that defines the report data location. Use the Expression Builder to populate this field. For example: 

```
#{UIBackingBean.reportData}
```

See Section 10.7, "Using a Push Data Model" for more information. |
| Locale          | Enter a default locale format using the ISO language code-country code combination, for example en-US. |

Table 10–3  Appearance Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width</td>
<td>Sets the report region width. Enter the value in pixels, for example, 1000 px.</td>
</tr>
<tr>
<td>Height</td>
<td>Sets the report region height. Enter the value in pixels, for example, 800 px.</td>
</tr>
<tr>
<td>RenderActionPanel</td>
<td>Controls whether to show or hide the report viewer action panel. The default is true. Note that if you set this to false, you must set renderReportOnLoad to true.</td>
</tr>
<tr>
<td>RenderFormatList</td>
<td>Controls whether to show or hide the output Format List. The default is true.</td>
</tr>
<tr>
<td>RenderLocaleList</td>
<td>Controls whether to show or hide the Locale List. The default is true.</td>
</tr>
<tr>
<td>RenderReportOnLoad</td>
<td>When set to true, the report is generated when the ADF page is launched. The default is true.</td>
</tr>
</tbody>
</table>
10.4 Configuring a Security Policy

For your application to run securely you must ensure that both server-side and client-side web service security policies are configured. In Enterprise installs of Business Intelligence, the server-side policies may be set up by provisioning scripts; in these cases you only configure the client-side service policy.

For more information about the Oracle WebLogic Server web service policies, see “Using Oracle Web Services Manager Security Policies” in Oracle Fusion Middleware Securing WebLogic Web Services for Oracle WebLogic Server.

Once the server-side policy is defined and attached as a global policy or a local policy, you enable it on the client-side by updating the xmlp-server-config.xml file.

Note: In a provisioned Oracle Fusion Applications environment the tasks in this section are not required because authentication is configured by the provisioning scripts.

For standalone environments, SSO must be set up and the credentials for login must be identical on the server side and the client side or the user will be challenged for credentials to log in to BI Publisher.

### Table 10–4 Valid Values for OutputFormat

<table>
<thead>
<tr>
<th>Output Format</th>
<th>Value to Enter for OutputFormat Property</th>
<th>Template Types That Can Generate This Output Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interactive</td>
<td>N/A</td>
<td>Not supported</td>
</tr>
<tr>
<td>HTML</td>
<td>html</td>
<td>BI Publisher, RTF, XSL Stylesheet (FO)</td>
</tr>
<tr>
<td>PDF</td>
<td>pdf</td>
<td>BI Publisher, RTF, PDF, Flash, XSL Stylesheet (FO)</td>
</tr>
<tr>
<td>RTF</td>
<td>rtf</td>
<td>BI Publisher, RTF, XSL Stylesheet (FO)</td>
</tr>
<tr>
<td>Excel (mhtml)</td>
<td>excel</td>
<td>BI Publisher, RTF, Excel, XSL Stylesheet (FO)</td>
</tr>
<tr>
<td>Excel (html)</td>
<td>excel2000</td>
<td>BI Publisher, RTF, Excel, XSL Stylesheet (FO)</td>
</tr>
<tr>
<td>Excel (*.xlsx)</td>
<td>xlsx</td>
<td>BI Publisher, RTF, XSL Stylesheet (FO)</td>
</tr>
<tr>
<td>PowerPoint (mhtml)</td>
<td>ppt</td>
<td>BI Publisher, RTF, XSL Stylesheet (FO)</td>
</tr>
<tr>
<td>PowerPoint (*.pptx)</td>
<td>pptx</td>
<td>BI Publisher, RTF, XSL Stylesheet (FO)</td>
</tr>
<tr>
<td>MHTML</td>
<td>mhtml</td>
<td>BI Publisher, RTF, Flash, XSL Stylesheet (FO)</td>
</tr>
<tr>
<td>PDF/A</td>
<td>pdfa</td>
<td>BI Publisher, RTF, XSL Stylesheet (FO)</td>
</tr>
<tr>
<td>PDF/X</td>
<td>pdfx</td>
<td>BI Publisher, RTF, XSL Stylesheet (FO)</td>
</tr>
<tr>
<td>Zipped PDFs</td>
<td>pdfz</td>
<td>BI Publisher, RTF, PDF, XSL Stylesheet (FO)</td>
</tr>
<tr>
<td>FO Formatted XML</td>
<td>xslfo</td>
<td>BI Publisher, RTF, XSL Stylesheet (FO)</td>
</tr>
<tr>
<td>Data (XML)</td>
<td>xml</td>
<td>BI Publisher, RTF, PDF, Excel, Flash, XSL Stylesheet (FO), XSL Stylesheet (HTML XML/Text)</td>
</tr>
<tr>
<td>Data (CSV)</td>
<td>csv</td>
<td>BI Publisher, RTF, PDF, Excel, Flash, XSL Stylesheet (FO), XSL Stylesheet (HTML XML/Text), Etext</td>
</tr>
<tr>
<td>XML</td>
<td>xml</td>
<td>XSL Stylesheet (HTML XML/Text)</td>
</tr>
<tr>
<td>Text</td>
<td>text</td>
<td>XSL Stylesheet (HTML XML/Text), Etext</td>
</tr>
<tr>
<td>Flash</td>
<td>flash</td>
<td>Flash</td>
</tr>
</tbody>
</table>
10.4.1 Viewing and Configuring the Server Policy

If your environment already includes a global policy or a local policy attached to the BI Publisher web service you do not need to add a policy. Use this procedure to verify the policy on the server side.

To view or set up the server policy:

1. Log in to Oracle Fusion Middleware Control. The URL is typically:  
   http://hostname.domain:port/em
2. Open the home page for the server instance where BI Publisher is installed. In simple installations this is the AdminServer.

   From the home page menu select Web Services as shown in Figure 10–10.

   **Figure 10–10  Selecting Web Services**

3. On the Web Services page, select the BI Publisher (bipublisher(11.1.1)) service PublicReportWSService as shown in Figure 10–11.

   **Figure 10–11  Selecting the BI Publisher Web Service**

4. In the detail page for the web service, attached global policies display in the Global Policy table. Local policies display in the Directly Attached table. If a policy
is already attached, proceed to Section 10.4.2, "Configuring the Client-Side Service Policy."

If there are no policies attached, attach a local (directly attached) policy by clicking Attach/Detach.

**Figure 10–12 Page Showing Attached Security Policies**

5. From the Available Policies table, select the policy to attach to the BI Publisher web service and click Attach. For example, oracle/wss_saml_or_username_token_service_policy.

6. Click OK then restart the BI Publisher application.

7. Proceed to Section 10.4.2, "Configuring the Client-Side Service Policy."

### 10.4.2 Configuring the Client-Side Service Policy

Configure the client-side service policy by making the appropriate entries in the xmlp-client-config.xml file.

On your client:

1. Navigate to the xmlp-client-config.xml in your project under Project > Web Content > WEB-INF.

2. To the properties listing, add the entry key="policy" and enter the client policy name, for example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM 'http://java.sun.com/dtd/properties.dtd'>
<properties>
  <comment>BIP Server Information</comment>
  <entry key="bipuri">http://example.com:7001/xmlpserver</entry>
  <entry key="sso.bipuri">http://example.com:7001/xmlpserver</entry>
```

```xml
```
10.5 Deploying and Running the Application

To deploy and run the application, first copy the required libraries to the client where JDeveloper is installed and then follow the deployment steps.

- **Copy Libraries**
- **Deploying the Application**

**Copy Libraries**

1. Ensure that Oracle Business Intelligence Publisher shared libraries are installed on your target WebLogic Server instance.
   - `oracle.xdo.runtime`: 
     
   
   $MW_HOME/jdeveloper/xdo/lib/xdoruntime.ear
   - `oracle.xdo.webapp`: 
     
   
   $MW_HOME/jdeveloper/xdo/lib/xdowebapp.war

2. When the shared libraries are installed, you can see them in the WebLogic Server console as shown in Figure 10–13.

**Figure 10–13 Libraries Shown in WebLogic Server Console**

Deploying the Application

To deploy and run the application:

1. From Project Properties, click **Deployment**, click **New** and enter a value for the J2EE application name. Figure 10–14 shows the Create Deployment Profile dialog.
2. Click WEB-INF/lib. BI Publisher Runtime does not have to be included so long as weblogic.xml and weblogic-application.xml are properly configured. Figure 10–15 shows libraries selected for deployment.

![Figure 10–15 Libraries Selected for Deployment](image)

3. Add a shared library reference to weblogic.xml and weblogic-application.xml as shown in Figure 10–16 and Figure 10–17.
4. From the JDeveloper menu, select Run > Start Server Instance. Do not run the JSPX page directly.

5. Right-click the application and select Deploy - (your BIP ADF Application name) - to IntegratedWLSConnection as shown in Figure 10–18. Make sure the BIP ADF Application deployment profile has your J2EE application name (webapp21 in this example) selected on Application Assembly.
Deploying and Running the Application

Using BI Publisher with Oracle JDeveloper

Figure 10–18  Deploying an Application to a Connection

The deployment completed message displays in the Log window.

6. Open your browser and enter

http://<hostname>:<port>/j2eeWebAppName/faces/report.jspx

in the address field. When the page launches, the contents display according to the
settings you defined for RenderActionPanel, RenderFormatList, RenderLocaleList,
and RenderReportOnLoad.

Figure 10–19 shows an example display when RenderActionPanel,
RenderFormatList, and RenderLocaleList are set to true and RenderReportOnLoad
is set to false. The Template list, Output Format list, and Locale list display and
can be updated by the user. To render the report, the user must click Go.

Figure 10–19  Displayed Fields

7. Click Go. Your report displays, as shown in Figure 10–20.
10.6 Passing Parameters from the Application Page

You can pass parameters entered from your application page to the report using a backing bean. The BIP region tag must define the parameters property and bind it to the backing bean method as shown:

```xml
<?xml version='1.0' encoding='UTF-8'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
  xmlns:xdo="http://xmlns.oracle.com/xdo/faces">
  <jsp:directive.page contentType="text/html;charset=UTF-8"/>
  <f:view>
    <af:document id="d1">
      <af:form id="f1">
        <af:inputText label="Employee ID" id="it1"
            binding="#{UIBackingBean.empId}"/>
      </af:form>
      <af:separator id="s1"/>
      <xdo:BIPRegion id="bipregion1"
        reportId="/Samples/Overview/BalanceLetterReport.xdo"
        width="1000px" height="800px"
        parameters="#{UIBackingBean.parameters}" reportData=""/>
    </af:document>
  </f:view>
</jsp:root>
```

Specify the backing bean in the Parameters property of the Property Inspector. See Table 10–2, “Advanced Properties” for more information.

10.7 Using a Push Data Model

Some reports require XML data from a source other than a BI Publisher data model. For this requirement, you can push data to the BI Publisher server to use as input for your report. For this scenario, store the data in a location that is accessible by your
application and create a backing bean that defines the getter method to retrieve the report data from its stored location. You still define the Report Path, Layout Name and other properties for the report as defined on the BI Publisher server.

The BIPRegion tag must define the reportData property for the tag, and bind it to the backing bean method. The following sample code shows this binding:

```xml
<?xml version='1.0' encoding='UTF-8'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
    xmlns:xdo="http://xmlns.oracle.com/xdo/faces">
    <jsp:directive.page contentType="text/html;charset=UTF-8"/>
    <f:view>
        <af:document id="d1">
            <af:form id="f1">
                <xdo:BIPRegion id="id3"
                    reportId="/Samples/Overview/Balance+Letter+Report.xdo"
                    width="1000px" height="800px"
                    reportData="#{UIBackingBean.reportData}"/>
            </af:form>
        </af:document>
    </f:view>
</jsp:root>
```

The following code sample shows the UIBackingBean class referenced in the previous sample:

```java
package fusionApp;
import java.io.RandomAccessFile;
import java.util.Hashtable;
import java.util.Properties;
import oracle.adf.view.rich.component.rich.input.RichInputText;
public class UIBackingBean {
    private RichInputText empId;
    private Properties mParameters;
    private byte[] reportData;

    public UIBackingBean() {
        mParameters = new Properties();
        reportData = null;
    }

    public void setEmpId(RichInputText empId) {
        this.empId = empId;
    }

    public RichInputText getEmpId() {
        return empId;
    }

    public Hashtable getParameters() {
        if(empId != null && empId.getValue() != null) {
            mParameters.put("userid", new String[] {empId.getValue().toString()});
            String[] values = (String[]) mParameters.get("userid");
            System.out.println("getParameters() is called : " + values[0]);
        }
        return mParameters;
    }
}
```
public byte[] getReportData() {
    String dataFile = "/tmp/reportData.xml";
    try {
        RandomAccessFile raf = new RandomAccessFile(dataFile, "r");
        reportData = new byte[(int) raf.length()];
        raf.read(reportData);
        raf.close();
        // write this to temp
        java.io.FileOutputStream outStream = new java.io.FileOutputStream("/tmp/output_reportData");
        outStream.write(reportData);
        outStream.close();
    } catch (Exception e) {
        System.out.println("Error reading file : " + e.getMessage());
    }
    return reportData;
}

To define a push data model:
1. Create a backing bean class that defines the getter method that retrieves your data
from its stored location.
2. In JDeveloper, create the BIP Region for your page and specify all properties for
the report that you wish to run on the BI Publisher server (Report Path, Layout
Name, Output Format, and so on).
3. In the Property Inspector, for the ReportData property, enter the expression that
defines the variable to pass to the backing bean method to call the appropriate
getter method to retrieve the report data from its stored location. Use the
Expression Builder to build the unified expression language (EL) syntax.

10.8 Conditionally Required Settings
Depending on the type of report you are running you may need to make other settings
for your reports to run as expected.

This section contains the following topics:
• Section 10.8.1, "Setting the MIME Types for Your Report Output Type"

10.8.1 Setting the MIME Types for Your Report Output Type
To ensure that all the output types can be successfully generated from your page, add
the MIME-type mappings to the web.xml file. Typically you only need to add the
MIME-type mappings for Excel output.

To add the MIME-type mappings:
1. Double-click the Project web.xml file to open it for editing.
2. Expand the MIME mappings region.
3. Click Add.
4. In the Property Inspector, enter the following:

- extension - (Required) Enter the file name extension of the document type you want to map to a particular MIME type for your web application. For example, pdf. This field corresponds to the <extension> tag of the <mime-mapping> subelement.

- mime-type - (Required) Enter the document MIME type that you want to map to the specified file name extension. For example, application/pdf for Adobe's Portable Document Format. This field corresponds to the <mime-type> tag of the <mime-mapping> subelement. Note, the official registry of Internet MIME types is managed by the Internet Assigned Numbers Authority (IANA) at www.iana.org.

The following table shows the required MIME types to generate the Microsoft Excel output options supported by BI Publisher:

<table>
<thead>
<tr>
<th>Extension</th>
<th>Mime-Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>xls</td>
<td>application/vnd.ms-excel</td>
</tr>
<tr>
<td>xlsx</td>
<td>application/vnd.openxmlformats-officedocument.spreadsheetml.sheet</td>
</tr>
<tr>
<td>xlsm</td>
<td>application/vnd.ms-excel.sheet.macroEnabled.12</td>
</tr>
<tr>
<td>mhtml</td>
<td>message/rfc822</td>
</tr>
</tbody>
</table>
This chapter describes how to set up an after-report trigger using an HTTP servlet. It includes the following sections:

- Section 11.1, "Overview of After-Report Triggers"
- Section 11.2, "Setting Up After-Report Triggers"

11.1 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.

11.1.1 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.

11.1.2 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 11.2.1, "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.
11.2 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 11–1 Sample Code for Setting Up After-Report Triggers**

```java
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);
}
```

11.2.1 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:

```xml
<web-app>
  <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>
</web-app>
```
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.

### 11.2.2 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 11–2 Sample Program Code**

```java
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.getDocumentDataContentType());
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;
}
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 12.1, "Using Layout Editor Plug-in Extensions"
- Section 12.2, "Implementing Plug-ins"
- Section 12.3, "Coding the Custom Plug-in"
- Section 12.4, "Property Support"
- Section 12.5, "Samples"
- Section 12.6, "Creating a Data Field Plug-in"

12.1 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.

12.2 Implementing Plug-ins

To implement a plug-in:

1. Code the JavaScript plug-in using the guidelines described in Section 12.3, "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.
12.3 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 12.3.1, "Plug-in Structure"
- Section 12.3.2, "JavaScript APIs That Can Be Used in Custom Plug-ins"

12.3.1 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/smiley.png"

  - **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:
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 12.4, "Property Support."

### 12.3.2 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.

#### 12.3.2.1 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]
    },
  ]
}
```
12.3.2.2 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

```
xd0.api.getPixelValue(lengthString)
```

The method takes one parameter:

lengthString
A string value that specifies the length. Supported units are "px", "pt", "in", and "cm".

12.4 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
12.4.1 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

12.5 Samples

Following are plug-in examples:

- [Section 12.5.1, "Example of Static Plug-in: Company Logo"]
- [Section 12.5.2, "Example Plug-in to Insert YouTube Video"]

12.5.1 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 JavaSript 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:

   ```javascript
   { 
     id: "com.oracle.xdo.logo",
     component: {
       name: "Logo",
       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 12–1.
12.5.2 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:

```html
<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:

```javascript
{
  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 12–3 shows how the properties appear in the Properties pane of the layout editor.

Figure 12–3 Properties Display in the Layout Editor for the YouTube Plug-in

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:

```javascript
render: function(context, containerElem, rows, fields, props) {
  var videoid = props['videoid'];
  var width = props['width'];
```
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.

```json
{
  id: "com.oracle.xdo...",
  component: {
    name: "Field Test"
  }
}
fields:
```
Creating a Data Field Plug-in

Adding Extensions to the Layout Editor

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:

```javascript
render: function(context, containerElem, rows, fields, props) {
  // setup column
  var html = '<table>
  for (var i=0, rowCount=rows.length; i<rowCount; i++) {
    html += '<tr>
    for (var j=0, colCount=col.length; j<colCount; j++) {
      html += '<td>
      html += col[j];
      html += '</td>
    }
    html += '</tr>
  }
```


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

A
after-report trigger, 11-1
APIs
core, 7-1

B
barcode formatting APIs, 7-43
barcodes
API description, 7-43
base data types, 2-1
buffering mode
delivery server, 8-29

C
CatalogService, 1-1, 6-1
complex data types, 2-2
configuration file
delivery manager, 8-40
core APIs, 7-1

D
data type mappings, 2-1
data types
base, 2-1
complex, 2-2
debugging
TCP Monitor utility, 1-2
Web services, 1-2
delivery
using operating system commands, 8-28
delivery channels
adding custom, 8-34
delivery manager
configuration file, 8-40
Java APIs, 8-1
delivery server, 8-33
buffering mode, 8-29
date expression, 8-32
direct mode, 8-29
document filter support, 8-31
global properties, 8-34
local file system delivery, 8-29
direct mode
delivery server, 8-29

E
e-mail delivery, 8-2

F
fax delivery, 8-13
FTP delivery, 8-18

G
global properties
delivery server, 8-34

H
HTTP
delivering documents over, 8-21
HTTP notification
sample program, 11-3
HTTP servlet
registering for after-report trigger, 11-2

I
in-session methods
about, 1-2

M
merging PDF files, 7-20
methods
in-session, 1-2

P
PDF files
merging, 7-20
printing, 8-8
providers.xml file, 9-3

R
ReportService, 1-1, 4-1
S
ScheduleService, 1-1, 3-1
secure FTP
delivery, 8-19
SecurityService, 1-1, 5-1

T
TCP Monitor utility, 1-2
triggers
after-report, 11-1

V
view objects
creating, 9-1

W
Web services
about, 1-1
CatalogService, 1-1
debugging, 1-2
ReportService, 1-1
ScheduleService, 1-1
SecurityService, 1-1
WebDAV delivery, 8-16

X
XML-to-Java mappings
for data types, 2-1