



XAware Connector, Version 5.0

Reference Guide

© 2007 XAware, Inc.
5555 Tech Center Drive, Suite 200
Colorado Springs, Colorado 80919
www.xaware.com • Phone (719) 884-5400

COPYRIGHT STATEMENTS

Copyrights © 1999-2007 XAware, Inc. All rights reserved. This statement pursuant to Title 17, United States Code, Section 401 (17 USC 401) hereby notifies all users of the copyright law protection under which this and all other XAware, Inc. documents and software are dispersed. Under the copyright laws, neither the documentation nor the software may be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form, in whole or in part, without the prior written consent, of XAware, Inc., except in the manner described in the documentation.

DISCLAIMER OF LIABILITY

While XAware, Inc. strives to maintain a high level of accuracy in its internal and external forms of documentation, neither XAware, Inc. nor any of its employees makes any guarantees, express or implied, to the completeness, or usefulness of any documentation, in whole or in part, that was either copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form with or without prior written consent granted by XAware, Inc., including its merchantability and fitness for a particular purpose other than the purpose for which the document was proposed initially upon its original date of publication by XAware, Inc.; furthermore, neither XAware, Inc. nor any of its employees assumes any legal liability or responsibility, in whole or in part, for the accuracy of said documentation.

TRADEMARKS

Microsoft®, Windows®, and Internet Explorer are registered trademarks of Microsoft Corporation. Java, J2EE, J2SE, J2ME, and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. Oracle® is a registered trademark of Oracle Corporation. UNIX® is a registered trademark of the Open Group. SAP® is a registered trademarks of SAP AG. Siebel® is a registered trademark of Siebel Systems, Inc. CORBA is a registered trademark of Object Management Group, Inc. WebSphere® is a registered trademark of International Business Machines Corporation. JBoss® is a registered trademark of JBoss, Inc. WebLogic® is a registered trademark of BEA Systems, Inc. Other product names mentioned in this book may be trademarks or registered trademarks of their respective companies and are hereby acknowledged.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>).

XAware TRADEMARKS

The following terms are trademarks or registered trademarks of XAware, Inc. in the United States and other countries: XAware, XA-iServer, XA-Designer, XA-Suite, XA-Connector, XA-Adapters, XA-Manage, BizDocument, BizComponent, BizDriver, BizView, BizFile, BizView session, XA-iServices, XA-iConnectors, XCelerator, BizDoc, BizComp, Enterprise Aware, Engine, Designer.



Contents

<i>Preface</i>	<i>iii</i>
<i>Chapter 1 - Overview</i>	<i>1</i>
Introduction	2
What is the lifecycle of a BizView session?	2
How does the Engine respond to requests?	3
How do you know which connector to use?	4
What about error handling with Connectors?	5
How to set Parameters with Variables?	6
<i>Chapter 2 - Using the HTTP-Servlet Connector</i>	<i>7</i>
Introduction	8
What is the format of the client request?	9
Invoking a BizView File	9
... with an HTTP GET	10
... with an HTTP POST	10
What are the configuration parameters for this connector?	11
Connector parameter list	12
How are errors handled with this connector?	14
<i>Chapter 3 - Using the SOAP Connector</i>	<i>17</i>
Introduction	18
What is the format of the client request?	18
What does the client application code look like?	18

Sample Document Literal Request.	19
Sample RPC Encoded Request Type.	21
Sample RPC Literal Request Type.	23
What are the configuration parameters for this connector?	25
Connector parameter list	26
How are errors handled with this connector?	27
<i>Chapter 4 - Using the JMS Connector.</i>	<i>29</i>
Introduction.	30
What does the client application code look like?	30
Implementing the Message Sender	30
Implementing the Message Listener	33
What are the configuration parameters for this connector?	34
Connector parameter list	34
How are errors handled with this connector?	35
<i>Chapter 5 - Using the EJB Connector.</i>	<i>37</i>
Introduction.	38
What is the format of the client request?	38
What does the client application code look like?	39
What are the configuration parameters for this connector?	41
Connector parameter list	41
How are errors handled with this connector?	42
<i>Chapter 6 - Batch Processing.</i>	<i>43</i>
Introduction.	44
Setting up.	44
Getting Help	45
Invoking BizView Files.	48
Using Multiple Versions of XAware	49
<i>Index.</i>	<i>51</i>



Preface

About This Book

This book describes the XAware Connectors, which provide the connectivity protocol between client applications and the XAware Engine. Additionally, information describing which connector you should use is included, along with supporting details for each connector.

Audience

This *Reference Guide* is for use by anyone involved in defining the client application requests or building the Connectors necessary to support the deployed BizView applications.

Required Knowledge

This document assumes that you have the following knowledge prior to using XAware.

Working knowledge of HTML and XML standards as published by World Wide Web Consortium (W3C) at:

<http://www.w3c.org>

In addition, advanced knowledge of your business data source, and the type of BizView you are creating, SQL, HTTP, Socket, etc., is important.

Typographic Conventions

The following typographic conventions are used throughout this manual.

<u>Convention</u>	<u>Description</u>
Bold	Used for headings and figure captions.
<i>Italics</i>	Used for the titles of other books to which we refer. Used for values and variables. For example, “The system sets the Type to <i>Unknown</i> .” Used for terms that are being defined. For example, “The process to define a step in a workflow is called <i>step configuration</i> .”
Courier	Used for code, text displayed on the computer screen, for the names of files, directories and URLs, and for text you must type. For example: Edit the <code>config.sys</code> file.
Code boxes	Used to set off code samples. Code samples that are complete files are shown with numbered lines. Code samples that are fragments are not numbered.
Keys	Brackets are used to show keys. To show keys that need to be pressed at the same time, a plus (+) sign is used. For example: Press <Shift + Esc>.
Menu options	A vertical bar (pipe) is used between each menu option. For example: Select File Print.
Procedures	Step-by-step procedure headings are bold and marked with a diamond-shaped bullet.
<ul style="list-style-type: none"> • Field names • Tabs • Columns • Buttons • Check boxes 	Regular font is generally used for these elements, unless the name is too long and confusing, in which case single quotation marks enclose the name.

Edition History

The following table shows the version of software that this edition of the *Reference Guide* accompanies:

Edition	Major Changes
XAware 5.0	The new architecture supports more flexibility in error handling with connectors. This change, along with the addition of connector properties, is described in this edition.

Related Documentation

The XAware Designer product includes an extensive help system that includes Help (select Help | XAware Designer Help Topics), Examples (select File | New | Examples) with supporting help pages, and Javadoc (<*Install Directory*>/javadoc/api/index.html).

Information specific to what is included in this release, including *Installation* instructions, are from the XAware site (<http://www.xaware.org>). The following formal documents are also found there, including:

- *XAware Designer User's Guide*
- *Deploying XAware Engine on IBM WebSphere*
- *Deploying XAware Engine on JBoss*
- *Deploying XAware Engine on Tomcat*
- *Deploying XAware Engine on BEA Weblogic*
- *XAware Connector Reference Guide*
- *XAware Engine API*



Note: Acrobat® Reader® is required to open, navigate, and print these documents. You can obtain a free download of the latest software from Adobe (<http://www.adobe.com>).

In addition, you can obtain white papers and access forums supporting XAware by logging in to the XAware site (<http://www.xaware.org>).

Contacting XAware

Support for the XAware product and documentation is available in the user forums at www.xaware.org. Please, take a look at the community, join a forum or hive, and let us know what you think!



Chapter 1

Overview

The purpose of an XAware Connector is to take a message from a transport format (like JMS or HTTP) and marshal it. This content describes information you should understand before you begin deploying your BizView files on a web or application server to respond to client requests. Several components of the XAware system are used to successfully respond to client requests with XML views. It is important to understand how these components interact with deployed BizViews.

Introduction	page 2
What is the lifecycle of a BizView session?	page 2
How does the Engine respond to requests?	page 3
How do you know which connector to use?	page 4
What about error handling with Connectors?	page 5
How to set Parameters with Variables?	page 6

Introduction

The XAware system is not complicated; however, multiple components must work together to produce successful responses to client application systems. It is important to understand how the requests are processed before you start deploying your files. These components are necessary to a successful client response.

- The Connector and any required JAR files must be installed on the host server.
- The client application request must be formed correctly to be read by Engine.
- The BizView files, and all related custom files, must be packaged and deployed.

The XAware Connector provides the connectivity protocol between client applications and the Engine. The Connectors are located on the server that is used to host the BizView applications. The host server must have Engine installed and running before client requests for XML views of business data can be returned. In general, each of the Connector act as front-end processors and do the following:

- receive the client application request specifying which BizView to invoke along with the parameters necessary for a successful execution,
- reorganize the information from the format provided in the message to the format required by the XAware API, and
- respond appropriately to the message with either the document returned from the Engine or the exception thrown when the execution was unsuccessful.

Each client application request is written differently depending on the type of connector, the BizView name, and required input parameters or input XML. The input parameters or input XML must be expected by the BizDocument, which means that some knowledge of the BizView files and the client system is important when building the connection between the client application requests and the BizView to be processed.

In addition, before client requests for XML views of business data can be accurately processed, the BizView files must be packaged and deployed. Any additional custom files relative to the BizView files must be deployed on the production server, including custom functoids, classes, and JavaScripts, to name a few.

What is the lifecycle of a BizView session?

The lifecycle of a BizView session starts at construction time, where the processing context is established and initialized.

- At *construction time*, the system establishes the BizView context with the JDOM structure. Input and output streaming is established, if appropriate. Then, input parameters and

input XML are established, if appropriate. The root context is pushed to the processing stack and executed.

- At *execution time*, the system processes the JDOM structure from the established processing context, cycling through the structure of the JDOM from the BizView. Each element in the structure is examined and the appropriate instructions are created for the element based on the element name and/or attribute name and value and each is executed as the processor loops through the stack, processing each instruction in the designed execution order.

After the JDOM structure has been executed, the BizView session returns the results to the request manager.

How does the Engine respond to requests?

Once you have designed and tested your BizView, you will deploy it to a host server to be available to respond to client applications. Before you deploy the BizView, it is important to understand how client applications will execute the BizView files. This section describes how BizViews respond to client application requests.

First, let's review the high-level process.

1. The client application sends a request to Engine, specifying a BizView and appropriate input parameters and/or XML to execute it successfully.
2. The XAware Engine locates the appropriate files in the deployed archives (*.xar) and loads the BizDocument, substituting the input parameters and XML with data passed in from the client and begins iterating through the instructions.
3. When encountered in the XML hierarchy, each BizComponent is processed and each BizDriver is used to establish a connection to the business data source.
4. The JDOM document (or exception if one is thrown) is returned to the Engine and passed back to the requesting application.

Now that you understand the overall process, let's discuss how Engine processes the client application requests. With Engine running on the host server (or your local machine as the case may be), and as an example of the HTTP Connector (XAServlet), you can type a simple URL into the browser's address line, like this.

```
http://localhost:8090/xaware/bizview/XAware/ContactInfo.xbd?Last-
Name=Smith?FirstName=*
```

In the example above, the URL—`http://localhost:8090`—is the host server where the BizView files are deployed. This URL refers to the local host server. The `'xaware/bizview/XAware/ContactInfo.xbd'` portion refers to the path and name of the service used in this example.

The BizDocument input parameters for the HTTP query string include the `?Parameter=Value`, where `?Parameter` is the name of the input parameter that will be passed from the BizDocument to the BizComponent at run-time; and `=Value` is the value of the input parameter that will be passed to the BizDocument. When executing BizDocuments in Designer, you can substitute a parameter value at run-time, but in a production environment, the client application will send the run-time input parameter values. In the example, two input parameters—`LastName` and `FirstName`—are sent with values in the client request.

How do you know which connector to use?

To build the connection between the client application and the BizView files, you must decide which connector to use. This section describes how to choose the right connector for your business situation. Remember, the client application determines the type of connector you will use, so it is important to be familiar with your client application as well.

Connector	Description
EJB	This connector is a stateless bean connector used by EJB client applications to process XAware BizViews on the Engine. Use this connector when you are writing an EJB client application and it requires you to connect to an EJB to process the XML view.
HTTP-Servlet	This connector is used by HTTP client applications to process data to and from a web server. Use this connector when you are using a browser to access the BizViews.
JMS ^a	This connector is used by JMS client applications that send (or publish) messages or request messages. Use this connector when you want to make asynchronous requests to the application server to execute BizViews.
SOAP	SOAP client applications use the SOAP connector to send an HTTP Post request, which includes a SOAP message that is defined in the body of the Post request. Use this connector when you want to make web service requests to Engine.
Batch Processing	Batch processing lets you automate the running of BizView services. Use this connector when you want to process entire sets of BizView services without requiring human intervention.

a. This connector is an asynchronous connector.



Tip: In addition, if you want to embed the XAware Engine into your client application, you can use the XAware API. See the *XAware Engine API* for information.



Note: A quick discussion on asynchronous versus synchronous Connector is important here. If a client application makes a request to a synchronous connector, the client will wait for a response. If a client application makes a request to an asynchronous connector, the client will continue with other processes until it receives a response. At the time a response is ready, the connector will notify the client application.

Information about forming client requests depending on the connector type is provided later in this document.

What about error handling with Connectors?

When a BizView executes on the XAware API, three return states must be recognized by the connector. The state is determined by the success/failure flag (called `wasSuccessful`) and the presence or absence of an exception thrown by the interface.

The states and desired behavior are described below:

Return State

Execution succeeded and `wasSuccessful` is set to `'true'`

Behavior

The resulting XML is returned to the calling client application.

Return State

Execution succeeded and `wasSuccessful` is set to `false`

Execution failed and an exception was propagated from the API

Behavior

This means that an error occurred in the BizView file and an error handler handled it there. The log viewer will indicate the BizDocument ended in failure.

The returned XML is the result of an error handler. The XML is assumed to contain all needed information for the calling client to detect the error. Some connectors can be configured to return an error condition in this case. This XML is returned as is, and the transport is not otherwise notified of an error.

This means an exception occurred that was not handled by any error handler, or an exception was explicitly thrown in the top-level BizView file.

The Java exception in its entirety is communicated back to the client in a transport-dependent manner, as described in the individual connector-specific chapters of this document.

How to set Parameters with Variables?

XAware provides a mechanism for connectors to communicate to and from their executing BizView files using variables. The variables let designers take connector-specific actions, such as asking the connector to respond with a SOAP fault when invoked as a SOAP service. The variables include global settings for the HTTP return error code, indications as to whether HTTP input or output streaming is to be used, and more.

The variables are configurable and JMX-managed as well, so certain behaviors can be customized for an installation, essentially overriding the configuration defined for a particular connector when it's necessary for the client. For example, if you know the client application will not want the result returned with pretty printing applied, then you can define the variable on the BizDocument to override the default connector setting.

A connector property can be accessed in the BizView files like this:

```
<outputEncoding value="$xavar:outputEncoding$" />
```

Or, to set a connector property, you can use the following functoid:

```
<prettyPrint value="$xaware:setSessionVariable(prettyPrint,raw)$" />
```



Chapter 2

Using the HTTP-Servlet Connector

This connector is used by HTTP client applications to process data to and from a web server. Use this connector when you are using a browser to access the BizViews.

Introduction	page 8
What is the format of the client request?	page 9
What are the configuration parameters for this connector?	page 11
How are errors handled with this connector?	page 14

Introduction

XAware's connector technology allows client applications to invoke a BizView file over one of several transports, one of which is described in this chapter. The connector software manages the requirements of the transport technology and invokes a BizView file, communicating responses and error conditions to the calling client using a mechanism appropriate for that technology. The XAware architecture provides a mechanism for connectors to communicate to and from their executing BizView files using variables, which allows designer to take connector-specific actions. These variables are configurable and JMX-managed as well, so certain behaviors can be customized for an installation.

This information describes how to use this connector type, including how to format the client request, what connector properties are available to you and how error handling is implemented.



Note: It is important to note that this connector also supports encrypted requests (HTTPS) using the same code.

Using the HTTP connector, a BizView can be invoked using either a GET or POST operation. Generally:

- a GET is used to invoke a BizDocument whose purpose it is to retrieve information from one or more data sources, and
- a POST is used to invoke a BizDocument whose purpose it is to send XML data into one or more data sources or processing resources.

Within this connector, the following steps are used to invoke a BizDocument:

1. Identify the BizView file
2. Identify the input parameters
3. Identify the input XML
4. Invoke the BizView file

What is the format of the client request?

The client request will be sent from the client application into the host server, where it will be used by Engine to initiate the defined BizView. A client request typically contains the BizView name and any input XML or input parameters required by the BizDocument for processing.

Invoking a BizView File

Your web client applications will use this connector to make HTTP requests including sending BizView processing commands to the Engine. This connector supports both HTTP Get and Post requests. The query string parameters define BizView input parameter values and are separated by the ampersand (&) character, and the names and values are separated by the equal (=) character.

```
http://hostserver:portnumber/xaware/bizview/path/file-
name.xbd?inputParameter1=value&inputParameter2=value&_XADATA=inputXML
```

The URL is broken down and described here:

- *hostserver:portnumber* is the name of the server hosting the BizView files and the port number to be used to access the BizView files
- *path/filename.xbd* is the fully qualified name of the BizView file to execute
- *?* delineates the query string parameters
- *inputParameter1=value* is the first input parameter and value that will be passed into the BizDocument
- *&* delineates the query string separator
- *_XADATA=inputXML* is the input XML required by the BizDocument



Note: While the intent is to specify deployed BizView files, it is also possible to specify an absolute reference on some platforms, but it requires a double slash after 'bizview'.

1. A BizView file is identified by the `httpPathInfo` variable, which is obtained by the system call, `HttpServletRequest.getPathInfo()`. The leading slash is stripped from this variable, which may be an alias that is translated by the XAware Engine.

See the following HTTP POST command (with the BizView file specified in bold):

```
http://localhost:8090/xaware/bizview/com/ps/getContact-Info.xbd?LastName=Smith&FirstName=*
```

2. Input parameters are specified by the `httpQueryString` variable, which is obtained by the system call `HttpServletRequest.getQueryString`.

See the following HTTP POST command (with the query string specified in bold):

```
http://localhost:8090/xaware/bizview/com/ps/getContact-Info.xbd?LastName=Smith&FirstName=*
```

Each key-value pair implies an input parameter and value to be passed into the identified BizView file.

3. Input XML should be in the body of the POST (not URL- encoded!). If the body does not appear to be XML, the connector will attempt to use the URL-encoded value of the query string parameter, `_XADATA`, which will be URL decoded and interpreted as XML and passed as input into the BizView file.

See the following HTTP POST command (with the input XML specified in bold):

```
http://localhost:8090/xaware/xaware/bizview/com/ps/purchaseOrder.xbd

<orderInfo>
  <repID>144</repID>
  <custID>W36325</custID>
  <orderNumber>W7499</orderNumber>
  <origDate>2007-03-31</origDate>
</orderInfo>
```

What are the configuration parameters for this connector?

Connector properties are one instance of variables that can be made available during processing—they are analogous to environment variables in many operating system environments. The connector properties are control parameters implemented as key-value pairs that are initialized by Spring, managed by JMX, and persisted across server executions.

They provide control over certain connector behaviors and can be used to communicate between a connector and a BizView file. Typical uses include specifying pretty print options and appropriate error handling behavior.

Connector parameter list

The following describes the configuration parameters for this connector.

Variable	Description
<code>errorTemplate</code>	Specifies the default error template used for the default error handler. This is triggered when an exception is raised from the XAware API. If not supplied, the following is the default: <pre><XAwareError><error>\$xavar:error\$</error></XAwareError></pre> The <code>\$xavar:error\$</code> variable is substituted with the error message returned by the BizDocument. The default error handler operates within the connector; it is therefore, beyond the point in time where the Engine processed the results. This is the single substitution that will be performed.
<code>httpBizViewExceptionCode</code>	Specifies a return code that can also be configured to be sent when an exception is propagated from the XAware API. used only when <code>propagateError=true</code> .
<code>httpBizViewFailureCode</code>	Specifies the HTTP return code in case of BizView file failure (<code>wasSuccessful=false</code>). Used only when <code>propagateError=true</code> .
<code>httpContextPath</code>	The result of calling <code>HttpServletRequest.getContextPath()</code> . Returns a portion of the request URI that indicates the context of the request. Usually <code>‘/xaware’</code> for a generic installation.
<code>httpHost</code>	Specifies the fully qualified domain name of the host returned by the system call <pre>java.net.InetAddress.getLocalHost()</pre>
<code>httpPathInfo</code>	The result of calling <code>HttpServletRequest.getPathInfo()</code> . Returns any extra path information associated with the URL the client sent when it made the request.
<code>httpPort</code>	Specifies the port to be used in the HTTP operation.
<code>httpQueryString</code>	The result of calling <pre>HttpServletRequest.getQueryString</pre> . Returns the query string that is contained in the request's URL after the path.

Variable	Description
httpServletPath	The result of calling <code>HttpServletRequest.getServletPath</code> . Returns the part of this request's URL that calls the servlet. Usually <code> '/bizview'</code> for a generic installation
invocationCommand	Specifies the invocation specification as received by the connector. For HTTP, this is the received URL.
outputEncoding	Specifies the encoding to use when serializing the JDOM structure. Note that this implies the transport encoding will be set to the same thing, where applicable, as in the case of an HTTP response. If the value is blank (zero length string), then a default encoding will be used. For HTTP, the default is UTF-8 if otherwise unspecified.
prettyPrint	Specifies the format of the output as a JDOM tree result or serialized XML. These are JDOM-defined configuration sets, as follows: <ul style="list-style-type: none"> • <code>raw</code> – prints the XML exactly as represented in the DOM with all whitespace preserved • <code>compact</code> – removes unneeded whitespace and results in the smallest possible XML • <code>pretty</code> – changes the whitespace to create indentation to match the hierarchy level
propagateError	Specifies whether the error received from the XAware API will cause the connector to propagate an error to the calling program in a connector-specific way. Two categories of errors can be propagated: <ul style="list-style-type: none"> • unhandled exceptions, and • BizView file failures. For HTTP, the error will result in a return code other than 200 (success).

Variable	Description
resourcePath	Specifies the resource path to pass to the XAware API. It is a semicolon-separated list of folders or archive (*.jar) files to search for the requested BizView file. This is generally not necessary since deployed archives are searched automatically, but this setting may be useful to specifically define references in archive files or to provide one or more locations to search outside the deployed archives.
urlParamMap	<p data-bbox="676 531 1325 627">Specifies the URL and parameter map such as the following: <code>urlParamMap=<servletPath>:<urlMatch>:<paramName>; [<urlMatch>:<paramName>]</code></p> <p data-bbox="676 647 1325 840">This feature enables a general URL rewrite capability which converts a segment from the URL into a key-value pair used as an input parameter. One common use case is REST-style conversion of a URL component into a <paramName>-value pair. For example, <code>urlParamMap=/bizview:/cust:/custNum</code> will translate the following:</p> <pre data-bbox="676 850 1290 904">www://xaware.com/xaware/bizview/cust/01234/com/xaware/cust.xbd</pre> <p data-bbox="676 917 1325 1014">and extract <code>/cust/01234</code> into a BizDocument where the input parameter, <code>cust</code>, equals a value of 01234 to invoke the following:</p> <pre data-bbox="676 1023 1305 1078">http://xaware.com/xaware/bizview/com/xaware/cust.xbd?custNum=01234</pre> <p data-bbox="676 1091 1325 1215">The <servletPath> is the value returned by <code>HttpServletRequest.getServletPath()</code>. A new servlet path can be configured with the servlet map element in <code>web.xml</code>, in a manner similar to the <code>/bizview</code> servlet map.</p>

See the next section for information about how errors are handled.

How are errors handled with this connector?

XAware can be configured to either propagate errors using the connector-specific technology, or translate the error into a default XML structure for return to the client application.⁷

Propagation of errors using HTTP codes is enabled by setting the `propagateErrors` as described:

- `propagateErrors=true` – means any exception received by the XAware API, or any BizView failure, will result in an HTTP error code being returned to the calling application rather than XML. For exceptions thrown by the XAware API, the return

code is defined by the connector property `httpBizViewExceptionCode`. A BizView failure occurs when the top-level error handler is invoked. In this case, the code defined by the connector property `httpBizViewFailureCode` is returned.

- `propagateErrors=false` – means the connector will always return XML. In the case of a BizView failure, the XML will be the result of a BizView file after invocation of the top level error handler. In the case of an exception thrown by the XAware API, a default error XML structure (defined in the connector property `defaultErrorTemplate`) is returned. A sample default error structure is shown here:

```
<Document>
  <Error>
    <detail>Error in executing BizDocument [BizDoc1.xbd]:
Failed to load BizComponent. Document file not found:
C:\Dev\XAware50\server\build\xahome\undefined</detail>
  </Error>
</Document>
```



Note: See “What about error handling with Connectors?” on page 5 for general error handling information.

This concludes the information available for the HTTP-Servlet Connector.



Chapter 3

Using the SOAP Connector

Web service applications use the SOAP connector to send an HTTP Post request, which includes a SOAP message that is defined in a body of the Post request. Use this connector when you want to use Engine to execute BizViews through a web service.

Introduction	page 18
What is the format of the client request?	page 18
What does the client application code look like?	page 18
What are the configuration parameters for this connector?	page 25
How are errors handled with this connector?	page 27

Introduction

XAware's connector technology allows client applications to invoke a BizView file over one of several transports, one of which is described in this chapter. The connector software manages the requirements of the transport technology and invokes a BizView file, communicating responses and error conditions to the calling client using a mechanism appropriate for that technology. The XAware architecture provides a mechanism for connectors to communicate to and from their executing BizView files using variables, which allows designer to take connector-specific actions. These variables are configurable and JMX-managed as well, so certain behaviors can be customized for an installation.

This information describes how to use this connector type, including how to format the client request, what connector properties are available to you and how error handling is implemented.

Using the SOAP connector, the following styles support the invocation of BizView files:

- Document Literal (Unwrapped)
- Document Literal (Wrapped)
- RPC/Literal
- RPC/Encoded

See the next section to format the client requests.

What is the format of the client request?

The client request will be sent from the client application into the host server, where it will be used by Engine to initiate the defined BizView. A client request typically contains the BizView name and any input XML or input parameters required by the BizDocument for processing.

In Designer, you can easily create a web services file using the project archive (.jar) file. This is a quick process that is dependent upon an existing BizDocument. You should have a complete and fully tested BizDocument before you attempt to create a new WSDL file. Information on making WSDL files is available in the Designer Help system.

What does the client application code look like?

With the use of the SOAP connector, the client application code must build and send a SOAP message as specified by the WSDL. You will generally use a tool which reads the WSDL to determine the SOAP input and output message formats (e.g., WTP). You will generate SOAP messages similar to one of the following and use it where a call to the Engine to initiate a BizView is required.

Sample Document Literal Request

The following demonstrates the message created for a document literal request type.

```

1  POST /xaware/XADocSoapServlet HTTP/1.1
2  SOAPAction: "http://xaware.com/BizService"
3  Connection: close
4  User-Agent: Jakarta Commons-HttpClient/2.0final
5  Host: localhost:8070
6  Content-Length: 361
7  Content-Type: text/xml; charset="UTF-8"
8
9  <SOAPENV:Envelope xmlns:SOAPENV="http://schemas.xmlsoap.org/soap/enve-
   lope/">
10     <SOAPENV:Body>
11         <type:SOAP.retrieveLandscapers.xbd xmlns:type="http://
   xaware.org/xas/ns1/type">
12             <type:state>Wyoming</type:state>
13         </type:SOAP.retrieveLandscapers.xbd>
14     </SOAPENV:Body>
15 </SOAPENV:Envelope>

```

The following results are returned to the SOAP client application using the sample client application code above. The data returned contains only those landscapers that match the input parameter value, Wyoming.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <SOAPENV:Envelope xmlns:SOAPENV="http://schemas.xmlsoap.org/soap/enve-
   lope/"><SOAPENV:Body><type:SOAP.retrieveLandscapers.xbdResponse
   xmlns:type="http://xaware.org/xas/ns1/type"><landscapers>
3
4
5     <XMLdata>
6         <Element>
7             <NAME>Thomas Landscape Architecture</NAME>
8             <PHONE>3073688596</PHONE>
9             <CITYSTATE>Bitterroot Springs, Wyoming</CITYSTATE>
10            <BIZRATE>3</BIZRATE>
11        </Element>
12    </XMLdata><XMLdata>
13        <Element>
14            <NAME>Grumman Landscaping</NAME>

```

```

15         <PHONE>3072578578</PHONE>
16         <CITYSTATE>Bitterroot Springs, Wyoming</CITYSTATE>
17         <BIZRATE>45</BIZRATE>
18     </Element>
19 </XMLdata><XMLdata>
20     <Element>
21         <NAME>Higher Ground Designs</NAME>
22         <PHONE>3078575875</PHONE>
23         <CITYSTATE>Bitterroot Springs, Wyoming</CITYSTATE>
24         <BIZRATE>3</BIZRATE>
25     </Element>
26 </XMLdata><XMLdata>
27     <Element>
28         <NAME>Organic Landscaping</NAME>
29         <PHONE>3075872547</PHONE>
30         <CITYSTATE>Bitterroot Springs, Wyoming</CITYSTATE>
31         <BIZRATE>5</BIZRATE>
32     </Element>
33 </XMLdata><XMLdata>
34     <Element>
35         <NAME>Design by Nature Landscaping</NAME>
36         <PHONE>3073658572</PHONE>
37         <CITYSTATE>Bitterroot Springs, Wyoming</CITYSTATE>
38         <BIZRATE>2</BIZRATE>
39     </Element>
40 </XMLdata><XMLdata>
41     <Element>
42         <NAME>Land Patterns</NAME>
43         <PHONE>3075683642</PHONE>
44         <CITYSTATE>Bitterroot Springs, Wyoming</CITYSTATE>
45         <BIZRATE>3</BIZRATE>
46     </Element>
47 </XMLdata><XMLdata>
48     <Element>
49         <NAME>Sundown Landscaping</NAME>
50         <PHONE>3075872596</PHONE>
51         <CITYSTATE>Bitterroot Springs, Wyoming</CITYSTATE>
52         <BIZRATE>4</BIZRATE>
53     </Element>

```

```

54         </XMLdata>
55
56 </landscapers></type:SOAP.retrieveLandscapers.xbdResponse></SOAP-
    ENV:Body></SOAPENV:Envelope>

```

Sample RPC Encoded Request Type

The following demonstrates the message created for an RPC Encoded request type.

```

1  POST /xaware/XARpcEncodedSoapServlet HTTP/1.1
2  SOAPAction: http://xaware.org/xas/ns1/rpcEncodedService/action/SOAP/
    retrieveLandscapers.xbd
3  Connection: close
4  User-Agent: Jakarta Commons-HttpClient/2.0final
5  Host: localhost:8070
6  Content-Length: 581
7  Content-Type: text/xml; charset="UTF-8"
8
9  <SOAPENV:Envelope xmlns:SOAPENV="http://schemas.xmlsoap.org/soap/enve-
    lope/">
10         <SOAPENV:Body xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    SOAPENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
11             <mes:SOAP.retrieveLandscapers.xbd xmlns:mes="http://
    xaware.org/xas/ns1/message">
12                 <state xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="xs:string">Wyoming</state>
13             </mes:SOAP.retrieveLandscapers.xbd>
14         </SOAPENV:Body>
15     </SOAPENV:Envelope>

```

The following results are returned to the SOAP client application using the sample client application code above. The data returned contains only those landscapers that match the input parameter value, Wyoming.

```

1  HTTP/1.1 200 OK
2  Server: Apache-Coyote/1.1
3  X-Powered-By: Servlet 2.4; JBoss-4.0.5.GA (build: CVSTag=Branch_4_0
    date=200610162339)/Tomcat-5.5
4  Content-Type: text/xml
5  Content-Length: 2247
6  Date: Fri, 08 Dec 2006 18:32:53 GMT

```

```

7 Connection: close
8
9 <?xml version="1.0" encoding="UTF-8"?>
10 <SOAPENV:Envelope xmlns:SOAPENV="http://schemas.xmlsoap.org/soap/enve-
    lope/"><SOAPENV:Body><mes:SOAP.retrieveLandscapers.xbdResponse
    xmlns:mes="http://xaware.org/xas/ns1/message"><Result><landscapers>
11
12
13     <XMLdata>
14         <Element>
15             <NAME>Thomas Landscape Architecture</NAME>
16             <PHONE>3073688596</PHONE>
17             <CITYSTATE>Bitterroot Springs, Wyoming</CITYSTATE>
18             <BIZRATE>3</BIZRATE>
19         </Element>
20     </XMLdata><XMLdata>
21         <Element>
22             <NAME>Grumman Landscaping</NAME>
23             <PHONE>3072578578</PHONE>
24             <CITYSTATE>Bitterroot Springs, Wyoming</CITYSTATE>
25             <BIZRATE>45</BIZRATE>
26         </Element>
27     </XMLdata><XMLdata>
28         <Element>
29             <NAME>Higher Ground Designs</NAME>
30             <PHONE>3078575875</PHONE>
31             <CITYSTATE>Bitterroot Springs, Wyoming</CITYSTATE>
32             <BIZRATE>3</BIZRATE>
33         </Element>
34     </XMLdata><XMLdata>
35         <Element>
36             <NAME>Organic Landscaping</NAME>
37             <PHONE>3075872547</PHONE>
38             <CITYSTATE>Bitterroot Springs, Wyoming</CITYSTATE>
39             <BIZRATE>5</BIZRATE>
40         </Element>
41     </XMLdata><XMLdata>
42         <Element>
43             <NAME>Design by Nature Landscaping</NAME>
44             <PHONE>3073658572</PHONE>
45             <CITYSTATE>Bitterroot Springs, Wyoming</CITYSTATE>
46             <BIZRATE>2</BIZRATE>

```

```

47         </Element>
48     </XMLdata><XMLdata>
49         <Element>
50             <NAME>Land Patterns</NAME>
51             <PHONE>3075683642</PHONE>
52             <CITYSTATE>Bitterroot Springs, Wyoming</CITYSTATE>
53             <BIZRATE>3</BIZRATE>
54         </Element>
55     </XMLdata><XMLdata>
56         <Element>
57             <NAME>Sundown Landscaping</NAME>
58             <PHONE>3075872596</PHONE>
59             <CITYSTATE>Bitterroot Springs, Wyoming</CITYSTATE>
60             <BIZRATE>4</BIZRATE>
61         </Element>
62     </XMLdata>
63
64 </landscapers></Result></mes:SOAP.retrieveLandscapers.xbdResponse></
    SOAPENV:Body></SOAPENV:Envelope>

```

Sample RPC Literal Request Type

The following demonstrates the message created for an RPC Literal request type.

```

1  POST /xaware/XARpcLitSoapServlet HTTP/1.1
2  SOAPAction: "http://xaware.org/xas/ns1/rpcLitService/action/SOAP/
    retrieveLandscapers.xbd"
3  Connection: close
4  User-Agent: Jakarta Commons-HttpClient/2.0final
5  Host: localhost:8070
6  Content-Length: 351
7  Content-Type: text/xml; charset="UTF-8"
8
9  <SOAPENV:Envelope xmlns:SOAPENV="http://schemas.xmlsoap.org/soap/enve-
    lope/">
10     <SOAPENV:Body>
11         <mes:SOAP.retrieveLandscapers.xbd xmlns:mes="http://
            xaware.org/xas/ns1/message">
12             <state>Wyoming</state>
13         </mes:SOAP.retrieveLandscapers.xbd>
14     </SOAPENV:Body>
15 </SOAPENV:Envelope>

```

The following results are returned to the SOAP client application using the sample client application code above. The data returned contains only those landscapers that match the input parameter value, Wyoming.

```

1  HTTP/1.1 200 OK
2  Server: Apache-Coyote/1.1
3  X-Powered-By: Servlet 2.4; JBoss-4.0.5.GA (build: CVSTag=Branch_4_0
   date=200610162339)/Tomcat-5.5
4  Content-Type: text/xml
5  Content-Length: 2247
6  Date: Fri, 08 Dec 2006 18:36:14 GMT
7  Connection: close
8
9  <?xml version="1.0" encoding="UTF-8"?>
10 <SOAPENV:Envelope xmlns:SOAPENV="http://schemas.xmlsoap.org/soap/enve-
   lope/"><SOAPENV:Body><mes:SOAP.retrieveLandscapers.xbdResponse
   xmlns:mes="http://xaware.org/xas/ns1/message"><Result><landscapers>
11
12
13     <XMLdata>
14         <Element>
15             <NAME>Thomas Landscape Architecture</NAME>
16             <PHONE>3073688596</PHONE>
17             <CITYSTATE>Bitterroot Springs, Wyoming</CITYSTATE>
18             <BIZRATE>3</BIZRATE>
19         </Element>
20     </XMLdata><XMLdata>
21         <Element>
22             <NAME>Grumman Landscaping</NAME>
23             <PHONE>3072578578</PHONE>
24             <CITYSTATE>Bitterroot Springs, Wyoming</CITYSTATE>
25             <BIZRATE>45</BIZRATE>
26         </Element>
27     </XMLdata><XMLdata>
28         <Element>
29             <NAME>Higher Ground Designs</NAME>
30             <PHONE>3078575875</PHONE>
31             <CITYSTATE>Bitterroot Springs, Wyoming</CITYSTATE>
32             <BIZRATE>3</BIZRATE>
33         </Element>
34     </XMLdata><XMLdata>
35         <Element>

```

```

36         <NAME>Organic Landscaping</NAME>
37         <PHONE>3075872547</PHONE>
38         <CITYSTATE>Bitterroot Springs, Wyoming</CITYSTATE>
39         <BIZRATE>5</BIZRATE>
40     </Element>
41 </XMLdata><XMLdata>
42     <Element>
43         <NAME>Design by Nature Landscaping</NAME>
44         <PHONE>3073658572</PHONE>
45         <CITYSTATE>Bitterroot Springs, Wyoming</CITYSTATE>
46         <BIZRATE>2</BIZRATE>
47     </Element>
48 </XMLdata><XMLdata>
49     <Element>
50         <NAME>Land Patterns</NAME>
51         <PHONE>3075683642</PHONE>
52         <CITYSTATE>Bitterroot Springs, Wyoming</CITYSTATE>
53         <BIZRATE>3</BIZRATE>
54     </Element>
55 </XMLdata><XMLdata>
56     <Element>
57         <NAME>Sundown Landscaping</NAME>
58         <PHONE>3075872596</PHONE>
59         <CITYSTATE>Bitterroot Springs, Wyoming</CITYSTATE>
60         <BIZRATE>4</BIZRATE>
61     </Element>
62 </XMLdata>
63
64 </landscapers></Result></mes:SOAP.retrieveLandscapers.xbdResponse></
    SOAPENV:Body></SOAPENV:Envelope>

```

What are the configuration parameters for this connector?

Connector properties are one instance of variables that can be made available during processing—they are analogous to environment variables in many operating system environments. The connector properties are control parameters implemented as key-value pairs that are initialized by Spring, managed by JMX, and persisted across server executions. They provide control over certain connector behaviors and can be used to communicate between a connector and a BizView file. Typical uses include specifying pretty print options and appropriate error handling behavior.

Connector parameter list

The following describes the configuration parameters for this connector.

Variable	Description
<code>generateSoapFault</code>	Using the true or false values, this parameter is used for SOAP communication. It can be set from a BizView file to signal to the SOAP connector that the returned body should be packaged as a SOAP fault message.
<code>httpContextPath</code>	The result of calling <code>HttpServletRequest.getContextPath()</code> . Returns a portion of the request URI that indicates the context of the request. Usually <code>/xaware</code> for a generic installation.
<code>httpHost</code>	Specifies the fully qualified domain name of the host returned by the system call <code>java.net.InetAddress.getLocalHost()</code> .
<code>httpPathInfo</code>	The result of calling <code>HttpServletRequest.getPathInfo()</code> . Returns any extra path information associated with the URL the client sent when it made the request.
<code>httpPort</code>	Specifies the port to be used in the HTTP operation.
<code>httpQueryString</code>	The result of calling <code>HttpServletRequest.getQueryString</code> . Returns the query string that is contained in the request's URL after the path.
<code>httpServletPath</code>	The result of calling <code>HttpServletRequest.getServletPath</code> . Returns the part of this request's URL that calls the servlet. Usually <code>/bizview</code> for a generic installation
<code>invocationCommand</code>	Specifies the invocation specification as received by the connector. For SOAP, this is the received URL.
<code>outputEncoding</code>	Specifies which encoding to use when serializing the JDOM structure. This implies the transport encoding is set to the same in the case of an HTTP response.

Variable	Description
<code>prettyPrint</code>	<p>Specifies the format of the output as a JDOM tree result or serialized XML. These are JDOM-defined configuration sets, as follows:</p> <ul style="list-style-type: none"> • <code>raw</code> – prints the XML exactly as represented in the DOM with all whitespace preserved • <code>compact</code> – removes unneeded whitespace and results in the smallest possible XML • <code>pretty</code> – changes the whitespace to create indentation to match the hierarchy level
<code>propagateError</code>	<p>Specifies whether the error received from the XAware API will cause the connector to propagate an error to the calling program in a connector-specific way. Two categories of errors can be propagated:</p> <ul style="list-style-type: none"> • unhandled exceptions, and • BizView file failures. <p>For SOAP, a SOAP fault is generated.</p>
<code>soapHeaderSelector</code>	<p>Using the <code>echo</code>, <code>all</code>, or <code>XPath</code> values, this parameter is used to determine which SOAP headers, if any, should be returned in the response. The behavior is the same as the servlet initialization parameters.</p>

See the next section for information about how errors are handled.

How are errors handled with this connector?

XAware can be configured to either propagate errors using the connector-specific technology, or translate the error into a default XML structure for return to the client application. Any unhandled exception thrown from the XAware API will be translated into a SOAP fault message. The BizView file can also set a connector property, `generateSoapFault=true`, to signal the connector that the result is to be packaged as a SOAP fault.



Note: See “What about error handling with Connectors?” on page 5 for general error handling information.

This concludes the information available about the SOAP connector.



Chapter 4

Using the JMS Connector

Java Message Service (JMS) is the API standard for accessing message-oriented middle ware from within Java applications. It is a programming model that can be used on both the client and the server side, so the development of interactive, mobile systems is greatly simplified.

Introduction	page 30
What does the client application code look like?	page 30
What are the configuration parameters for this connector?	page 34
How are errors handled with this connector?	page 35

Introduction

XAware's connector technology allows client applications to invoke a BizView file over one of several transports, one of which is described in this chapter. The connector software manages the requirements of the transport technology and invokes a BizView file, communicating responses and error conditions to the calling client using a mechanism appropriate for that technology. The XAware architecture provides a mechanism for connectors to communicate to and from their executing BizView files using variables, which allows designer to take connector-specific actions. These variables are configurable and JMX-managed as well, so certain behaviors can be customized for an installation.

This information describes how to use this connector type, including how to format the client request, what connector properties are available to you and how error handling is implemented.



Tip: This connector require you to deploy the `xaware.ear` and `xaware.war` as described in the deploying guides. See “Related Documentation” on page v for information.

What does the client application code look like?

The following JMS client application request is accessing the BizView called `poView`. This BizView expects a `customer_name` parameter. The message bean listens on the topic, `XATopicRequest`, for a map message, which can contain up to three name-value pairs. When the message bean receives a request, it uses the `messageID` referenced in the `JMSCorrelationID` on the response. You will write code similar to the following and include it in your client code where a call to the Engine for the BizView is required.



Note: The following example includes the client code to produce a JMS publish and receive. In your client application, you will typically use only one or the other, but not both.

Implementing the Message Sender

The following sample Java client application demonstrates the message sender implementation.

```

1      public void sendMessage(String sJmsJndiFactory, String sJmsUrl,
2          String sJmsConnectionFactory)
3          throws NamingException, JMSEException {
4
5          // These two lines indicate the topic names on which the message
6          // will be published
7          String sTopicRequestName = "XAwareTopicRequest";
8          String sTopicResponseName = "XAwareTopicResponse";
9
10         // Finds the TopicConnectionFactory via JNDI lookup
11         Hashtable env = new Hashtable();
12         env.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY, sJmsJndi-
13             Factory);
14         env.put(javax.naming.Context.PROVIDER_URL, sJmsUrl);
15         javax.naming.Context ctx = new InitialContext(env);
16         TopicConnectionFactory topicConFactory =
17             (TopicConnectionFactory) ctx.lookup(sJmsConnectionFactory);
18
19         // Creates a subscriber and start a listener
20         TopicConnection subConn = topicConFactory.createTopicConnec-
21             tion();
22         TopicSession subSession =
23             subConn.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
24         Topic subTopic = (Topic) ctx.lookup(sTopicResponseName);
25         TopicSubscriber tsubscriber = subSession.createSubscriber(sub-
26             topic);
27         tsubscriber.setMessageListener(this);
28         subConn.start();
29
30         // Creates a publisher
31         TopicConnection pubConn = topicConFactory.createTopicConnec-
32             tion();
33         TopicSession pubSession =
34             pubConn.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
35         Topic pubTopic = (Topic) ctx.lookup(sTopicRequestName);
36         TopicPublisher topicPublisher = pubSession.createPublisher(pub-
37             Topic);
38         pubConn.start();
39
40         // Sends a map message to process a BizView
41         // and returns the results as XML
42         MapMessage msg = pubSession.createMapMessage();

```

```

37      // Defines the input parameters to be passed into the BizDocument
38      String sInputParam =
39          "<myinput><customer_name>ACME</customer_name></myinput>";
40      // Defines the '_BIZVIEW' portion of the client request
41      // and the BizView to execute
42      msg.setString("_BIZVIEW", "poView");
43      // Defines the '_INPUTPARAM' portion of the client request
44      // and references sInputParam
45      msg.setString("_INPUTPARAM", sInputParam);
46      topicPublisher.publish(msg);
47  }

```

The sample code is described in the following table.

Line(s)	Description
1	Declares the first function in the sample code, and defines the function that will send the JMS request. The caller must provide the JNDI names used to access the TopicConnectionFactory on the JMS server.
6–7	These lines indicate the topic names on which the JMS connector will publish and receive requests.
10–15	These lines perform a JNDI lookup to find the TopicConnectionFactory.
18–24	These lines create a TopicConnection and TopicSubscriber to receive messages from the response topic. They also establish the enclosing class (which implements <code>javax.jms.MessageListener</code>) as a <code>MessageListener</code> on this topic, so that the <code>onMessage()</code> method is triggered for response messages.
27–32	These lines create a TopicConnection and TopicPublisher to send messages to the request topic.
36	Creates a <code>MapMessage</code> for sending a new request message.
38	Indicates the input parameter content that will be sent into the BizDocument as input. The format of the XML must match the format expected by the BizDocument. The <code>sInputParam</code> identifier will be used later in the sample to reference these input parameters.
42	Defines the '_BIZVIEW' portion of the client request, including the name of the BizView to be executed using this sample code.

Line(s)	Description
45	Defines the <code>'_INPUTPARAM'</code> portion of the client request, including a reference to the input parameter string identifier.
46	Defines the action this connector will take, which is to publish a message to the JMS request topic.

Implementing the Message Listener

The following sample java client application demonstrates the message listener.

```

1  public void onMessage(Message msg) {
2      String msgText = null;
3      try {
4          if (msg instanceof TextMessage) {
5              msgText = ((TextMessage) msg).getText();
6          } else {
7              msgText = msg.toString();
8          }
9      } catch (JMSEException e) {
10         System.err.println(
11             "Exception encountered reading TextMessage: " + e);
12         return;
13     }
14     StringReader reader = new StringReader(msgText);
15     SAXBuilder builder = new SAXBuilder();
16     Document xaDoc = null;
17     try {
18         xaDoc = builder.build(reader);
19     } catch (JDOMException e) {
20         System.err.println(
21             "Exception encountered parsing message into XML: " + e);
22         return;
23     }
24 }

```

The sample code is described in the following table.

Line(s)	Description
1	Defines the function that will receive a JMS response message. This method is declared in the <code>javax.jms.MessageListener</code> interface. Because this class is registered as a <code>MessageListener</code> on the response topic in the previous example, this method is triggered when a response message is received.
2–13	These lines get the message text from the <code>Message</code> .
14–23	These lines build a <code>JDOM Document</code> from the XML contained within the message text. Application-specific processing of the response XML would follow.

What are the configuration parameters for this connector?

Connector properties are one instance of variables that can be made available during processing—they are analogous to environment variables in many operating system environments. The connector properties are control parameters implemented as key-value pairs that are initialized by Spring, managed by JMX, and persisted across server executions. They provide control over certain connector behaviors and can be used to communicate between a connector and a BizView file. Typical uses include specifying pretty print options and appropriate error handling behavior.

Connector parameter list

The following describes the configuration parameters for this connector.

Variable	Description
<code>errorTemplate</code>	<p>Specifies the default error template used for the default error handler. This is triggered when an exception is raised from the XAware API. If not supplied, the following is the default:</p> <pre><XAwareError><error>\$xavar:error\$</error></XAwareError></pre> <p>The <code>\$xavar:error\$</code> variable is substituted with the error message returned by the <code>BizDocument</code>. The default error handler operates within the connector; it is therefore, beyond the point in time where the Engine processed the results. This is the single substitution that will be performed.</p>
<code>invocationCommand</code>	<p>Specifies the invocation specification as received by the connector. For JMS, it is the BizView file specification.</p>

Variable	Description
<code>outputEncoding</code>	Specifies which encoding to use when serializing the JDOM structure. This implies the transport encoding is set to the same in the case of an HTTP response.
<code>prettyPrint</code>	Specifies the format of the output as a JDOM tree result or serialized XML. These are JDOM-defined configuration sets, as follows: <ul style="list-style-type: none"> • <code>raw</code> – prints the XML exactly as represented in the DOM with all whitespace preserved • <code>compact</code> – removes unneeded whitespace and results in the smallest possible XML • <code>pretty</code> – changes the whitespace to create indentation to match the hierarchy level

See the next section for information about how errors are handled.

How are errors handled with this connector?

XAware can be configured to either propagate errors using the connector-specific technology, or translate the error into a default XML structure for return to the client application. No special error communication is available on this interface. The connector will create a default error XML structure to return any propagated exceptions to the calling client.



Note: See “What about error handling with Connectors?” on page 5 for general error handling information.

This concludes the information available about the JMS connector.



Chapter 5

Using the EJB Connector

This connector is a stateless bean connector used by EJB client applications to process XAware BizViews on the Engine. You will use this connector when you are writing a Java client application and it requires you to connect to an EJB to process the XML view.

Introduction	page 38
What is the format of the client request?	page 38
What does the client application code look like?	page 39
What are the configuration parameters for this connector?	page 41
How are errors handled with this connector?	page 42

Introduction

XAware's connector technology allows client applications to invoke a BizView file over one of several transports, one of which is described in this chapter. The connector software manages the requirements of the transport technology and invokes a BizView file, communicating responses and error conditions to the calling client using a mechanism appropriate for that technology. The XAware architecture provides a mechanism for connectors to communicate to and from their executing BizView files using variables, which allows designer to take connector-specific actions. These variables are configurable and JMX-managed as well, so certain behaviors can be customized for an installation.

This information describes how to use this connector type, including how to format the client request, what connector properties are available to you and how error handling is implemented.



Tip: This connector require you to deploy the `xaware.ear` and `xaware.war` as described in the deploying guides. See “Related Documentation” on page v for information.

The EJB connector is specifically defined using deployment descriptors to work with the JBoss server. If you want to use this connector on a server that is not JBoss, you will have to request a Professional Services engagement. See “Contacting XAware” on page vi for information.

What is the format of the client request?

The client request will be sent from the client application into the host server, where it will be used by Engine to initiate the defined BizView. A client request typically contains the BizView name and any input XML or input parameters required by the BizDocument for processing.

The client request will be sent from the client application into the host server, where it will be initiated by Engine and executed. The EJB connector uses the following client request to initiate the BizView.

```
public string executeBizDoc ("BizView Name", "input XML", "input parameters")
```

Where *BizView Name* is the logical name of the BizView (as it appears in the `XASystem.xml` file; *input XML* is the input XML to pass from the client application into the BizDocument; and *input parameters* are any required input parameters to pass from the client application into the BizDocument.

You will put your input parameters in XML format, as shown here.

```
public string executeBizDoc ("Customers", "", "<xaInput><first>Jane</first><last>Doe</last></xaInput>")
```

In this example client request, the BizView name is Customers. Notice in this example, no input XML is required, so the client request contains an empty string for that value. Two input parameters—first and last—are being passed with values in this example.

See the following client request example.

```
public string executeBizDoc ("PurchaseOrders", "<purchaseOrder><order><item><itemNumber>2325</itemNumber></item></order></purchaseorder>", "")
```

In this example client request, the BizView name is PurchaseOrders. In this example, input XML will be passed into the BizView as input. No input parameters are expected by the BizDocument, so an empty string value is included for that parameter of the client request.

What does the client application code look like?

The client application code is developed by you and used to connect to the Engine and execute XML views of data. Essentially, the client application is developed using the J2EE APIs to connect to an XAware EJB which will process the request and return back XML. XAware provides the EJB on the Engine, which is used to process the data.

You will write code similar to the following and include it in your client code where a call to the Engine for the BizView is required. See the following sample code.

```
1 // XAware EJB JNDI name
2 String sEjbJndiName = "xaejb";
3 // JNDI factor name
4 String sEjbJndiFactory = "org.jnp.interfaces.NamingContextFactory";
5 // URL for the application server
6 String sEjbUrl = "jnp://localhost:1099";
7 // Use to indicate the BizView host server IP and port
8 Properties p = new Properties();
9 p.put("java.naming.factory.initial", sEjbJndiFactory);
10 p.put("java.naming.provider.url", sEjbUrl)
11 // Create an initial context
12 Context ctx = new InitialContext(p);
13 // Hook up EJB
```

```

14 XAejbHome xaHome = (XAejbHome)ctx.lookup(sejbJndiName);
15 // Create EJB instance
16 XAejb purchaseejb= xaHome.create();
17 // Use to indicate the input parameters required by the BizView
18 String inputparam= "<myinput><username>John</username><password>Smith</
    password></myinput>";
19 // Use to invoke the EJB from the application server
20 String responseXML = purchaseejb.executeBizDoc("purchaseOrderView", "",
    inputparam);
21 // Use as the client request to initiate the purchaseOrderView BizView

```

The sample code is described in the following table.

Line(s)	Description
2	Sets the Java Naming and Directory Interface (JNDI) name to <code>xaejb</code> . JNDI enables Java-based applications to access multiple naming and directory services. You access the XAware EJB using the unique JNDI name, <code>xaejb</code> .
4	It is important to remember that XAware is delivered with the JBoss server, so this line would be modified if you are using another application server or web server, such as BEA.
6	Indicates the Engine location, which must be defined in the client application code. This value defines the location of the Jboss server included with your XAware installation. This value indicates using a local host, but when you build similar code for your client application, you will insert your host server's IP address and port instead. You will also modify the value if you were using another server, such as a BEA WebLogic application server. Using BEA application server as an example, you might modify the value as shown here. <pre>String sejbUrl = "t3://localhost:7001";</pre> The value shown here is <i>similar</i> to what you will use. Do not use this exact value.
8–10	Takes the factory name and URL and puts them into a format commonly used by the Context object.
12–14	Calls the <code>xaejb.executeBizDoc</code> method and returns the response (or BizDocument results). The BizView name is <code>purchaseOrderView</code> , the empty string indicates that no input XML will be sent with this client request, and the input parameters will be sent in XML format with values assigned to be passed to the BizDocument for execution.

Line(s)	Description
18	Indicates the input parameters required by the BizView, which must be defined in the client application code. The client request will include the input parameters in XML format with values to be passed into the BizDocument and used in the BizComponent's execution.
20	Used to invoke the EJB from the application server.

What are the configuration parameters for this connector?

Connector properties are one instance of variables that can be made available during processing—they are analogous to environment variables in many operating system environments. The connector properties are control parameters implemented as key-value pairs that are initialized by Spring, managed by JMX, and persisted across server executions. They provide control over certain connector behaviors and can be used to communicate between a connector and a BizView file. Typical uses include specifying pretty print options and appropriate error handling behavior.

Connector parameter list

The following describes the configuration parameters for this connector.

Variable	Description
<code>errorTemplate</code>	Specifies the default error template used for the default error handler. This is triggered when an exception is raised from the XAware API. If not supplied, the following is the default: <pre><XAwareError><error>\$xavar:error\$</error></XAwareError></pre> The <code>\$xavar:error\$</code> variable is substituted with the error message returned by the BizDocument. The default error handler operates within the connector; it is therefore, beyond the point in time where the Engine processed the results. This is the single substitution that will be performed.
<code>invocationCommand</code>	Specifies the invocation specification as received by the connector. For EJB, it is the BizView file specification.
<code>outputEncoding</code>	Specifies which encoding to use when serializing the JDOM structure. This implies the transport encoding is set to the same in the case of an HTTP response.

Variable	Description
<code>prettyPrint</code>	<p>Specifies the format of the output as a JDOM tree result or serialized XML. These are JDOM-defined configuration sets, as follows:</p> <ul style="list-style-type: none"> • <code>raw</code> – prints the XML exactly as represented in the DOM with all whitespace preserved • <code>compact</code> – removes unneeded whitespace and results in the smallest possible XML • <code>pretty</code> – changes the whitespace to create indentation to match the hierarchy level
<code>propagateError</code>	<p>Specifies whether the error received from the XAware API will cause the connector to propagate an error to the calling program in a connector-specific way. Two categories of errors can be propagated:</p> <ul style="list-style-type: none"> • unhandled exceptions, and • BizView file failures.

See the next section for information about how errors are handled.

How are errors handled with this connector?

XAware can be configured to either propagate errors using the connector-specific technology, or translate the error into a default XML structure for return to the client application. An exception thrown from the API will be propagated to the calling application if `propagateError="true"`.



Note: See “What about error handling with Connectors?” on page 5 for general error handling information.

This concludes the information available for the EJB connector.



Chapter 6

Batch Processing

It is often convenient to process your BizViews using a command line program. XAware provides a command line program (for Windows and Linux/Unix systems) that lets you invoke BizView files from a Windows or Linux/Unix command line. This section describes how to implement the command line program and use it.

Introduction

Common uses of command line processes are to execute entire sets of BizView files without having to be present during the processing. One primary purpose for creating such a process is to have it established so that a scheduled task can initiate the execution of the batch process. Commonly, this is done at a time convenient to other business activities, such as at night when demand for computer resources is low.

The XAware script invokes the XAware Engine in batch mode. The Engine has direct access to all deployed BizViews. In addition, it may have direct access to the project located in the current working directory.

The instructions in this chapter describe how use a command line program provided with the installation. The procedures for setting up scheduled tasks are operating system-specific and therefore beyond the scope of this document. Refer to your available documentation or consult a system administrator for the appropriate instructions and information.

Setting up

The XAware batch script is available in the *<Install Directory>/bin* folder of your install. The use of the batch script requires these variables to be set:

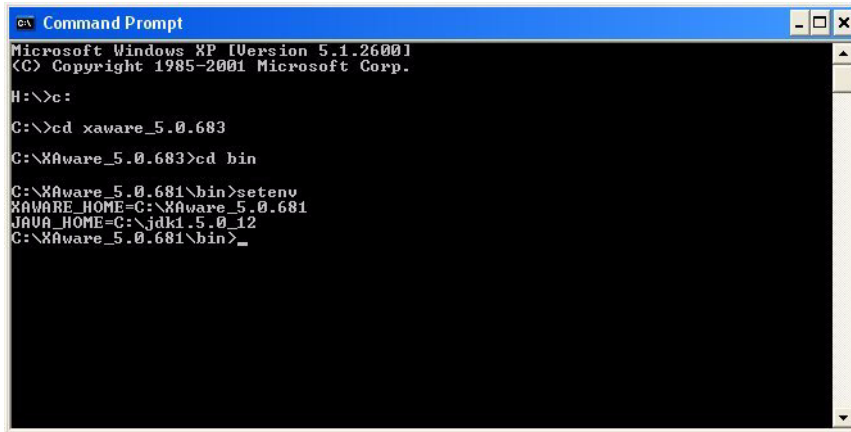
- XAWARE_HOME
- JAVA_HOME



Note: Java 1.5 is required!

A convenience script, *SetEnv.cmd* (*setenv.sh* for Linux/Unix), is included in the *<Install Directory>/bin* folder of your install. Running this script establishes XAWARE_HOME based on your installed version of the product. If JAVA_HOME is not already set, it will be set to the Java version included with your installation of XAware.

The current values of the environment variables will be echoed in a command window, as shown here.



```

ca Command Prompt
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

H:\>cd c:

C:\>cd xaware_5.0.683

C:\XAware_5.0.683>cd bin

C:\XAware_5.0.681\bin>setenv
XAWARE_HOME=C:\XAware_5.0.681
JAVA_HOME=C:\jdk1.5.0_12
C:\XAware_5.0.681\bin>_

```



Note: You can also set these environment variables directly into your environment.

In addition, you should include the `BizDoc.cmd` (`bizdoc.sh` for Linux/Unix) script in your `PATH` environment variable. You can accomplish this using one of the following methods:

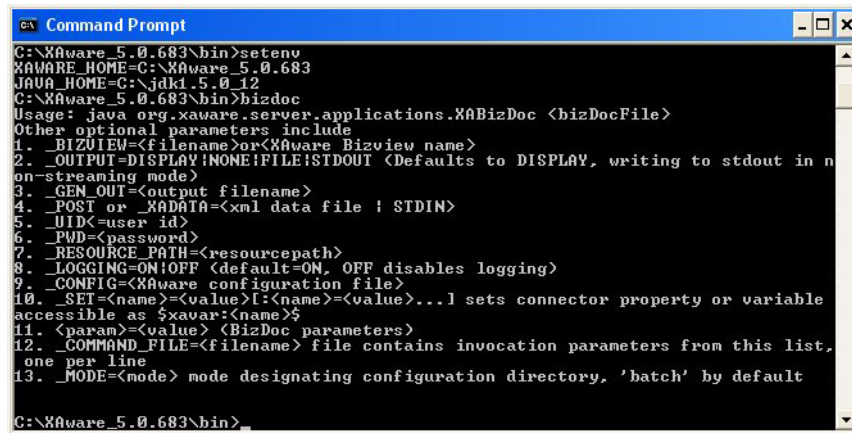
- Include `<Install Directory>/bin` in your system's `PATH` environment variable
- Copy the `BizDoc.cmd` (`bizdoc.sh` for Linux/Unix) script to a folder that is already in your `PATH`

Getting Help

Running the script with no parameters prints the help instructions:

1. Type BizDoc (bizdoc in a Linux/Unix environment) in the command window and press <Enter>.

The optional parameters provide information into the BizView processing, as shown here.



```

C:\XAware_5.0.683\bin>setenv
XAWARE_HOME=C:\XAware_5.0.683
JAVA_HOME=C:\jdk1.5.0_12
C:\XAware_5.0.683\bin>bizdoc
Usage: java org.xaware.server.applications.XABizDoc <bizDocFile>
Other optional parameters include
1. _BIZVIEW=<filename>or<XAware Bizview name>
2. _OUTPUT=DISPLAY|NONE|FILE|STDOUT (Defaults to DISPLAY, writing to stdout in n
on-streaming mode)
3. _GEN_OUT=<output filename>
4. _POST or _XADATA=<xml data file | STDIN>
5. _UID=<user id>
6. _PWD=<password>
7. _RESOURCE_PATH=<resourcepath>
8. _LOGGING=ON|OFF (default=ON, OFF disables logging)
9. _CONFIG=<XAware configuration file>
10. _SET=<name>=<value>[:<name>=<value>...] sets connector property or variable
accessible as $xavar:<name>$
11. <param>=<value> (BizDoc parameters)
12. _COMMAND_FILE=<filename> file contains invocation parameters from this list,
one per line
13. _MODE=<mode> mode designating configuration directory, 'batch' by default
C:\XAware_5.0.683\bin>_

```

Similar to the Edit Input Parameters for Execution dialog box that appears when you execute a BizView inside the Designer environment, the parameters shown in the following table can be used by the BizView execution process.



Note: The format for the parameters in the following table is param=value, as in `_LOGGING=OFF`, or `<filename>=myCompany/verify.xbd`.

Optional Parameter	Description
<code>_BIZVIEW</code>	This parameter defines the path and file name or BizView name assigned to the BizDocument you want this batch process to execute. This is an alternate form of <code><filename></code> .
<code>_COMMAND_FILE</code>	This parameter defines a file that contains invocation parameters from this list—one per line. Each line invokes a BizView file.
<code>_CONFIG</code>	This parameter defines an XAware configuration file (XAsystem.xml).
<code>_GEN_OUT</code>	This parameter defines the path and file name for the output results. It can be used when the <code>_OUTPUT</code> parameter value is defined as <code>FILE</code> , or it can be used alone. For example, if the <code>_OUTPUT</code> parameter value is <code>NONE</code> , but this parameter value is defined as <code>C:/outputFile.txt</code> , the output will be saved to this file but will not appear anywhere else.

Optional Parameter	Description
_LOGGING	This parameter defines whether logging is off or on when the BizView is processing. The default is on, so this parameter is typically used to disable logging for the BizView process.
_MODE	This parameter designates the configuration directory. It is 'batch' by default. This is the custom configuration for spring and logging in the <Install Directory>/conf subdirectory.
_OUTPUT	<p>This parameter defines the output of the batch process. The supported values are DISPLAY, FILE, NONE, or STDOUT which are described here.</p> <ul style="list-style-type: none"> • DISPLAY means that the results of the batch processing appear in the window where the batch process occurs. Typically, this is a command prompt window. • FILE means that the results of the batch process are exported to a file. This value requires the _GEN_OUT parameter. • NONE means that the results of the batch processing are ignored and not displayed anywhere. • STDOUT means to write the results out in streaming mode. The BizDocument is run in streaming mode and output is sent to stdout.
_POST or _XADATA	This parameter defines the input XML path and file name if the BizView process you are executing with the batch process requires input XML. It can also use a STDIN, which reads input from the console's stdin. The BizDocument is set to instreaming if it supports it.
_PWD	This parameter defines the password if security is established for the BizView being executed with the batch process.

Optional Parameter	Description
<code>_RESOURCE_PATH</code>	<p>This optional parameter value is a string of paths that resolve the relative file referenced defined in the BizView files. Each path in the resource path is separated by semi-colons and you will want to declare each of the folder and file names in the deployed project structure. One example could look like this:</p> <pre data-bbox="586 455 1296 566">_RESOURCE_PATH=C:/xaware/myCompany/myApplication/application.xar;C:/xaware/myCompany/myApplication/common/policy.xdr;C:/xaware/myCompany/myApplication/policy/retrievePolicies.xbd;</pre> <p>Using this example, the system will be able to find the folders and files that are used to define your deployed application. If a file is referenced with a relative reference and stored in more than one of the declared paths, then the first occurrence of that file or archive will be used and the others will be ignored. It is important to use unique naming for your folders and file names to ensure the correct file is used.</p>
<code>_SET</code>	<p>This defines the connector properties, which can then be accessed using the standard <code>\$xavar.property\$</code> variables. For example, the format is: <code>_SET=property1=value, property2=value</code>.</p>
<code>_UID</code>	<p>This parameter defines the user ID if security is established for the BizView you are executing with the batch process.</p>
<code><bizdoc></code>	<p>This parameter defines the input parameter names and values if the BizView being executed with the batch process requires input parameter values.</p>
<code><filename></code>	<p>This parameter defines the BizView file to run using:</p> <ul data-bbox="586 1161 925 1271" style="list-style-type: none"> • a fully qualified name, • an absolute path, or • a path relative to the project.

See the next section to invoke BizView files from the command line.

Invoking BizView Files

◆ Invoking a Deployed BizView

Once the environment is set up, you can invoke BizView files using the `BizDoc.cmd` (`bizdoc.sh` for Linux/Unix) script command.

1. From the command window, access the bin directory (`<Install directory>/bin`) and type `bizdoc <filename>` and press `<Enter>`.

◆ Invoking an Undeployed BizView

To run an undeployed BizView file, you will navigate to an XAware project folder (or any folder that is in the XAware source root) and invoke the BizView file with the fully qualified name of that component. Commands invoking undeployed BizViews work only when file dependencies are included within the same project, and when all file references are relative to the root of the project. The command looks like this:

1. From `<Install directory>/bin` directory, type `bizdoc <folder>/<filename>` and press `<Enter>`.

For example, to run an example BizDocument loaded into the product workspace, requiring an input parameter value, the command would look something like this:

```
C:\XAware\designer\workspace\Transform Fields>bizdoc myCompany/
transformDatewithFuncoid.xbd "date=2007-09-15 10:25:47"
```



Note: Commands typed in Windows will require quotes around the parameters.

The results returned would look a little like this:

```

Call stack is empty.
=====
2007-10-30 12:14:30.873 000001.0010 FINER   completing transaction with success
= false
2007-10-30 12:14:30.873 000001.0010 DEBUG   BizUIViewSession::closeSessionStreams
ENTER
2007-10-30 12:14:30.873 000001.0010 DEBUG   BizUIViewSession::closeSessionStreams
EXIT
2007-10-30 12:14:30.873 000001.0010 INFO    End BizDoc:myCompany/transformDatewi
thFuncoid.xbd duration=406 ms Success
<?xml version="1.0" encoding="UTF-8"?>
<transform>
  <transformDate>20070915000000</transformDate>
</transform>

```

Using Multiple Versions of XAware

If you regularly use multiple versions of the XAware product, you will want to set up your environment as described in these brief steps.

- a. Copy the batch script to a folder that is defined in your PATH.
- b. To invoke BizView files using a specific version of XAware, navigate your command line window to the directory of the installation you want to use.

- c. Then, run the `SetEnv.cmd` (`setenv.sh` for Linux/Unix) to establish the `XAWARE_HOME` and `JAVA_HOME` environment variables.
- d. Now, you can navigate to any XAware project directory and invoke the script as desired. The script will use the XAware version you selected when you executed the set environment script.

This concludes the information for batch processing.



Index

Symbols

- & character 9
- = character 9

A

- asynchronous 5

B

- batch connector
 - environment variables 44
 - help with 45
 - setting it up 44
- batch processing
 - parameter 48
 - parameter 48
 - _BIZVIEW parameter 46
 - _COMMAND_FILE parameter 46
 - _CONFIG parameter 46
 - _GEN_OUT parameter 46
 - _LOGGING parameter 47
 - _MODE parameter 47
 - _OUTPUT parameter 47
 - _POST parameter 47
 - _PWD parameter 47
 - _RESOURCE_PATH parameter 48
 - _SET parameter 48
 - _UID parameter 48
 - _XADATA parameter 47
 - introduction 44
 - invoking deployed BizViews 48
 - invoking undeployed BizViews 49
 - with multiple XAware versions 49

C

- client application request 2

Connectors

- asynchronous 5
- Batch Processing 4
- EJB 4
- HTTP-Servlet 4
- introduction 2
- JMS 4
- SOAP 4
- synchronous 5
- which one to use? 4

E

- EJB Connector
 - client application code 39
 - client request format 38
 - client request with input parameters 39
 - client request with input XML 39
 - JNDI name 40
 - using with BEA application server 40
 - xacjb.executeBizDoc method 40
- EJB connector
 - configuration parameter list 41
 - error handling 42
 - errorTemplate parameter 41
 - invocationCommand parameter 41
 - outputEncoding parameter 41
 - prettyPrint parameter 42
 - propagateError parameter 42

H

- HTTP Connector 3
 - client request format 9
 - introduction 8
- HTTP connector
 - error handling 14
 - errorTemplate parameter 12

- httpBizViewExceptionCode parameter 12
- httpBizViewFailureCode parameter 12
- httpContextPath parameter 12
- httpHost parameter 12
- httpPathInfo parameter 12
- httpPort parameter 12
- httpServletPath parameter 13
- invocationCommand parameter 13
- outputEncoding parameter 13
- parameter list 12
- prettyPrint parameter 13
- propagateError parameter 13
- resourcePath parameter 14
- urlParamMap parameter 14

httpHost 12

J

- Java Naming and Directory Interface (JNDI) name 40
- JMS connector
 - client application code 30
 - configuration parameters 34
 - error handling 35
 - errorTemplate parameter 34
 - introduction 30
 - invocationCommand parameter 34
 - outputEncoding parameter 35
 - prettyPrint parameter 35

R

- required knowledge iii

S

- SOAP connector
 - client application code 18
 - client request format 18
 - configuration parameters 26
 - error handling 27
 - generateSoapFault 26
 - httpContextPath 26
 - httpHost 26
 - httpPathInfo 26
 - httpPort 26
 - httpQueryString 26
 - httpServletPath 26
 - introduction 18
 - invocationCommand 26
 - outputEncoding 26
 - prettyPrint 27

- propagateError 27
- soapHeaderSelector 27
- synchronous 5

X

- XAServlet 3
- XAware technical support vi