User’s Guide

iPlanet™ XML Adapter Designer

Version 2.0

February 2002

Contents

List of Figures

List of Tables

List of Procedures

List of Code Examples

Preface

This preface provides information on the following topics:

- "Audience for This Guide"
- "Organization of This Guide"
- "Text Conventions" on page 17
- "Documentation Resources" on page 18

Audience for This Guide

This guide is intended for XML Adapter Designer (XAD) developers and administrators.

Organization of This Guide

The following table briefly describes the contents of each chapter:

Chapter	Description
Chapter 1, "Introduction"	Provides a high-level description of the XAD product and its features.
Chapter 2, "Adapter Architecture"	Explains the architecture of XAD adapters and the processing of requests at runtime.
Chapter 3, "Building an Adapter"	Explains how to use XAD to build an XML adapter. Also describes the components of the generated adapter.

Chapter	Description
Chapter 4, "XSLT Generation"	Provides information on using the Interface Editor to generate outbound and inbound XSLT rules and on testing the generated rules in the iPlanet Integration Server (iIS) XML/XSL Plan workshop.
Chapter 5, "Communicating With the Adapter: HTTP and JMS"	Describes the procedures you must perform to set up communication between an XAD adapter and its client.
Chapter 6, "Deploying and Testing an Adapter"	Explains how to compile, deploy, run, and test an XAD adapter. Also provides information on load balancing.
Chapter 7, "Working With Data Types"	Provides important information about working with data types in XAD.
Appendix A, "Running the iIS Client Example"	Explains how to set up and run the iIS example application provided in the XAD distribution as an XAD client.
Appendix B, "XAD Example Adapters"	Describes how to configure and run the example adapters provided with the XAD distribution.
Appendix C, "XAD Adapters at Runtime"	Describes how XAD adapters process requests at runtime.

Text Conventions

This section provides information about the conventions used in this document.

This user guide provides instructions to users on both Windows and Solaris operating systems. Throughout this appendix, the string *XAD_ROOT* appears in paths and refers to the directory in which the XAD software is installed (the value of your `XAD_ROOT` environment or user variable; `$XAD_ROOT` on Solaris platforms or `%XAD_ROOT%` on Windows platforms). In addition, you might need to substitute slashes for backslashes or vice versa in paths, depending on your operating system.

Table 1 Document Conventions

Format	Description
italics	Italicized text represents a placeholder. Substitute an appropriate clause or value where you see italic text. Italicized text is also used to designate a document title, for emphasis, or for a word or phrase being introduced.
`monospace`	Monospace text represents example code, commands that you enter on the command line, directory, file, or path names, error message text, class names, method names (including all elements in the signature), package names, reserved words, and URLs.
`[]`	Square brackets to indicate optional values in a command line syntax statement.
ALL CAPS	Text in all capitals represents file system types (GIF, TXT, HTML and so forth), environment variables (JMQ_HOME), or acronyms (XAD, JSP).
Key+Key	Simultaneous keystrokes are joined with a plus sign: Ctrl+A means press both keys simultaneously.
Key-Key	Consecutive keystrokes are joined with a hyphen: Esc-S means press the Esc key, release it, then press the S key.

Documentation Resources

On-line manuals for the XAD product are on the XAD CD. The XAD Release Notes, Platform Support Matrix, and on-line manuals are available online at http://docs.iplanet.com/docs/manuals/xad.html.

Table 2 lists the documents included with the XAD product.

Table 2 XAD Documentation Set

Document	Audience	Description
XAD Installation Guide	Developers and administrators	Explains how to install XAD software on Solaris and Windows platforms.
XAD User's Guide	Developers and administrators	Provides information needed to build, compile, deploy, and test XAD adapters.

Chapter 1

Introduction

XML Adapter Designer

iPlanet™ XML Adapter Designer (XAD) enables you to integrate applications and legacy systems by allowing you to generate Java™ and C adapters with standard Extensible Markup Language (XML) interfaces rather than build them from the ground up. It provides you with easy-to-use tools for generating code as well as testing and maintaining the adapters.

You can use XAD to develop adapters that can integrate any application into an enterprise-wide solution. XAD offers several major benefits:

- XAD provides a standard adapter architecture that helps you to reduce risk and maintain consistency. This improves time-to-market and increases re-use.
- Generating adapters rather than coding them reduces the time and expense required to implement them. Developers do not need to focus their efforts on integration technologies such as XML dialects, communication protocols and deployment languages. Instead, they can concentrate on the business requirements of their application.
- XAD allows you to implement adapters using a number of Internet standards. It supports the XAD XML dialect and allows you to plug in your own dialect engine for other XML dialects. It supports communication over both the HTTP protocol and the Java Messaging Service (JMS) protocol, according to the needs of your application. You can also build adapters that communicate using the SOAP support available with iPlanet Integration Server (iIS).

- The deployment facilities of XAD simplify installation, improve run-time performance and simplify run-time management. XAD allows you to test and manage adapters from any location with an easy-to use Java client application.
- All components of XAD are internationalized. For information on localizing XAD, refer to the XAD release notes available at http://docs.iplanet.com/docs/manuals/xad.html.

Standard Adapter Architecture

Using XAD, you can create any number of adapters that conform to the same architecture. Each adapter has an interface defining the target application and sends and receives messages that conform to the same Document Type Definition (DTD). Using a standard architecture and interface type reduces the time spent learning, maintaining, modifying and testing adapters.

Adapter Generation

The adapter's generation tool allows you to create a neutral representation of your target application's API. Using the same definition, you can generate Java or C code for an adapter for any application. The adapter contains an XML dialect engine, freeing developers from the effort of writing code to parse and process XML messages. You can easily customize the generated code to add business logic. You can also modify the definition if the API changes and regenerate the code while preserving your customizations. The generation tool generates sample XML requests that you can use for testing your adapter and use as templates for the calling application.

XSLT Rules Generation

If your adapter participates in an iIS business process, you can map the names of functions or methods in your API to activities in your business process and generate the inbound and outbound XSLT rules to transform documents coming from and going to the iIS process.

Testing and Management Facilities

XAD provides a Java client application to test and administer your generated adapters. This test client allows you to load and edit sample XML requests and send them to the generated adapter over HTTP or JMS. It checks requests for validity and well-formedness and captures and reports any errors. You can also use this tool for load-testing and tracing.

XAD also provides facilities for managing tracing, the return of parameter data types, and the state of the adapter (active or inactive).

XAD Components

An XAD installation consists of the following components:

XAD Generator This component creates and modifies definitions of target applications and automatically generates Java or C code, as well as sample XML messages and XSLT rules.

HTTP Servlet and JMS Transport Driver XAD supports both the HTTP protocol and JMS and provides two generic components that you can use to communicate with any generated Java adapter (XAD C adapters have integrated HTTP servers).

XML Adapter Test Client This graphical tool tests adapter functionality and provides administrative facilities. It allows you to import an XML request, sends it to the adapter and returns the response as an XML document. It can be used to send messages over HTTP and JMS.

Example Adapters and Example iIS Application Appendix B explains how to set up and run the example adapters. Appendix A explains how to set up and run an example adapter with an example iIS client application.

Chapter 2

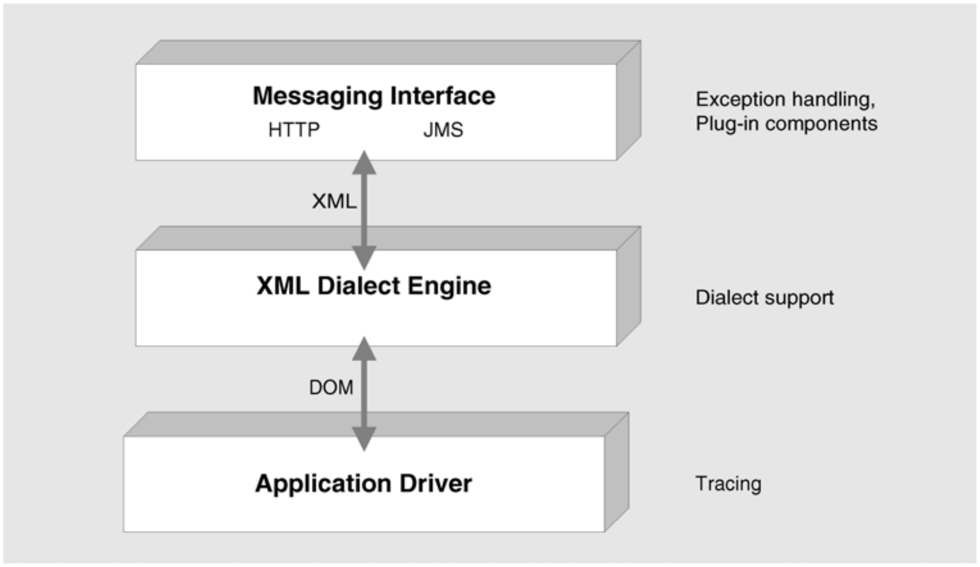
Adapter Architecture

This chapter describes the architecture of XAD adapters.

Internal Architecture

Figure 2-1 shows the internal architecture of an XAD adapter. The first layer is the application driver or wrapper for the enterprise information system (EIS) API. This wrapper contains the integration logic. It performs any customized business functionality just before issuing a call to the API.

Figure 2-1 Internal Architecture of an XAD Adapter



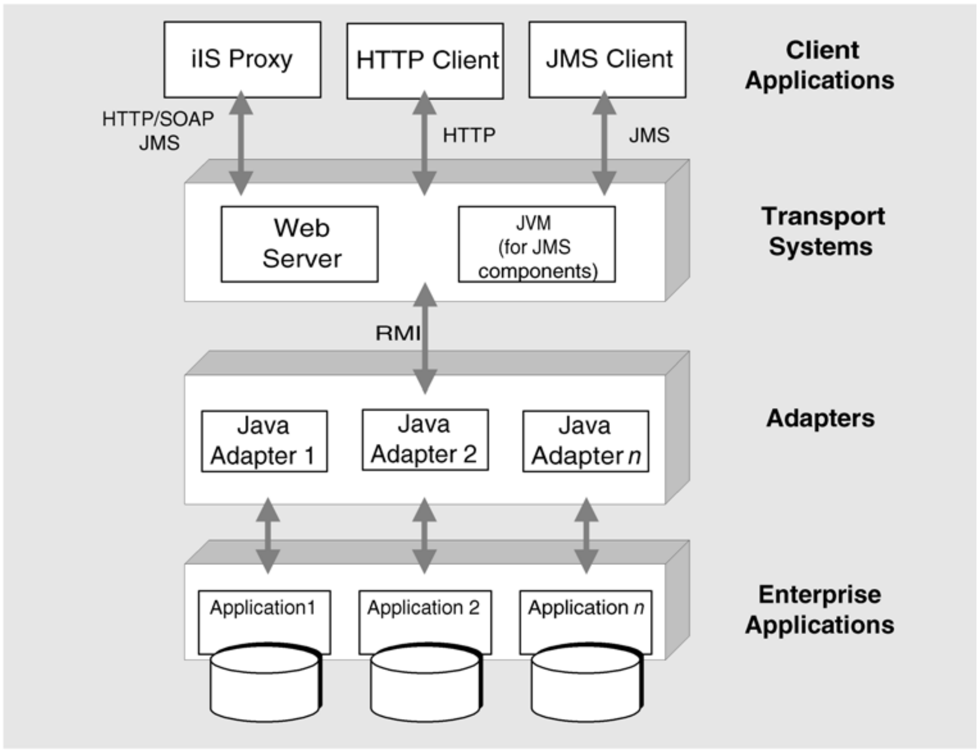
The XML dialect engine performs the input and output translations. This layer interprets messages by walking through the document object model (DOM). It parses the XML document and validates that it is well-formed and contains the required information. Adapters parse and build messages in different XML dialects. An adapter can support multiple dialects at the same time; when it receives an XML message it responds with a message in the appropriate dialect.

XAD adapters support the XAD dialect and can also connect to iIS proxies configured for SOAP. XAD also provides an interface that allows you to write your own XML dialect engine to plug into your adapter.

XAD Java Adapter in the Enterprise Application

Figure 2-2 shows how XAD generated Java adapters fit into an enterprise application. EIS applications expose their APIs to the adapters. Adapters communicate with the transport drivers using Java RMI. Client applications can be integrated across different transport systems, depending on what is available to the client.

Figure 2-2 XAD Java Adapter in the Enterprise Application

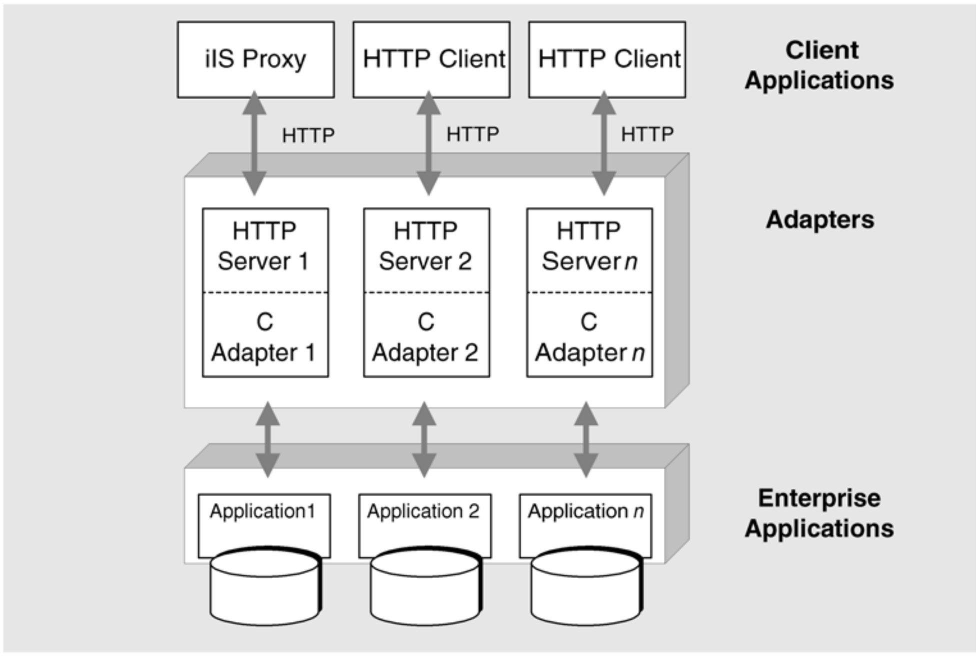


At runtime, the client sends an XML request to the transport driver. XAD provides two transport drivers, a listener servlet for communicating over HTTP and a JMS listener for communicating over JMS. The transport driver forwards this request as a string using RMI to the `XADRemoteServer` class in the adapter engine.

XAD C Adapter in the Enterprise Application

Figure 2-3 shows how XAD generated C adapters fit into an enterprise application. EIS applications expose their APIs to the adapters. Each adapter has an XAD provided HTTP server running in the same process. Client applications communicate with the adapter through the HTTP server.

Figure 2-3 XAD C Adapter in the Enterprise Application



At runtime, the client sends an XML request to the transport driver. In the case of an XAD C adapter, the transport driver is an HTTP server internal to the adapter.

Chapter 3

Building an Adapter

This chapter explains how to use XAD to build an XML adapter.

Adapter Development Overview

The following development overview lists the tasks required to build and deploy your XML adapter. Tasks involved with building your adapter are described in detail later in this chapter. The tasks of deploying and testing your adapter and generating sample test files are described in detail in other chapters.

➤ **To build and deploy an adapter**

1. Choose a name for your adapter.
2. Create an adapter directory structure (see “Creating an Adapter Directory Structure” on page 28).
3. Start the Interface Editor (see “Using the XAD Interface Editor” on page 30).
4. Create an XML definition of your target API (see “Creating an XML Definition of Your Target API” on page 30).
   a. Create a new interface (see “To create a new interface” on page 32).
   b. Create operations (see “To create a new operation” on page 32).
   c. Create parameters (see “To create a new parameter” on page 33).
5. Save the definition (see “Saving the API Definition” on page 37).
6. Generate the adapter (see “Generating an Adapter” on page 39).
7. Generate sample XML requests and responses (see “About Sample XML Files” on page 51).

8. *For Java adapters only*: Configure the adapter engine (see "Configuring the Adapter Engine" on page 81).
9. Configure the adapter engine runtime environment (see "Configuring the Adapter Runtime Environment" on page 82).
10. Set up the transport mechanism for communication between the client and adapter (see Chapter 5, "Communicating With the Adapter: HTTP and JMS").
11. Compile and run the adapter (see "Running the Adapter" on page 85).

The exact procedures involved in each step depend on whether you are creating a Java adapter or a C adapter.

Creating an Adapter Directory Structure

Before you begin development on your adapter, you should set up a development directory structure. Although the directory structure described in this section is not required, it is recommended that you create this structure for each new adapter to ensure consistency and in order to minimize modifications you make to the scripts that run each component. If you do not use this recommended directory structure, you must modify the XAD scripts according to the directory structure you use.

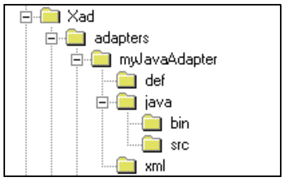
Before you begin, select a name for your new adapter. You will use this name in several places and it must match exactly in order for everything to run. The procedures in this section use *adapterName* to represent the name you have chosen for your adapter.

➤ **To set up a directory structure for a Java adapter**

1. Within the *XAD_ROOT*`\adapters` directory, create a new subdirectory using the name of your new adapter. Within this *adapterName* directory, create three subdirectories. Name these directories `def`, `java`, and `xml`. Inside the `java` directory create two directories named `bin` and `src`.

   Figure 3-1 shows the directory structure for generating a Java adapter named `myJavaAdapter`.

Figure 3-1 Directory Structure for Generating a Java Adapter



2. From the *XAD_ROOT*`\adapters\XADDemo_Adapter\java` directory, copy the script file `setadapterenv` into the *XAD_ROOT*`\adapters\`*adapterName*`\java` directory.

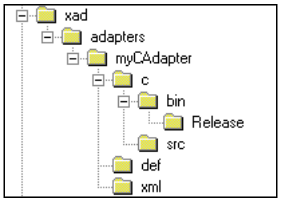
   You will edit this file later, as explained in "Configuring the Adapter Runtime Environment" on page 82.

➤ To set up a directory structure for a C adapter

1. Within the *XAD_ROOT*`\adapters` directory, create a new subdirectory using the name of the new adapter. Within this *adapterName* directory, create three subdirectories. Name these directories `def`, `c`, and `xml`. Inside the `c` directory create two directories named `bin` and `src`. Inside the `bin` directory create a directory named `Release`.

   Figure 3-1 shows the directory structure for generating a C adapter named `myCAdapter`.

Figure 3-2 Directory Structure for Generating a C Adapter



2. From the *XAD_ROOT*`\adapters\XADSamples_C\c` directory, copy the script file `setadapterenv` into the *XAD_ROOT*`\adapters\`*adapterName*`\c` directory.

   You will edit this file later, as explained in "Configuring the Adapter Runtime Environment" on page 82.

3. Copy the C header file for your target API into the *XAD_ROOT*`\adapters\`*adapterName*`\c\src` directory.

You are now ready to create an API definition and use it to generate your adapter and other sample documents. The procedures for performing these tasks are described in the following section.

Using the XAD Interface Editor

The Interface Editor is a graphical tool that you use to create and edit an XML document describing the API of a given package. After creating your API definition, you generate the Java or C code that executes the calls to your target API. The generated code provides methods or functions that you can customize to add business logic.

You also use the Interface Editor to generate sample XML requests based on your target API. You can then send these requests to the generated adapter for testing. In addition, you can generate XSLT rules for adapters that take part in an iIS business process.

The following section provides instructions on using the Interface Editor to create and edit XML API definition files and to generate source code for your adapter.

Creating an XML Definition of Your Target API

Before creating the XML definition of your target API, perform the following steps:

1. Make or obtain a list of the methods or functions the adapter will call.

   Be sure to include information about the method or function parameters.

2. Ensure that your iPlanet Unified Development Server (UDS) environment is running.

3. Start the XAD Generator.

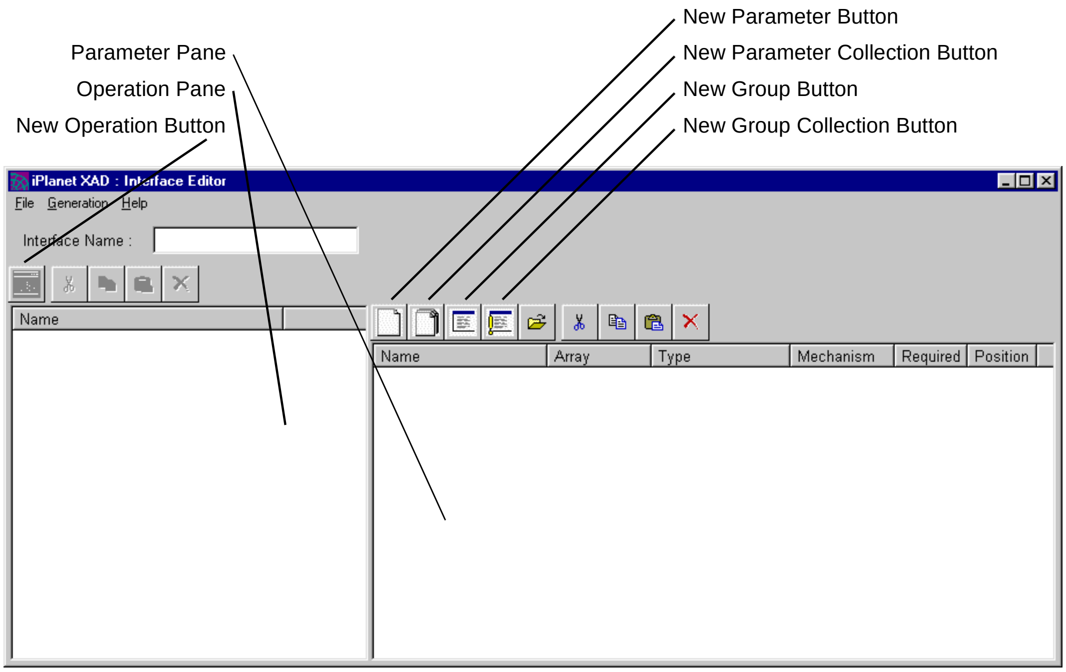
   You can do this in several ways:

   - Run the XAD_Generator application from your UDS Application Launcher.

- Execute the `xad_run_generator` script in the *XAD_ROOT*`\install\bin` directory.
- On a Windows platform, select Start > Programs > Forte Applications > XAD_Generator.

An empty Interface Editor window opens. Figure 3-3 shows the Interface Editor and indicates the buttons you use to create new operations and parameters for your API definition.

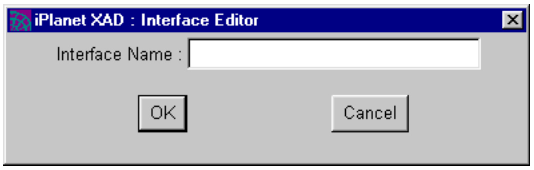
Figure 3-3 The XAD Interface Editor



➤ To create a new interface

1. In the Interface Editor, select File > New.

   An Interface Name dialog box opens.

2. Type a name for your API definition and click OK.

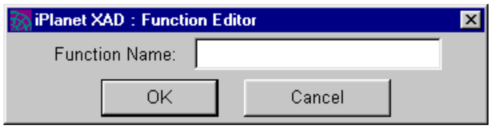
   For the value of this field, you should preferably use the adapter name you used in creating your directory structure, but this is not required. This field is case sensitive.

➤ To create a new operation

1. Click the New Operation button (see Figure 3-3).

   The Function Editor opens.

2. Type the name of the method or function of the target API that the adapter is going to call and click OK.

3. The name of the new operation appears in the left-hand pane of the Interface Editor.

Creating Parameters

➤ **To create a new parameter**

1. In the left-hand pane of the Interface Editor, select the operation for which you want to create a parameter.

2. Open the Parameter Editor by clicking the parameter button appropriate for the kind of parameter you want to create.

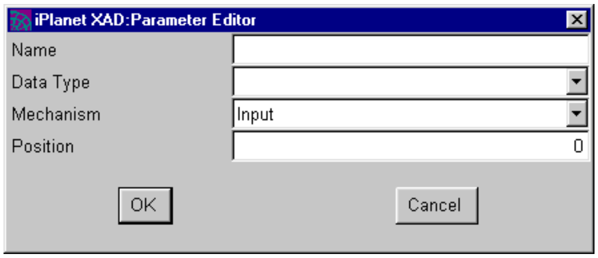
   Figure 3-3 shows the parameter buttons in the Interface Editor. Table 3-1 describes the parameter buttons.

Table 3-1 Interface Editor Parameter Buttons

Button Name	Purpose
New Parameter	Create a simple parameter, for example `int` or `Integer`.
New Parameter Collection	Create an array of simple parameters.
New Group	Create a parameter whose data type is a user-defined class or `struct`.
New Group Collection	Create an array of parameters whose data type is a user-defined class or `struct`.

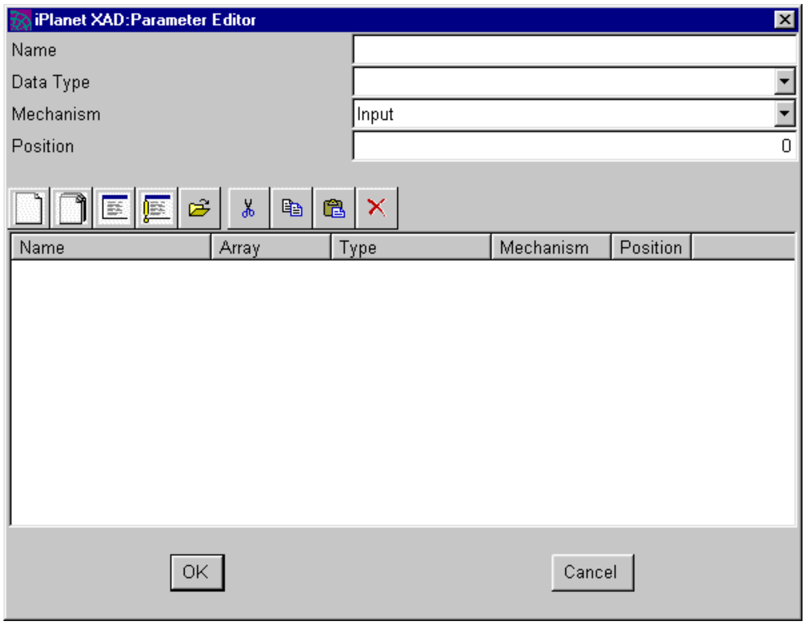
The appearance of the Parameter Editor depends on which parameter button you click to open it. Figure 3-4 shows the Parameter Editor that opens when you click the New Parameter button. Figure 3-5 shows the Parameter Editor that opens when you click the New Group button.

Figure 3-4 Parameter Editor for Creating a Simple Parameter



Name	
Data Type	
Mechanism	Input
Position	0

Figure 3-5 Parameter Editor for Creating a Group Parameter



3. Use the information in Table 3-2 to fill out the fields in the Parameter Editor.

Table 3-2 Parameter Editor Fields

Field	**Description**
Name	Name of the parameter.
Data Type	*If you opened the Parameter Editor by clicking the New Parameter or New Parameter Collection button*, choose an abstract data type from the list, or type in any standard, language-specific data type. For example, for a Java adapter you could type in `int`, `Integer`, or `String`. Examples of C data types you can type in include `int`, `char *`, and `float`. You must specify the data type of C pointers completely. For example, you must specify the data type of a pointer to a long integer, as `long int *` (rather than `long *`). Likewise, specify `short int *` (rather than `short *`), `unsigned short int *` (rather than `unsigned short *`), and so forth. Do not enclose language-specific data types in curly braces. Curly braces denote abstract data types. *If you opened the Parameter Editor by clicking the New Group or New Group Collection button*, type the name of your parameter's user-defined data type. For Java, the name must be fully qualified. For more information on working with data types in XAD, see Chapter 7, "Working With Data Types."
Mechanism	Specifies how your parameter is passed. Available values are INPUT, OUTPUT, INOUT, and RETURN, with the following restrictions: • Do not choose INOUT for scalar data types, such as `int`. • *For Java adapters*, if the parameter is a simple data type, such as `int` or `String`, do not choose OUTPUT or INOUT.
Position	Specifies the order in which the parameters are passed to the method or function. You should specify the position of each parameter as you create it in the Interface Editor. The alternative is to use the default value of `0` for each parameter. In this case the parameters are positioned in the order in which you create them. However, if you sort the parameters, you will lose track of the order in which they were created.

Table 3-2 Parameter Editor Fields *(Continued)*

Field	Description
Array	If the mechanism is INPUT, OUTPUT, or INOUT, specify the maximum number of elements required in the array. Do not set this number higher than necessary because the array will be created with this number of elements even if fewer are needed. For Java adapters, if the mechanism is RETURN, this field is ignored because the array is sized dynamically. This field appears only if you click the New Parameter Collection button.

4. If you are creating a Group parameter or an array of Group parameters, add a parameter for each data field of the class or `struct`.

   a. In the Parameter Editor, click the appropriate button for the data type.

      Your choices are as before:

      - New Parameter
      - New Parameter Collection
      - New Group
      - New Group Collection

      Another Parameter Editor opens.

   b. Fill out the fields and click OK.

      If any of the parameters are of user-defined data types, continue drilling down on them until you arrive at a simple data type or array of simple data types.

5. Click OK.

   The Parameter Editor closes and the information for each parameter appears in the right-hand pane of the Interface Editor.

   For more information on working with Abstract Data types, refer to Chapter 7, "Working With Data Types."

Saving the API Definition

You can save API definitions created in the Interface Editor as XML documents. You can re-import and edit these documents at any time.

➤ **To save an API definition**

1. When you have finished entering functions and parameters, select File > Save. Your API definition will be saved as a `.xml` file. Save this file to your *adapterName*`\def` directory.

The XML API Definition

Code Example 3-1 shows a simple API definition. The API name is `Customer` and the definition contains one method, `getCustomer`.

Code Example 3-1 XML Definition for a Simple API

```
<FFX_PACKAGE VERSION="1.0" PACKAGE_NAME="Customer">
  <EXECUTOR/>
  <FUNCTION>
    <FUNCTION_NAME>getCustomer</FUNCTION_NAME>
    <PARAMETER_LIST>
      <ITEM MECHANISM="INPUT" REQUIRED="TRUE" POSITION="1">
        <PARAMETER NAME="Id" DATATYPE="{Integer}"/>
      </ITEM>
      <ITEM MECHANISM="RETURN" REQUIRED="TRUE" POSITION="2">
        <PARAMETER NAME="Name" DATATYPE="{String}"/>
      </ITEM>
    </PARAMETER_LIST>
  </FUNCTION>
</FFX_PACKAGE>
```

Editing an API Definition

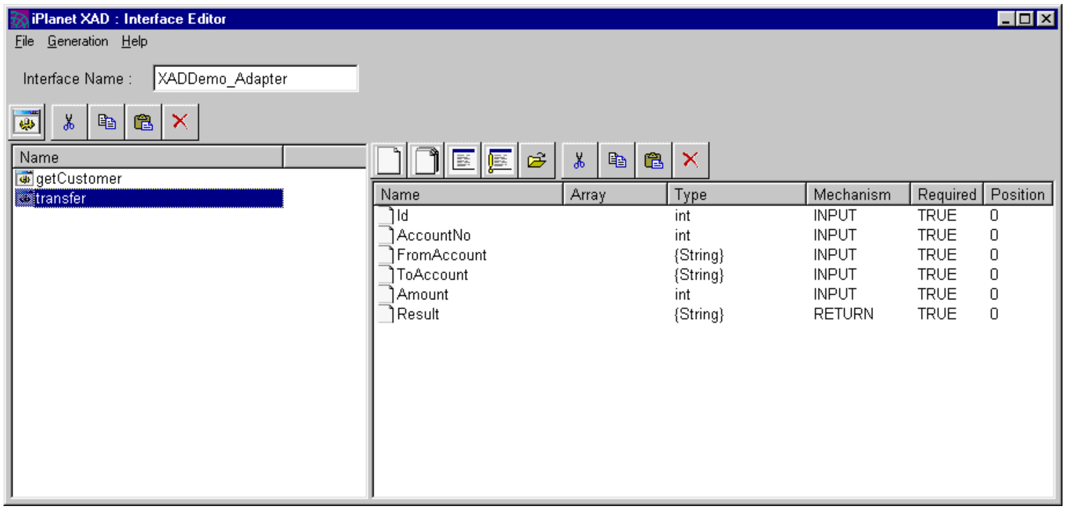
➤ To edit an API definition

1. In the Interface Editor, select File > Open.

   A file browser opens.

2. Select the XML file that contains the API definition and click OK.

   The name of the interface appears in the Interface Name field at the top of the Interface Editor and the functions and their parameters are displayed in the left and right hand panes, respectively.



Interface Name : XADDemo_Adapter

Name
getCustomer
transfer

Name	Array	Type	Mechanism	Required	Position
Id		int	INPUT	TRUE	0
AccountNo		int	INPUT	TRUE	0
FromAccount		{String}	INPUT	TRUE	0
ToAccount		{String}	INPUT	TRUE	0
Amount		int	INPUT	TRUE	0
Result		{String}	RETURN	TRUE	0

3. To change the adapter name, double-click in the Interface Name field and type the new name.

4. To edit an operation or parameter, double-click the operation or parameter name.

   The Function or Parameter Editor opens.

5. Edit the name or other information and click OK.

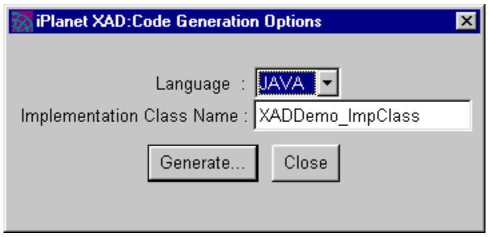
NOTE If you generate code or sample XML files and then edit the API definition, you must regenerate the code and sample XML files to propagate your changes.

Generating an Adapter

Using the XML representation of your target API, you generate source code for your adapter.

➤ **To generate source code for your adapter**

1. In the Interface Editor, choose Generation > Code Generation. The Code Generation Options dialog box opens.


iPlanet XAD:Code Generation Options
Language : JAVA
Implementation Class Name : XADDemo_ImpClass
Generate...
Close

2. In the Language field, choose the language in which to generate the code.

   Your options are Java and C. If you choose Java, the other field in the dialog box is named Implementation Class Name. If you choose C, the other field is named Implementation Header Name.

3. Supply a value for the other field in the dialog box as follows:

Field	Description
Implementation Header Name	The name of the C header file of your target API. Do not include the file extension. You can specify only one header file per adapter.
Implementation Class Name	The fully-qualified name of the class that implements your target API. Do not include the file extension. You can specify only one class per adapter.

4. Click Generate.

A file browser opens for you to select a location in which to save your code.

a. If you are generating Java code, navigate to the *adapterName*`\java\src` directory, where *adapterName* is the name you chose for your adapter when you set up your adapter directory structure.

If you are generating C code, navigate to the *adapterName*`\c\src` directory.

b. Save your adapter as *adapterName.*

If you are generating Java code, Java source files based on the API definition are generated. The default names of the files are *adapterName*`.java`, and *adapterName*`_PrePost.java`.

If you are generating C code, C source and header files based on the API definition are generated. The default names of these files are *adapterName*`.c`, *adapterName*`.h`, and *adapterName_PrePost*`.c`.

5. Click Close to close the dialog box.

The Generated Adapter

Depending on which code generation language you choose, your adapter is generated as either source files for Java classes or C source and header files.

Java Adapter Source Code

Generating a Java adapter creates two Java source files:

- *adapterName*`.java`
- *adapterName*`_PrePost.java`

The source file *adapterName*`.java` defines a subclass of `XADRemoteServer`. This class is an RMI server and is responsible for accepting client requests and building and executing calls to the target API. For each operation in the API definition, a corresponding method is generated in this class. These generated methods are named using the form *APIMethodName*`_Wrapper`, where *APIMethodName* is the name of the corresponding operation in the API definition.

The other generated class, *adapterName*_`PrePost.java`, is a customizable helper class for the RMI server. For each operation in the API definition, two corresponding methods are generated in this class. These generated methods are named using the forms `pre`_*APIMethodName*, and `post`_*APIMethodName*, where *APIMethodName* is the name of the corresponding operation in the API definition. These methods are called by the RMI server before and after calls to the target API.

Table 3-3 describes the methods of the generated RMI server class. Table 3-4 describes the methods of the RMI server's customizer class.

Table 3-3 Methods of the Java Adapter's RMI Server Class

Method Name	**Parameters**	**Return Value**	**Purpose**
adapterName	none	none	Constructor method for adapter. Instantiates the adapter engine and the *adapterName*_`PrePost` class.
`main`	`String[] args`		Instantiates and registers the adapter.
`processCall`	`XADInteractionSpec`	none	Called by the adapter engine. Acts as a router for requests by checking for the method name and calling the appropriate wrapper method for each method.
*APIMethodName*_`Wrapper`	`XADInteractionSpec`	none	Extracts method name and parameters from `XADInteractionSpec` object. Calls the `pre`_*APIMethodName* method, calls the target API method then calls `post`_*APIMethodName* method.

Table 3-4 Methods of the Java Adapter's Customizable Helper Class

Method Name	**Parameters**	**Return Value**	**Purpose**
`pre`_*APIMethodName*	`XADInteractionSpec`	none	These methods are called by the *APIMethodName*_`Wrapper` method before the call to the target API. Their parameters become input parameters to the API.

Table 3-4 Methods of the Java Adapter's Customizable Helper Class *(Continued)*

Method Name	**Parameters**	**Return Value**	**Purpose**
`post_`*APIMethodName*	`XADInteractionSpec`	none	These methods are called after the call to the target API. The parameters become output parameters and return values. You can also use these methods to perform operations on the data returned by the API.

C Adapter Source Code

Generating a C adapter creates a header file and two C source files:

- *adapterName*`.h`
- *adapterName*`.c`
- *adapterName_PrePost*`.c`

The *adapterName*`.c` file defines an HTTP server that is responsible for accepting client requests and building and executing calls to the target API. For each operation in the API definition, a corresponding function is generated in this file. These generated functions are named using the form *APIFunctionName*`_Wrapper`, where *APIFunctionName* is the name of the corresponding operation in the API definition.

The *adapterName*`_PrePost.c` file provides customizable helper functions for the HTTP server. For each operation in the API definition, two corresponding functions are generated in this file. These generated functions are named using the forms `pre_`*APIFunctionName*, and `post_`*APIFunctionName* where *APIFunctionName* is the name of the corresponding operation in the API definition. These functions are called by the HTTP server before and after calls to the target API.

Table 3-5 describes the functions in the generated HTTP server source file.

Table 3-5 Functions of the C Adapter's HTTP Server

Function Name	Parameters	Return Value	Purpose
`processRequest`	`FAD_OperationPtr` `FAD_OperationPtr*`	`FAD_ErrorPtr`	Called by the adapter engine. Acts as a router for requests by checking for the function name and calling the appropriate wrapper function for each function.
*APIFunctionName*_`Wrapper`	`FAD_OperationPtr` `FAD_OperationPtr*`	`FAD_ErrorPtr`	Extracts function name and parameters from the `FAD_OperationPtr` structure. Calls the `pre_`*APIFunctionName* function, calls the target API function, and then calls the `post_`*APIFunctionName* function.

Table 3-6 describes the functions in the customizable helper source file.

Table 3-6 Functions of the C Adapter's Customizable Helper Source File

Function Name	Parameters	Return Value	Purpose
`pre_`*APIFunctionName*	Pointers to all input and input-output parameters of target API.	none	These functions are called by the *APIFunctionName*_`Wrapper` function before the call to the API. Their parameters become input parameters to the API.
`post_`*APIFunctionName*	Pointers to all return output and input-output parameters of target API.	none	These functions are called after the call to the API. Their parameters become output parameters and return values. You can also use these functions to perform operations on the data returned by the API.

Customizing Your Generated Source Code

Your generated adapter provides for code customization by way of a customizable class file in the Java adapter and a customizable C source file in the C adapter. These files are discussed in "The Generated Adapter" on page 40. This section describes these customization topics:

- The mechanism for retaining code customization after regenerating an adapter
- How to set a response XML message to a different encoding than the request

Backup Files

If you edit any of the customizable helper methods or functions of your adapter and afterwards regenerate the source code, your customizations are saved to a backup file.

For example, if you customize any of the methods in *adapterName*`_PrePost.java` and then regenerate your adapter, the *adapterName*`_PrePost.java` file is automatically saved as *adapterName*`_PrePost.bak` before the new file is created. You can then copy your customizations from the backup file to the regenerated file. If you regenerate a second time, the backup file will be overwritten.

This feature works the same for C adapters as for Java adapters.

Overriding the Default XML Encoding

By default, an XAD adapter sets the encoding of the response XML message to be the same as the request XML message. In some cases, however, you might need the response XML message to use a different encoding than the request. For example, the request might be in English, but you want the response to be in Russian. These languages require different encodings.

In a Java adapter, you can override the default encoding of the response XML message by making a call to the `setEncoding` method on the `XADInteractionSpec` object. You make this call from within one of the adapter's customizable methods—either the `pre_`*APIMethodName* or `post_`*APIMethodName* method.

The `setEncoding` method takes the `XADInteractionSpec` object as an input parameter. See "Methods of the Java Adapter's Customizable Helper Class" on page 41 for more information on the `post_`*APIMethodName* method.

The following code shows an example of a customized `post_`*APIMethodName* method that makes a call to the `setEncoding` method. In this example, the method call overrides the default encoding by setting the encoding to `iso-8859-5`.

```
public void post_transfer(XADInteractionSpec inSpec) throws
XADInteractionSpecException{
  inSpec.setEncoding("iso-8859-5");
  /***************
  String Result = new String();  // RETURN Parameter
  ///////////////////////////////////////
  //Assign your values to the parameters here




  ///////////////////////////////////////

  //
  //Unpacking Result
  //
  FFXtmpParam = new paramDetails();
  FFXtmpParam.paramName = "Result";
  FFXtmpParam.paramType = "{String}";
  FFXtmpParam.mechanism = "RETURN";
  FFXtmpParam.setStringValue(Result);
  inSpec.setReturnParam("Result", "String", FFXtmpParam );
  ***************/
   return;
}
```

In a C adapter, you can override the default encoding of the response XML message by making a call to the adapter engine's `setEncoding` function. You make this call from within one of the adapter's customizable functions—either the `pre_`*APIFunctionName* or `post_`*APIFunctionName* method (see "Functions of the C Adapter's Customizable Helper Source File" on page 43).

The following code shows an example of a customized `post_`*APIFunctionName* function that makes a call to the `setEncoding` function. In this example, the function call overrides the default encoding by setting the encoding to `iso-8859-5`.

```
void Post_transfer(char **Result){
  setEncoding("iso-8859-5");
}
```

Overriding the encoding of a request affects only the single response to that request. Subsequent responses are in the encoding specified in the request, unless they are also overridden. If the request does not specify an encoding, UTF-8 is used by default.

The value of the parameter you provide to the `setEncoding` method or function is not checked. If you provide an invalid value to a Java adapter, the encoding is set to UTF-8 by default. If you provide an invalid value to a C adapter, the data might be unreadable.

Chapter 4

XSLT Generation

This chapter provides information on using the Interface Editor to generate outbound and inbound XSLT rules and testing the generated rules in the iIS XML/XSL Plan workshop.

Generating XSLT Rules

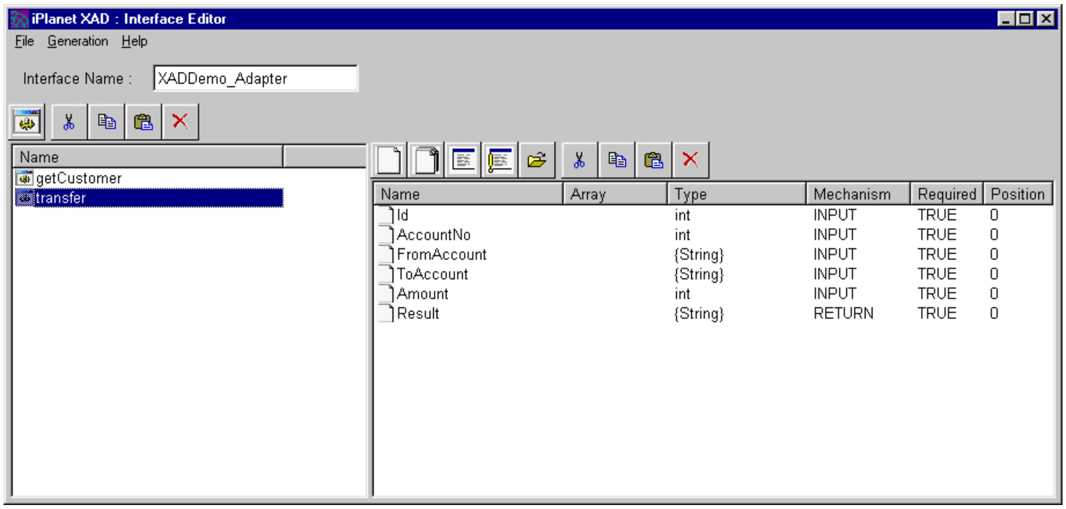
XML Adapter Designer supports the generation of XSLT rules that enable developers to integrate and test adapters with an iIS backbone. You can map the functions and parameters defined in your API definition to activities and attributes in an iIS process definition. XAD generates inbound and outbound XSLT rules based on these mappings (inbound and outbound relative to the iIS proxy). You can import the rules into the iIS XML/XSL Plan Workshop and test them against generated sample state and response documents. You can then register them with an iIS backbone. For more information on working with an iIS backbone, see the *iIS Backbone System Guide*.

Before You Begin

In order to call a function in your application from an iIS process, the name of the function must correspond to the value of an application code attribute. This attribute is created in the application dictionary item associated with the activity to which you are mapping the function.

➤ To generate outbound XSLT rules

1. Start the XAD Interface Editor.
2. Import an existing API definition by selecting File > Open or create a new definition according to the steps in "Adapter Development Overview" on page 27.

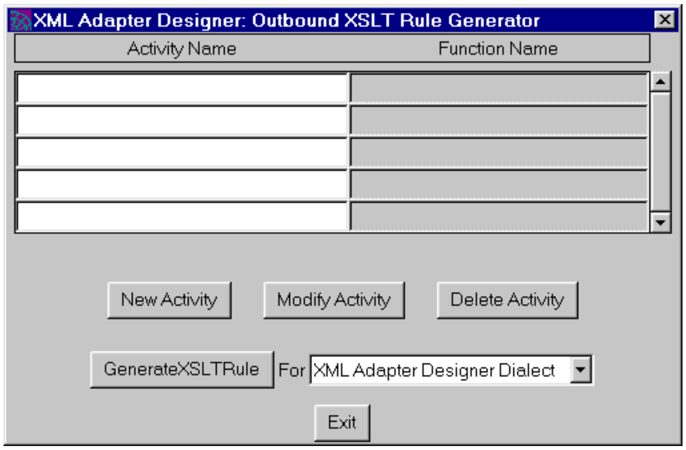


Interface Name : XADDemo_Adapter

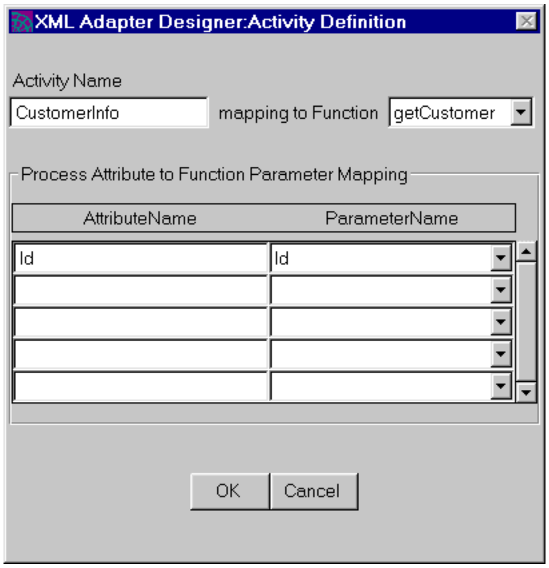
Name
getCustomer
transfer

Name	Array	Type	Mechanism	Required	Position
Id		int	INPUT	TRUE	0
AccountNo		int	INPUT	TRUE	0
FromAccount		{String}	INPUT	TRUE	0
ToAccount		{String}	INPUT	TRUE	0
Amount		int	INPUT	TRUE	0
Result		{String}	RETURN	TRUE	0

3. Select Generation > Outbound XSLT Rules from the menu. The XSLT Rule Generator dialog box appears.



4. Click on New Activity. The Activity Definition dialog box opens.


XML Adapter Designer:Activity Definition
Process Attribute to Function Parameter Mapping
OK
Cancel

Activity Name: CustomerInfo

mapping to Function: getCustomer

AttributeName	ParameterName
Id	Id

Mapping iIS Activities to Functions

You use this dialog box to map the names of activities in an iIS process definition to the functions in the API definition displayed in the Interface Editor. Enter the exact name of the iIS activity in the Activity Name field and select the function you are mapping it to from the drop list. For more information about process definitions and activities, see the *iIS Process Development Guide*.

Mapping iIS Attributes to Function Parameters

The mapping area displays the function's parameter names and by default assigns attribute names that match the parameter names. Modify the attribute names so that they match exactly the attribute names in the activity. The names and data types of process attributes must match exactly or the iIS process engine will not be able to use them.

When defining activities for outbound rules, only input parameters are visible. Since outbound rules transform documents from the iIS process engine into requests to the adapter, these documents only contain parameters passed to the adapter. Click on OK.

5. The Outbound XSLT Rule Generator now displays the names of the mapped activity and function names. You can map another activity and function by clicking New Activity and selecting another function name.
6. When you have finished mapping activities and functions, select the XML dialect that you want to use to generate the rule from the drop list. The Generator supports the XAD XML dialect for XSLT rules.
7. Click on Generate XSLT Rule. The dialog box asks you to save the generated rule to a file. The default file name is *package*`_Outbound.xsl`.

Modifying and Deleting an Activity Definition

You can also modify or delete the activity definition by right-clicking in the Activity Name field. If you select Modify, the Activity Definition Window opens. If you select Delete, you must confirm that you want to delete that activity definition prior to deleting it.

➤ **To generate inbound XSLT rules**

1. Follow the steps above, selecting Generation > Inbound XSL rules in step 3.

   When you map activities to functions to generate Inbound rules, only the output parameters are visible. These are the parameters returned by the adapter in its response document which is transformed using the inbound XSLT rules and then sent to the iIS process engine.

   The default name of the generated file is *packagename*`_Inbound.xsl`.

The XML definition file, including the generated XSLT rules is displayed below. The XSLT rules are defined within the `XSLT_OUTBOUND_MAPPING` and `XSLT_INBOUND_MAPPING` elements.

Code Example 4-1 XML Definition for a Simple API

```
<FFX_PACKAGE VERSION="1.0" PACKAGE_NAME="Customer">
  <EXECUTOR/>
  <FUNCTION>
    <FUNCTION_NAME>getCustomer</FUNCTION_NAME>
    <PARAMETER_LIST>
      <ITEM MECHANISM="INPUT" REQUIRED="TRUE" POSITION="1">
        <PARAMETER NAME="Id" DATATYPE="{Integer}"/>
      </ITEM>
      <ITEM MECHANISM="OUTPUT" REQUIRED="TRUE" POSITION="2">
        <PARAMETER NAME="Name" DATATYPE="{String}"/>
      </ITEM>
```

Code Example 4-1 XML Definition for a Simple API *(Continued)*

```
      </PARAMETER_LIST>
      <XSLT_OUTBOUND_MAPPING>
        <ACTIVITY>
          <ACTIVITY_NAME>CustomerInfo</ACTIVITY_NAME>
          <MAP PROCESSATTRIBUTE="Id" PARAMETER="Id"/>
        </ACTIVITY>
      </XSLT_OUTBOUND_MAPPING>
      <XSLT_INBOUND_MAPPING>
        <ACTIVITY>
          <ACTIVITY_NAME>CustomerInfo</ACTIVITY_NAME>
          <MAP PROCESSATTRIBUTE="Name" PARAMETER="custName"/>
        </ACTIVITY>
      </XSLT_INBOUND_MAPPING>
    </FUNCTION>
  </FFX_PACKAGE>
```

Saving and Reloading the Activity-to-Function Mappings

When you save the interface definition by selecting File > Save in the Interface Editor window, the activity-to-function mappings you have created are also saved in the XML document generated by the Interface Editor. When you open the XML definition, the activity-to-function mappings are reloaded and are visible when you select Generation > Inbound XSLT Rules or Generation > Outbound XSLT Rules.

About Sample XML Files

You can generate the following sample XML files for each operation in your API definition:

- XML request
- XML response
- iIS state document

You can then use these sample XML files for testing purposes. For example, you can send a sample XML request to your generated adapter using the test client. You can then compare the XML response returned by the adapter to the sample XML response you generated. See "Testing the Adapter and Transport Driver" on page 125 for an example of using the test client to send XML requests to an adapter.

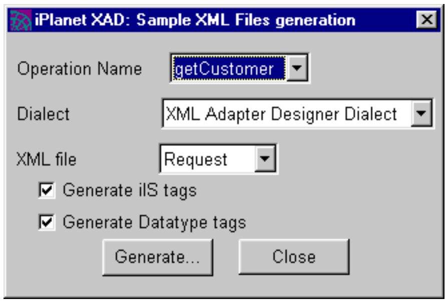
If your adapter's client is an iIS application, you can test your generated XSLT rules by applying them to the sample iIS state documents and XML responses. See "Testing XSLT Rules" on page 55 for more information on this topic.

Generating Sample XML Files

➤ **To generate an XML sample file**

1. Open an API definition file in the Interface Editor. From the menu, select Generation > Sample XML Files.

   The Sample XML Files Generation dialog box opens.


iPlanet XAD: Sample XML Files generation

Operation Name: getCustomer

Dialect: XML Adapter Designer Dialect

XML file: Request

☑ Generate iIS tags

☑ Generate Datatype tags

2. Fill out the fields in the dialog box as follows:

Field	Description
Operation Name	The operation for which the request is generated.
Dialect	The XML dialect in which the request is generated. Options are: • XML Adapter Designer Dialect • SOAP Dialect (a subset of SOAP v 1.1)
XML File	The type of XML file that is generated. Options are: • Request • Response • State Document

Field	Description
Generate iIS Tags	Select this option to generate an `iISInfo` element in your generated request or response (generated state documents always contain an `iISInfo` element). The `iISInfo` element is used to hold the activity and process IDs created by the iIS process engine. The `iISInfo` element can also hold other types of information if required by your application.
Generate Datatype Tags	`Datatype` tags are used to identify the data types of parameters. If you already know the data types an application is returning and want to avoid extra processing, disable this option. Otherwise, your generated requests will contain `Datatype` elements (generated state documents and SOAP dialect requests always contain `Datatype` elements).

3. Click Generate.

   A file browser opens for you to choose a location and name for the file.

4. Save the file in the *adapterName*`\xml` directory.

   The default file name is *operationName*`.xml`, where *operationName* is the name of the operation for which you are generating the sample file.

5. You can generate sample files for each operation in the Operation field drop list. When you finish, click Close.

The Generated Sample XML Files

Code Example 4-2 shows a generated sample XML request. You can test your adapter by sending it sample XML requests with the test client.

Code Example 4-2 Sample XML Request for `getCustomer` Operation

```
<?xml version="1.0" encoding="UTF-8"?>
<XADMsg Version="2.0">
  <COMMAND>
    <iISInfo>
      <Process Id="theProcessId" Name="theProcessName"/>
      <Activity Id="ActivityId" Name="theActivity"/>
    </iISInfo>
    <Operation Name="getCustomer">
```

Code Example 4-2 Sample XML Request for `getCustomer` Operation *(Continued)*

```
      <PList>
        <Param Name="Id" Type="{Integer}">1</Param>
      </PList>
    </Operation>
  </COMMAND>
</XADMsg>
```

Code Example 4-3 shows a generated sample XML response. You can use your sample XML responses to validate inbound XSLT rules in an iIS client.

The XAD Generator inserts example data in both these files for the `Process Id`, `Process Name`, `Activity Id`, `Activity Name`, and `Param` content. In a production environment, this information comes from the application.

Code Example 4-3 `getCustomer` Operation Sample XML Response

```
<?xml version="1.0" encoding="UTF-8"?>
<XADMsg Version="2.0">
  <ANSWER>
    <iISInfo>
      <Process Id="theProcessId" Name="theProcessName"/>
      <Activity Id="ActivityId" Name="theActivity"/>
    </iISInfo>
    <Operation Name="getCustomer">
      <PList>
        <Param Name="Name" Type="{String}">sample string</Param>
      </PList>
    </Operation>
  </ANSWER>
</XADMsg>
```

Code Example 4-4 shows a generated sample iIS state document. You can use your sample state documents to validate outbound XSLT rules in an iIS client. The iIS process engine sends state documents to the proxy to be sent to the adapter. The outbound XSLT rule transforms the state documents into requests that the adapter understands.

Code Example 4-4 Sample State Document

```
<?xml version="1.0" encoding="UTF-8"?>
<FNState Version="1.0">
  <FNCndState State="ActivityStarted">
    <FNIdentity ProcessID="123" ProcessName="XAD_DEMO_PD"
    ActivityID="456" ActivityName="Custom_Activity"
    ActivityAppCode=""/>
    <FNProcessAttributeList>
      <FNProcessAttribute Name="Id"
      Type="Integer">58</FNProcessAttribute>
      <FNProcessAttribute Name="Name" Type="String">sample
      string</FNProcessAttribute>
    </FNProcessAttributeList>
  </FNCndState>
</FNState>
```

Testing XSLT Rules

This section explains how to test the inbound and outbound XSLT rules generated for the `getCustomer` function shown in the previous section. You can manually import the generated XSLT rules and XML sample documents into the iIS Development Workshop. You can then test the rules and deploy them into an iIS backbone.

➤ **To test the sample rules in an iIS backbone**

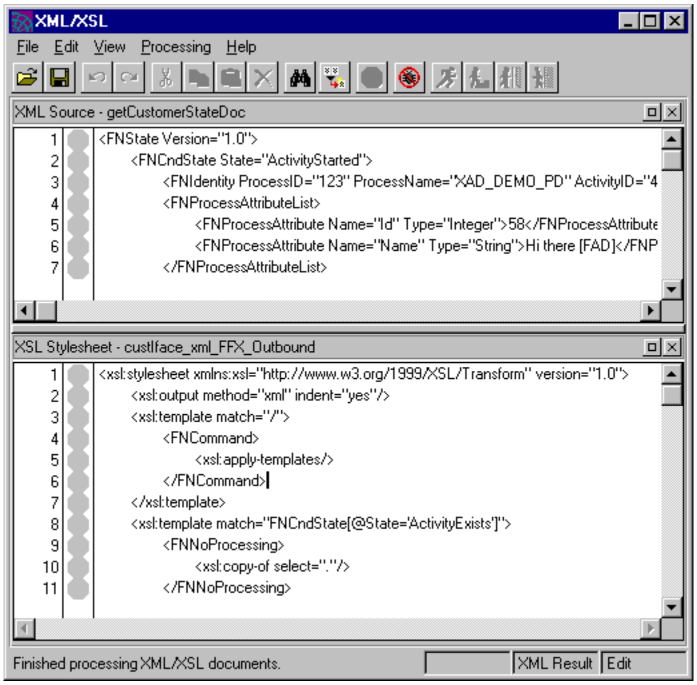
1. In your UDS workspace, select Plan > New Process Development Plans > XML/XSL Plan. Enter a name for your plan in the New XML/XSL Plan dialog box and click OK.

2. Select Component > Import XML/XSL Plan and select one of your generated XSLT rules. Repeat this step for each generated rule and sample XML response and state document.

Testing Outbound XSLT Rules

In the example below, we import the outbound XSLT rule we created for the `getCustomer` method. This rule transforms state documents going from the iIS process engine to the adapter, so we test it against the sample state document generated for the `getCustomer` method.

3. Double click on the Outbound XSLT rule. The outbound rule document opens and an Open Second Document dialog appears. Select `getCustomerStateDoc`. This document opens in a second window.



XML Source - getCustomerStateDoc

```
<FNState Version="1.0">
    <FNCndState State="ActivityStarted">
        <FNIdentity ProcessID="123" ProcessName="XAD_DEMO_PD" ActivityID="4
        <FNProcessAttributeList>
            <FNProcessAttribute Name="Id" Type="Integer">58</FNProcessAttribute
            <FNProcessAttribute Name="Name" Type="String">Hi there [FAD]</FNP
        </FNProcessAttributeList>
```

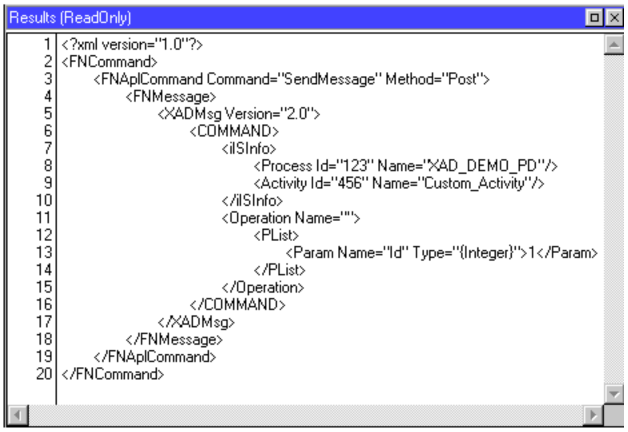
XSL Stylesheet - custIface_xml_FFX_Outbound

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="xml" indent="yes"/>
    <xsl:template match="/">
        <FNCommand>
            <xsl:apply-templates/>
        </FNCommand>
    </xsl:template>
    <xsl:template match="FNCndState[@State='ActivityExists']">
        <FNNoProcessing>
            <xsl:copy-of select="."/>
        </FNNoProcessing>
```

Finished processing XML/XSL documents. | XML Result | Edit

4. Validate each document by activating its window and choosing Processing > Validate. Then select Processing > Process XML/XSL. If the state document is successfully transformed, a third window displays the document that results from applying the outbound rule to the state document. This document is the request that will be sent to the adapter which is enclosed in the `XADMSG` element within the `FNMessage` element.

The results document is shown in the following diagram.



```
<?xml version="1.0"?>
<FNCommand>
    <FNAplCommand Command="SendMessage" Method="Post">
        <FNMessage>
            <XADMsg Version="2.0">
                <COMMAND>
                    <ilSInfo>
                        <Process Id="123" Name="XAD_DEMO_PD"/>
                        <Activity Id="456" Name="Custom_Activity"/>
                    </ilSInfo>
                    <Operation Name="">
                        <PList>
                            <Param Name="Id" Type="{Integer}">1</Param>
                        </PList>
                    </Operation>
                </COMMAND>
            </XADMsg>
        </FNMessage>
    </FNAplCommand>
</FNCommand>
```

Observing the Transformation Processing

You can also observe the line-by-line processing of the XML state document and the building of the new XML request document by choosing Processing > Debug. Cursors appear in both the state document and the rules document. Each time you click on Step In, the cursors move to show you what line they are processing in each document. As the processor finds matching patterns, it creates the action specified in the rules and builds the output document in the results window. Each time it encounters the `<xsl:apply-templates/>` tag, it finds the next tag to be processed, goes back to the beginning of the XSLT rules and searches for matching patterns from the beginning of the rules.

Outbound Rule Patterns

In the XAD-generated outbound rules, when the processor finds the root node, specified by a `/` symbol, it creates an `FNCommand` element in the results document. It next looks for an `FNCndState` element with the value `Activity Exists`. It does not find this pattern in the `getCustomer` state document so it ignores the action specified. It does find an `FnCndState` element with the value `ActivityStarted` so it performs the action specified. This action is to create an `FNAplCommand` with Command and Method attributes. These attributes have the values `Send Message` and `Post`, both of which are application commands understood by the proxy. In the results document, the line

```
<FNAplCommand Command=”Send Message” Method=”Post”>
```

appears. The processor adds the `FNMessage` and `XADMsg` tags to the results document and starts applying the rules from the beginning again.

The processing continues, matching the pattern `FNIdentity` and adding an `iISInfo` element. This element contains part of the message sent to the adapter. It contains a `Process` element with `ID` and `Name` attributes whose values are taken from the `ProcessID` and `ProcessName` elements in the state document. It also contains an `Activity` element with `Id` and `Name` attributes whose values are also taken from the state document.

The processor performs similar steps of matching and creating elements in the results document for the `FNProcessAttributeList` and `FNProcessAttribute` element. The Operation element’s `Name` attribute value is the name of the application dictionary item associated with the given activity. When the processing completes, the closing tags are generated.

Processing Inbound Rules

To test the processing of inbound rules, open the inbound rule and the generated response document and follow the steps described above. If the processing is successful, the adapter’s response is transformed into a command to the iIS process engine. In Code Example 4-5, the `CompleteActivity` process engine command is obtained by transforming the `getCustomer` response. Parameter information from the response document is transformed into `FNProcessAttribute` elements.

Code Example 4-5 Sample Process Engine Command Document

```
<?xml version="1.0"?>
<FNCommand>
  <FNCndCommand Command="CompleteActivity">
    <FNIdentity ProcessID="theProcessId"
ProcessName="theProcessName" ActivityID="ActivityId"
ActivityName="theActivity"/>
              <FNProcessAttribute Name="Name"
Type="TextData">sample text</FNProcessAttribute>
  </FNCndCommand>
</FNCommand>
```

Generated XSLT Rules

Code Example 4-6 shows the generated XSLT rules for the example in the XAD dialect.

Code Example 4-6 Outbound XSLT Rule XAD Dialect

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:output method="xml" indent="yes"/>
  <xsl:template match="/">
    <FNCommand>
      <xsl:apply-templates/>
    </FNCommand>
  </xsl:template>
  <xsl:template match="FNCndState[@State='ActivityExists']">
    <FNNoProcessing>
      <xsl:copy-of select="."/>
    </FNNoProcessing>
  </xsl:template>
  <xsl:template match="FNCndState[@State='ActivityStarted']">
    <FNAplCommand Command="SendMessage" Method="Post">
      <FNMessage>
        <XADMsg Version="2.0">
          <COMMAND>
            <xsl:apply-templates/>
          </COMMAND>
        </XADMsg>
      </FNMessage>
    </FNAplCommand>
  </xsl:template>
  <xsl:template match="FNIdentity">
    <iISInfo>
      <xsl:element name="Process">
        <xsl:attribute name="Id">
          <xsl:value-of select="@ProcessID"/>
        </xsl:attribute>
        <xsl:attribute name="Name">
          <xsl:value-of select="@ProcessName"/>
        </xsl:attribute>
      </xsl:element>
      <xsl:element name="Activity">
        <xsl:attribute name="Id">
          <xsl:value-of select="@ActivityID"/>
        </xsl:attribute>
        <xsl:attribute name="Name">
          <xsl:value-of select="@ActivityName"/>
        </xsl:attribute>
      </xsl:element>
    </iISInfo>
  </xsl:template>
  <xsl:template match="FNProcessAttributeList">
    <xsl:element name="Operation">
```

Code Example 4-6 Outbound XSLT Rule XAD Dialect *(Continued)*

```
      <xsl:attribute name="Name">
        <xsl:value-of select="//@ActivityAppCode"/>
      </xsl:attribute>
      <PList>
        <xsl:apply-templates/>
      </PList>
    </xsl:element>
  </xsl:template>
  <xsl:template
match="FNProcessAttributeList/FNProcessAttribute[@Name='Id']">
    <Param>
      <xsl:attribute name="Name">Id</xsl:attribute>
      <xsl:attribute name="Type">{Integer}</xsl:attribute>
      <xsl:value-of select="."/>
    </Param>
  </xsl:template>
  <xsl:template match="text()"/>
</xsl:stylesheet>
```

The Inbound rules transform the adapter's response into a document that the process engine can understand. It finds the `ANSWER` tag, builds `FNCNDCommand` and `FNIdentity` elements copying the values of the attributes into the response document's `iISInfo` element. It then copies the parameter values into the corresponding process attributes as specified in the mapping, for example, creating a `Type` attribute with the value `TextData`.

Code Example 4-7 Inbound Rules: XAD Dialect

```
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:output method="xml" indent="yes"/>
  <xsl:template match="/">
    <FNCommand>
      <xsl:apply-templates/>
    </FNCommand>
  </xsl:template>
  <xsl:template match="ANSWER">
    <FNCndCommand Command="CompleteActivity">
      <xsl:apply-templates/>
    </FNCndCommand>
  </xsl:template>
  <xsl:template match="ANSWER_ERROR">
    <FNCndCommand Command="AbortActivity">
      <xsl:apply-templates/>
    </FNCndCommand>
  </xsl:template>
  <xsl:template match="iISInfo">
    <FNIdentity>
      <xsl:attribute name="ProcessID">
        <xsl:value-of select="./Process/@Id"/>
      </xsl:attribute>
      <xsl:attribute name="ProcessName">
        <xsl:value-of select="./Process/@Name"/>
      </xsl:attribute>
      <xsl:attribute name="ActivityID">
        <xsl:value-of select="./Activity/@Id"/>
      </xsl:attribute>
      <xsl:attribute name="ActivityName">
        <xsl:value-of select="./Activity/@Name"/>
      </xsl:attribute>
    </FNIdentity>
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="Param">
    <xsl:choose>
      <xsl:when test="@Name='Name'">
        <FNProcessAttribute>
          <xsl:attribute name="Name">Name</xsl:attribute>
          <xsl:attribute name="Type">TextData</xsl:attribute>
          <xsl:value-of select="."/>
        </FNProcessAttribute>
      </xsl:when>
    </xsl:choose>
  </xsl:template>
</xsl:transform>
```

Chapter 5

Communicating With the Adapter: HTTP and JMS

This chapter describes the procedures you must perform to setup communication between your client and adapter.

Client applications communicate with XAD Java adapters using the HTTP protocol or Java Messaging Service (JMS). XAD provides both an HTTP transport driver and a JMS transport driver that can locate and forward requests to any XAD Java adapter.

Client applications communicate with XAD C adapters using the HTTP protocol. XAD C adapters contain an integrated HTTP server for communicating with the client.

Communicating With a Java Adapter Over JMS

The procedures described in this section refer to scripts provided with your installation of XAD. Some of these scripts were created to run the example application but can be modified to work with any adapter.

Prerequisites

The procedures in this chapter assume that you have already installed supported implementations of JMS and LDAP (if you are using LDAP as your JNDI provider). See the XAD platform support matrix at `http://docs.iplanet.com/docs/manuals/xad.html` for more information on supported software.

Set Up Procedures

Setting up the transport layer requires the following procedures:

- Setting required environment variables
- Setting up topics and queues
- Registering topics and queues
- Setting the JMS Transport Driver's runtime properties

➤ To set up the environment

1. Set the values of the `XAD_JMS_BROKER` variable and the `XAD_JMS_LIB` variable as indicated in Table 5-1.

   `XAD_JMS_BROKER` is located in the *XAD_ROOT*`/install/bin/xad_jms_env` script and `XAD_JMS_LIB` is located in the *XAD_ROOT*`/install/bin/xad_env` script.

Table 5-1 XAD Environment Settings

Variable	Value
`XAD_JMS_BROKER`	The name of the system where the JMS broker is running, for example, `jupiter.varrius.com`
`XAD_JMS_LIB`	The location where the JMS broker messaging client libraries are installed. The default value is `%JMQ_HOME%\LIB` on Windows platforms or `$JMQ_HOME/LIB` on Solaris™ platforms.

The XAD JMS transport driver (`XADJMSRouter`) uses the JMS broker software's messaging client libraries, so the JMS broker must be installed and `XAD_JMS_LIB` must be set on the system running the JMS transport driver. This assumes, that the JMS transport driver is on the same system as the adapter. However, the transport driver can connect to a broker on a different system by setting `XAD_JMS_BROKER` to the name of that system; however, you still need the client libraries on the same system as the JMS Transport Driver.

The `XAD_JMS_BROKER` setting is also used by the naming service and the `xad_jmq2register` script to register topics and queues.

2. Make sure the `XAD_JMS_LIB` variable reflects the client libraries used by your JMS broker software.

   If you are using iMQ 2.0, these libraries are contained in `jmq.jar`. Specifically, the libraries the XAD Transport Driver uses are: `fscontext.jar`, `jms.jar`, `jndi.jar`, `providerutil.jar`, and `ldap.jar`. If you are using a different JMS broker, modify the `xad_run_jms` script and set the `JMQ_CPath` variable to the equivalent libraries that your software uses.

Setting up Queues and Topics

You can communicate over JMS using either queues or topics. Use queues for point-to-point communication where only the client sending the request receives a reply. Use topics where you want any subscriber to be able to listen for events of interest.

At this point, if you have already set up whatever topics and queues your system is using and you have registered them with one of the supported naming services, you can skip ahead to the section "Setting the JMS Transport Driver's Runtime Properties" on page 67 in this chapter.

➤ **To set up queues**

1. You set up queues in the `XADDemo_Adapter.properties` file located in *XAD_ROOT*`\install\java\props\jms\XADDemo_Adapter`. This file sets the JMS Transport Driver's runtime properties.

   Open the file in a text editor and locate the lines:

   ```
   Queue = XAD_Q

   ReplyQueue = XAD_REPLY_Q
   ```

2. You then set up the queues in the `xad_jms_env` script located in *XAD_ROOT*`\install\bin`. Open the file in a text editor and locate the lines:

   ```
   set XAD_JMS_Q_NAME=XAD_Q

   set XAD_JMS_RQ_NAME=XAD_REPLY_Q
   ```

   Set the value of `XAD_JMS_Q_NAME` and `XAD_JMS_RQ_NAME` to the same queue and reply queue names that you used in the properties files. `XAD_JMS_RQ_NAME` is used only if the client does not send a Reply Queue.

➤ **To set up topics**

1. As with queues, you set up topics in the properties file `XADDemo_Adapter.properties` located in *XAD_ROOT*`\install\java\props\jms\XADDemo_Adapter`. Open the file in a text editor and locate the lines:

   ```
   Topic = TestTopic
   ReplyTopic = OutTopic
   ```

2. Set the value of `Topic` to the topic name and `ReplyTopic` to the reply topic name.

   `ReplyTopic` is used only if the client does not send its own reply topic.

3. Change the value of the `ModeOfCommunication` property to `pubsub`.

4. You then set up the topics in the `xad_jms_env` script located in *XAD_ROOT*`\install\bin`. Locate the following lines:

   ```
   set XAD_JMS_Q_NAME=XAD_Q
   set XAD_JMS_RQ_NAME=XAD_REPLY_Q
   ```

5. Set the value of `XAD_JMS_Q_NAME` and `XAD_JMS_RQ_NAME` to the same values you set `Topic` and `ReplyTopic` in the properties file. `XAD_JMS_RQ_NAME` is only used if the client does not send a reply topic.

6. Change the value of the `JMS_DOMAIN` variable to `PublishSubscribe`.

Registering Queues and Topics

Prior to registering the topics and queues with the naming service, you must also provide information about your JNDI provider. You can use either a file system provider or an LDAP Server.

➤ **To set the JNDI provider**

1. You set up queues in the *XAD_ROOT*`\install\bin\xad_jms_env` script. Open this file in a text editor and locate the lines:

   ```
   set JNDI_URL=file://%XAD_ROOT%/install/java/props/jms
   /XADDemo_Adapter
   set JNDI_PROVIDER=file
   ```

2. `JNDI_PROVIDER` should be set to `file` if using the file system as your provider, and `ldap` if using LDAP as your provider.

3. `JNDI_URL` should be set to the location of your provider. For example, if you are using LDAP, the format of the URL should be similar to the one provided in the file:

   ```
   ldap://ft7llez.varrius.com:389/o=varrius.com
   ```

   If you are using the file system, specify the location of the properties file as shown in Step 1.

➤ **To register topics and queues**

1. Ensure that your queues (or topics) are registered in the naming service.

   If you are using iMQ 2.0, you can perform this task by running the script `xad_jmq2register` located in *XAD_ROOT*`\install\bin`.

   If you are using a different product than iMQ 2.0, refer to that product's documentation for information about registering topics and queues.

2. If the file system is your JNDI provider, verify that a `.bindings` file has been created in the *XAD_ROOT*`\install\java\props\jms\`*XAD_AdapterName* directory.

3. If LDAP is your JNDI provider, use your LDAP server administration tools to verify that your queues (or topics) are registered.

Setting the JMS Transport Driver's Runtime Properties

You set the adapter's runtime properties by copying and renaming the `XADDemo_Adapter.properties` file in *XAD_ROOT*`\install\java\props\jms\XADDemo_Adapter` and modifying it according to your system. See "Creating an Adapter Directory Structure" in Chapter 3, "Building an Adapter."

The table below specifies the properties that can be modified to work with any adapter. In most cases the file lists the properties twice with different values, so you can use the comment mark (#) to specify that whatever value you don't want to use be ignored. Properties with an asterisk must be set to the same values that were set in the `xad_jms_env` script when registering topics and queues in the previous section.

➤ **To set the runtime properties**

1. Open `XADDemo_Adapter.properties` in a text editor and modify according to the description in Table 5-2 on page 68.

   You must provide a value for all the required properties or the JMS Transport Driver will not run. If a property is not required and you do not set a value, the Transport Driver will use the default value.

Table 5-2 JMS Transport Driver Runtime Properties

Property	Value	Required	Default Value
`AdapterName`	Name of the generated adapter	Yes	None
`JNDIPrefix`	Prefixed to the value of `AdapterName` to construct a unique adapter name before `AdapterName` is looked up in the JNDI provider.	No	None
`ActionFile`	Name of file for logging output.	No	`jms_XADDemo_Adapter_act.log`
`ErrorFile`	Name of file for logging errors.	No	`jms_XADDemo_Adapter_err.log`
`logging`	`true` to turn logging on. `false` to turn logging off.	No	`false`

`JNDIContextJMS`	JNDI service provider where topics and queues are registered:
	`com.sun.jndi.ldap.LdapCtxFactory`
	or
	`com.sun.jndi.fscontext.RefFSContextFactory`
		Yes	None

`JNDIProviderJMS`	URL of `JNDIContextJMS`, such as:
	`ldap://ft7llez.varrius.com:389/o=varrius.com`
	or
	`file://D:/XAD/install/java/props/jms/XADDemo_Adapter`
		Yes	None

`AcknowledgeMode`	Acknowledgement type: `client` or `auto`	No	`Auto`
`SubscriptionID`	Subscription ID for topics	No	Transport Driver creates a non-durable `TopicSubscriber`.

Table 5-2 JMS Transport Driver Runtime Properties *(Continued)*

Property	Value	Required	Default Value
`ModeOfCommunication`[1]	`ptp` for point-to-point. `pubsub` for publish and subscribe.	Yes	None
`TopicConnectionFactory`	Value used to register the topic connection with the JNDI service provider.	Yes, if `ModeOfCommunication = pubsub`	None
`Topic`	Name used to register topic with the JNDI service provider.	Yes, if ModeOfCommunication = `pubsub`	None
`ReplyTopic`	Name used to register reply topic with the JNDI service provider. Also used as default outbound reply topic if not supplied in client's request.	Yes, if ModeOfCommunication = `pubsub`	None
`QueueConnectionFactory`	Value used to register the queue connection with the JNDI service provider.	Yes, if ModeOfCommunication = `ptp`	None
`Queue`[2]	Name used to register the queue with the JNDI service provider.	Yes, if ModeOfCommunication = `ptp`	None
`ReplyQueue`[3]	Name used to register the queue with the JNDI service provider. Also used as default outbound reply queue if not supplied in client's request.	Yes, if ModeOfCommunication = `ptp`	None
`JNDIContextConnector`	JNDI provider for adapter, for example: `com.sun.jndi.rmi.registry.RegistryContextFactory`	No	value of `JNDIContextJMS`
`JNDIProviderConnector`	URL of JNDIContextConnector, for example: `rmi://localhost:1099/`	No	value of `JNDIProviderJMQ`

1. If you specify ptp, the values for QueueConnectionFactory, Queue and ReplyQueue are required. If you specify pubsub, the values for TopicConnectionFactory, Topic and ReplyTopic are required.
2. The value set here must match the XAD_JMS_Q_NAME value set in the xad_jms_env script if you used the jmq2register script to register your queues.
3. The value set here must match the XAD_JMS_RQ_NAME value set in the xad_jms_env script if you used the jmq2register script to register your queues.

➤ **To run the JMS transport driver**

1. Ensure that the RMI registry and the adapter are running before you start the JMS listener. If you are using LDAP, also ensure that the LDAP server is running.
2. In a text editor, open the `setadapterenv` script located in *XAD_ROOT*`\adapters\XADDemo_Adapter\java` and locate the lines:

   ```
   XAD_ADAPTER_NAME=

   XAD_JMS_PROP_FILE =
   ```
3. Change the value of `XAD_ADAPTER_NAME` to the name of your adapter and set the value of `XAD_JMS_PROP_FILE` to the location of your adapter properties file.
4. Run the `setadapterenv` script.
5. Run the `xad_run_jms` script.

Communicating With a Java Adapter Over HTTP

To communicate with a Java adapter using HTTP or HTTPS, XAD provides a listener servlet that you deploy in a web server. An external application sends an XML request to the servlet, which forwards the request to the adapter. The adapter returns a response to the servlet, which then communicates the response back to the calling application.

Listener Servlet Features

The servlet class, `XADServlet`, provides the features listed below.

- You can use it to communicate with any Java adapter developed in XAD.
- You can use a single listener servlet deployment to communicate with multiple adapters running in your environment.
- It automatically updates information about the adapter’s location and JNDI service provider without requiring the user to stop and restart the servlet or web server.
- You can create secure communication between the client application and `XADServlet` by deploying it in an HTTPS-enabled web server instance.

Listener Servlet Properties

The listener servlet, `XADServlet`, loads the information it needs to locate the adapter from a properties file. The properties file contains information about how to locate the adapter such as the type of JNDI service provider it is using, and the system name and the port number on which the JNDI provider is running.

JNDI Provider

The listener servlet uses Java RMI as the JNDI provider. The RMI registry and the adapter, must be on the same system.

Locating the Adapter

The servlet looks for an adapter using the query string it receives as part of a request. In the URL,

`http://`*node*`:`*portnumber*`/XADServlet?`*adaptername*

the query string is *adaptername* and corresponds to the name of the adapter. For example, in the following URL,

```
http://jupiter.varrius.com/adapter/XADServlet?XADDemo_Adapter
```

`jupiter.varrius.com` is the name of the system where the web server is running, `/adapter` is the virtual path of the servlet in the web server, `XADServlet` is the name of the servlet and `XADDemo_Adapter` is the name of the adapter the servlet is asked to locate.

When the servlet receives a request for an adapter, it checks to see if it has the JNDI information for the adapter in memory. If it does not, it loads this information from the properties file into memory. The servlet then extracts the trailing adapter name from the URL and uses RMI to locate that name and retrieve an object reference to the corresponding adapter.

Use the following instructions to modify the properties file.

Deploying the XAD Listener Servlet

If you are using HTTP to pass messages between the iIS client and the adapter, you must deploy the XAD listener servlet. Deploying the listener servlet entails:

- Configuring the servlet's properties file
- Installing the servlet on the web server
- Configuring the servlet attributes on the web server

You need to deploy only one listener servlet for all of the XAD Java adapters running in your environment. The listener servlet and its utility classes are provided in the `xadservlet.jar` and `xadutil.jar` files located in the *XAD_ROOT*`/install/java/lib` directory.

Configuring the Listener Servlet Properties

When configuring the listener servlet properties, you provide the JNDI provider information in the properties file. You must use the same JNDI provider that you used when registering the adapter.

➤ **To set the listener servlet properties**

1. Copy the `XADServlet.properties` file from *XAD_ROOT*`\install\java\props\servlet` to the configuration file directory of your web server. This is the directory where the virtual machine used by the web server is running. If you are using the iPlanet Web Server, for example, the directory could be `c:\iplanet\servers\https-`*machineName*`\`config. For other products, consult the documentation for that product.

2. Open the file in a text editor and locate the following lines:

```
JNDIPrefix =

JNDIContextConnector =
com.sun.jndi.rmi.registry.RegistryContextFactory

JNDIProviderConnector = rmi://localhost:1099/
```

3. Edit the `JNDIProviderConnector` property as follows:

   `JNDIProviderConnector = rmi://`*hostname*`:`*portno*`/`

   Replace *hostname* with the name of the machine where the RMI registry is running and *portno* with the port number on which the RMI registry is listening. The default port is 1099.

   These entries must match the information provided in the adapter engine properties file.

4. Edit the `JNDIPrefix` property if needed.

   The servlet uses the value of the `JNDIPrefix` property to locate the adapter. It prefixes this value to the URL query string (which specifies the adapter name). If no value is assigned to `JNDIPrefix`, the query string alone is used to locate the adapter.

   You can use this property if you are using long names (for example, a name that includes a package name) to identify adapters in your JNDI provider. For example, you might use a long name when registering an adapter but provide only the adapter name in the request. The servlet prefixes the `JNDIPrefix` value to the adapter name in the URL to construct a unique adapter name before looking it up in the JNDI provider.

You can modify the contents of the servlet's properties file at any time. The servlet refreshes the information about the location of adapters in the environment whenever the properties file is modified.

Installing the Listener Servlet

The following procedure explains how to install the XAD listener servlet on iPlanet Web Server (iWS) and configure the servlet's attributes.

➤ **To install the XAD listener servlet and configure the servlet attributes**

1. If your web server and XAD software are on different machines, copy the following JAR files from the *XAD_ROOT*`/install/java/lib` directory to some location on the web server machine: `xadservlet.jar`, `jndi.jar`, `rmiregistry.jar`, `providerutil.jar`, `ejb.jar`, and `xadutil.jar` (you will later include these files in the web server's class path).

2. Register the directory containing the servlet JAR files.

   a. Start the iWS Administration Server.

   b. Open the iWS administration page in a web browser.

      This page is available at the following URL:

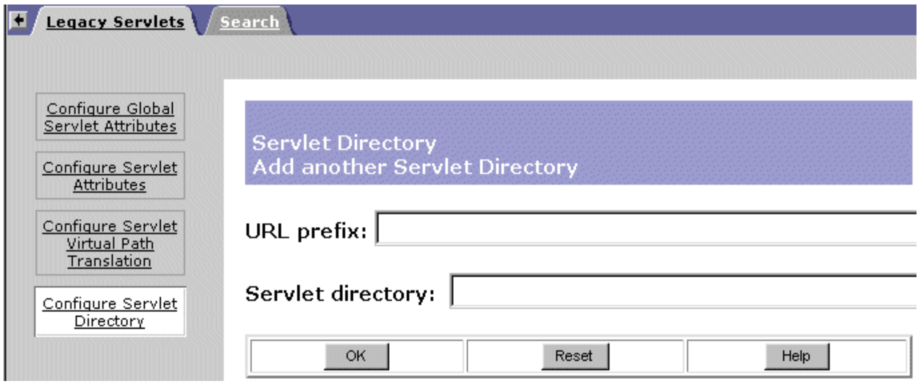
      `http://`*host*`:`*portnumber*`/https-admserv/bin/index`

      In this URL, replace *host* with the name of the machine on which you are running the web server. Replace *portnumber* with the port number on which the web server is running (the default is 8888).

   c. From the Select a Server drop-down list, select the web server in which you want to register the servlet and click Manage.

   d. Click the Legacy Servlets tab and then click the Configure Servlet Directory link on the left side.

      The Servlet Directory page opens.


Legacy Servlets
Search
Configure Global Servlet Attributes
Configure Servlet Attributes
Configure Servlet Virtual Path Translation
Configure Servlet Directory
OK
Reset
Help

Servlet Directory
Add another Servlet Directory

URL prefix:

Servlet directory:

   e. In the URL Prefix field, enter the prefix to be used for accessing the servlet directory, for example,

      `/adapter`

   f. In the Servlet Directory field, enter the absolute path to the directory containing the servlet, for example,

      `d:/XAD/install/java/lib`

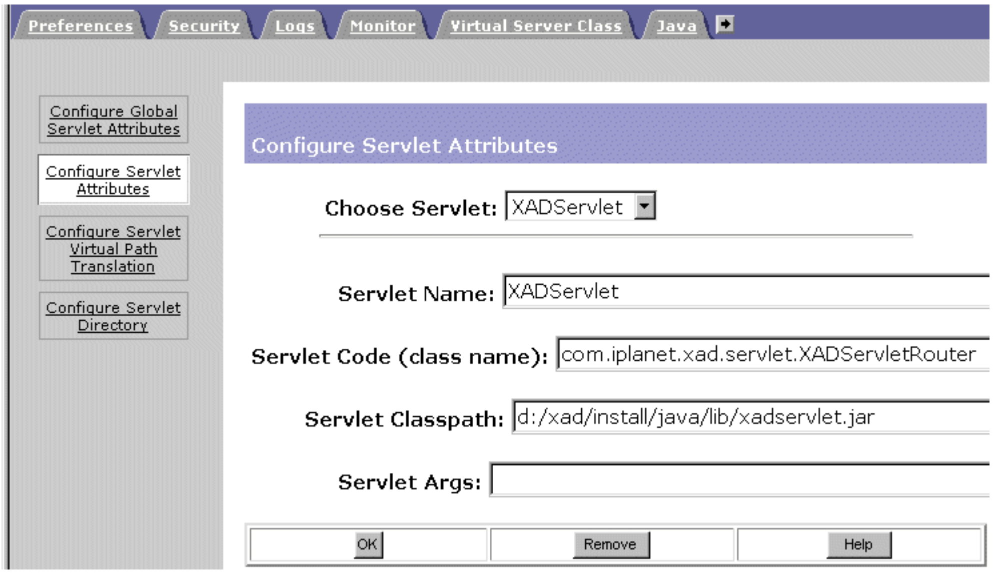
   g. Click OK.

      A dialog box opens to confirm the addition of the servlet directory.

3. Configure the servlet attributes.

a. Click on the Configure Servlet Attributes link.

The Configure Servlet Attributes page opens.


Preferences
Security
Logs
Monitor
Virtual Server Class
Java
Configure Global Servlet Attributes
Configure Servlet Attributes
Configure Servlet Virtual Path Translation
Configure Servlet Directory
OK
Remove
Help

Configure Servlet Attributes

Choose Servlet: XADServlet

Servlet Name: XADServlet

Servlet Code (class name): com.iplanet.xad.servlet.XADServletRouter

Servlet Classpath: d:/xad/install/java/lib/xadservlet.jar

Servlet Args:

b. Set the values for the fields on the page as follows:

Field	**Value**
Servlet Name	`XADServlet`
Servlet Code	`com.iplanet.xad.servlet.XADServletRouter`
Servlet Classpath	The absolute path to the servlet JAR file, for example, `d:/XAD/install/java/lib/xadservlet.jar`.
Server args	Leave this field blank. Server arguments are read from a properties file.

c. Click OK.

A dialog box opens to confirm that you have modified the servlet attributes.

4. Configure the servlet virtual path.

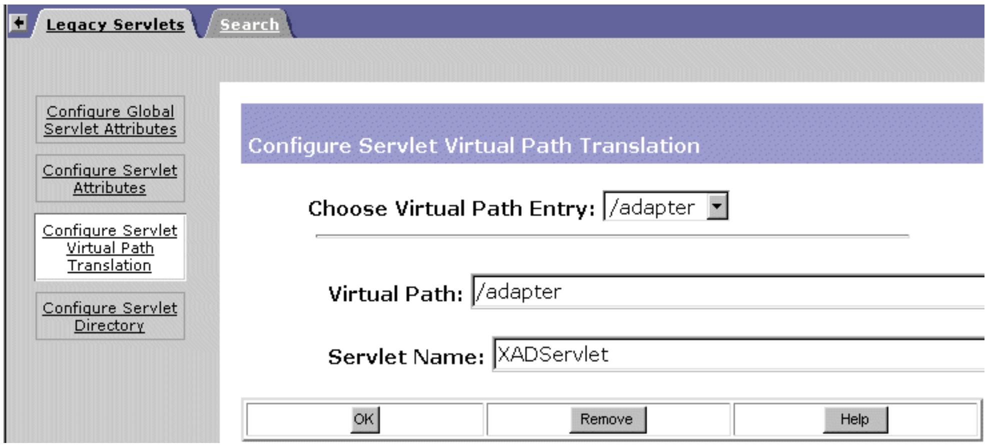
   a. Click on the Configure Servlet Virtual Path Translation link.

      The Configure Servlet Virtual Path Translation page opens.

   b. In the Virtual Path field, enter the prefix name as specified in the Servlet Directory page, for example:

      `/adapter`

   c. In the Servlet Name field, enter `XADServlet`.


Legacy Servlets
Search
Configure Global Servlet Attributes
Configure Servlet Attributes
Configure Servlet Virtual Path Translation
Configure Servlet Directory
OK
Remove
Help

Configure Servlet Virtual Path Translation

Choose Virtual Path Entry: /adapter

Virtual Path: /adapter

Servlet Name: XADServlet

   d. Click OK.

5. Configure the JVM attributes.

   a. Click the Java tab and then click on the Configure JVM Attributes link at the left.

      The Configure JVM Attributes page opens.

   b. Modify the Classpath field to include the location of the following JAR files:

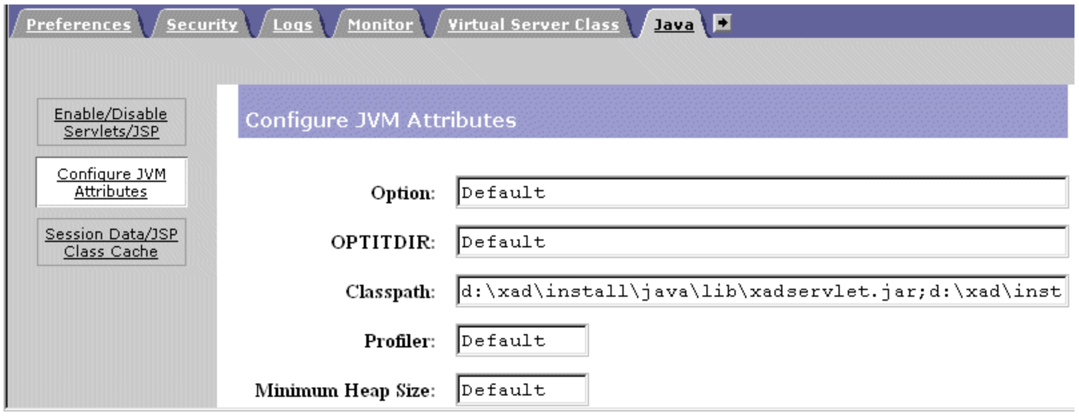
      `xadservlet.jar`, `jndi.jar`, `rmiregistry.jar`, `providerutil.jar`, `ejb.jar`, and `xadutil.jar`.

      These files are installed in the *XAD_ROOT*`/install/java/lib` directory. If your web server is on a different machine than your XAD installation, you should already have copied these files to your web server in Step 1.

For example, if you installed the XAD software in a directory named xad on the D drive of a Windows operation system and your web server is on the same machine, add the following paths to the Classpath field:

```
d:\xad\install\java\lib\xadservlet.jar;
d:\xad\install\java\lib\jndi.jar;
d:\xad\install\java\lib\rmiregistry.jar;
d:\xad\install\java\lib\providerutil.jar;
d:\xad\install\java\lib\ejb.jar;
d:\xad\install\java\lib\xadutil.jar;
```

c. Click OK.



Option:	Default
OPTITDIR:	Default
Classpath:	d:\xad\install\java\lib\xadservlet.jar;d:\xad\inst
Profiler:	Default
Minimum Heap Size:	Default

6. Copy the servlet properties and i18n properties files to your web server.

a. Copy the `XADServlet.properties` file from *XAD_ROOT*`\install\java\props\servlet` to the configuration file directory of your web server, for example, `iPlanet\servers\https-rex.varrius.com\config`.

b. Copy the `XADi18N.properties` file from *XAD_ROOT*`\install\java\props\i18n` to the configuration file directory of your web server (the same directory as in Step a).

7. Apply your configuration changes to the web server.
   a. Click the Apply link at the upper right of the page.

      The Apply Changes page opens.
   b. Click the Apply Changes button.

      A message dialog box opens to indicate that the server has restarted.
   c. Click OK.

Running Multiple Adapters

You can use the same instance of `XADServlet` to communicate with all the Adapters running in your environment.

➤ **To add an additional adapter**

1. Register the new adapter with the same JNDI service provider as the first adapter.
2. Send an XML request to the `XADServlet` using the new adapter name as the query string.

Troubleshooting

This section describes common problems that could occur when sending messages to an adapter through the servlet.

Properties File Not Found

If the servlet cannot locate the properties file, it writes an error message to the web server's log files. The message will specify the directory in which the properties file is to be placed. Move the properties file into that directory. In addition, the servlet returns the HTTP error 404 ("SC_NOT_FOUND") and a "Properties File not found" message to the requesting client.

Adapter Not Found

If the servlet looks up the location of the adapter, but the adapter is not at that location when the servlet sends the request, it returns the HTTP error 404 ("SC_NOT_FOUND") to the client.

The test client also opens an Error dialog box containing the text, “Adapter may be down javaio.IOException: Stream closed.”

Check to make sure that the adapter is registered with the right location in the JNDI service provider.

Incorrect JNDI Service Provider Information

If the values set for `JNDIProviderConnector` and `JNDIContextConnector` in the `XADServlet.properties` file are incorrect, the servlet returns the HTTP error 404 (“SC_NOT_FOUND”) and a “Please check the values of `JNDIProviderConnector` and `JNDIContextConnector` in the properties file” message to the requesting client.

Adding HTTPS Support

You can encrypt the data transmitted between your client application and the servlet by deploying the servlet in a web server that supports HTTPS (Secure Hyptertext Transfer Procotol). HTTPS is built on the secure sockets layer (SSL) protocol to transmit data over the web. Built upon private key encryption technology, SSL provides data encryption, server authentication, message integrity, and client authentication for any TCP/IP connection.

Refer to your iPlanet Web Server documentation for information on how to enable HTTPS support in the iPlanet Web Server. After configuring the web server for HTTPS, you can then deploy the servlet in this instance of the web server using the same process as for HTTP deployment. You access the deployed servlet using the same URL as for HTTP, replacing `http://` with `https://`

Communicating With a C Adapter Over HTTP

XAD C adapters have their own integrated HTTP server. You configure the port that the HTTP server listens on through an argument to the adapter’s executor.

The C adapter is single threaded, but you can start multiple instances listening on different ports. The C adapter’s HTTP server does not support HTTPS.

For more information on starting multiple C adapters and configuring the adapter’s HTTP listener port, see “Load Balancing” on page 97.

Chapter 6

Deploying and Testing an Adapter

This chapter explains how to compile, deploy, run and test your adapter. You test your adapter and the transport driver using the XAD test client. The test client allows you to send XML requests to the adapter over the HTTP and JMS protocols. The test client displays the adapter's response as an XML document.

This chapter also provides information on load balancing and using the XAD administration tools.

This chapter does not describe how to localize your adapter. Information on how to localize XAD adapters is included in the XAD Release Notes available at http://docs.iplanet.com/docs/manuals/xad.html.

Configuring the Adapter Engine

Prior to running a Java adapter, you must configure the adapter engine to use the RMI registry as the JNDI service provider. You do not have to perform this configuration for a C adapter.

➤ **To configure the adapter engine**

1. Open the `XADEngine.properties` file located in *XAD_ROOT*`\install\java\props\adapterengine`.

2. Locate the following line:

```
JNDIProviderConnector = rmi://localhost:1099/
```

3. Edit this line as follows:

   `JNDIProviderConnector = rmi://`*hostname*`:`*portnumber*`/`

   Replace *hostname* with the name of the machine where the RMI registry is running and *portnumber* with the port number of which the RMI registry is listening. The default port number is 1099.

Configuring the Adapter Runtime Environment

Before you run your adapter, you have to configure its runtime environment.

➤ **To configure the runtime environment for a Java adapter**

1. Open the `setadapterenv` script located in your *XAD_ROOT*`\adapters\`*adapterName*`\java` directory in a text editor.
2. Change the value of `XAD_ADAPTER_NAME` variable to the name you have chosen for your adapter, for example:

   ```
   XAD_ADAPTER_NAME=myJavaAdapter
   ```
3. Set the `XAD_JAVA_IMP_CLASSPATH` variable to the path of the target API's implementation class file.

   XAD looks for the class file first in the `bin` directory and then in the location specified by `XAD_JAVA_IMP_CLASSPATH`.
4. If you did not copy the Java class file that implements your target API into the *XAD_ROOT*`\adapters\`*adapterName*`\java\bin` directory, set the `XAD_JAVA_IMP_CLASSPATH` variable to the class file's location.

   XAD looks for the class file first in the `bin` directory and then in the location specified by `XAD_JAVA_IMP_CLASSPATH`.
5. If you are using JMS as your transport, perform the following steps:

   a. Note the following line in the `setadapterenv` script.

   ```
   XAD_JMS_PROP_FILE=%XAD_ROOT%\install\java\props\jms\%XAD_ADAP
   TER_ROOT%\%XAD_ADAPTER_NAME%.properties
   ```

   This variable specifies the location of the JMS Transport Driver's runtime properties file.

b. In your *XAD_ROOT*\install\java\props\jms directory create a new subdirectory using the name of your adapter. Copy the sample XAD_DemoAdapter.properties file from the XAD_DemoAdapter directory. Rename the file with the name of the adapter. This automatically sets XAD_JMS_PROP_FILE to the location of your adapter properties file.

c. Edit the new properties file by modifying the value of AdapterName, ActionFile, and ErrorFile values to use the name of your adapter. For example, the ActionFile value should have the format jms_*adapterName*_act.log

   For more information about the JMS Transport Driver's runtime properties file, refer to "Setting the JMS Transport Driver's Runtime Properties" in Chapter 5, "Communicating With the Adapter: HTTP and JMS."

➤ To configure the runtime environment for a C adapter

1. Open the setadapterenv script located in your *XAD_ROOT*\adapters*adapterName*\c directory.

2. Change the value of the XAD_C_ADAPTER_NAME variable to the name of your adapter, for example:

```
XAD_C_ADAPTER_NAME=myCAdapter
```

3. Change the value of the XAD_C_ADAPTER_IMP_NAME variable to the name of the object file or library file that implements your target API.

   Do not include the file extension. A correct example would be:

```
XAD_C_ADAPTER_IMP_NAME=myAPIImplementationFile
```

4. If your API implementation file is an object file, ensure that the value of the XAD_C_IMP_OBJS variable points to the object file.

   For example, if you placed your object file in the XAD Release directory:

   - On a Solaris platform, set XAD_C_IMP_OBJS to this value:

```
$XAD_C_ADAPTER_ROOT/bin/Release/$XAD_C_ADAPTER_IMP_NAME.o
```

   - Or, on a Windows platform, set XAD_C_IMP_OBJS to this value:

```
%XAD_C_ADAPTER_ROOT%/bin/Release/%XAD_C_ADAPTER_IMP_NAME%.obj
```

   If your API implementation file is not an object file, ensure that this variable is set, but that it has no value, for example:

```
XAD_C_IMP_OBJS=
```

5. If your API implementation file is a library file, ensure that the value of the `XAD_C_IMP_LIBS` variable points to the library file. Use the correct file extension for your library.

   For example, if your implementation file is a dynamic library and you placed it in the XAD `Release` directory:

   - On a Solaris platform set `XAD_C_IMP_LIBS` to this value:

     ```
     $XAD_C_ADAPTER_ROOT\bin\Release\$XAD_C_ADAPTER_IMP_NAME.so
     ```

   - On a Windows platform set `XAD_C_IMP_LIBS` to this value:

     ```
     %XAD_C_ADAPTER_ROOT%\bin\Release\%XAD_C_ADAPTER_IMP_NAME%.lib
     ```

   If your API implementation file is not a library file, ensure that this variable is set, but that it has no value, for example:

   ```
   XAD_C_IMP_LIBS=
   ```

6. Set the `XAD_C_ADAPTER_PORT` variable to the number of the port on which you want the adapter's HTTP server to listen. The default port number is `12345`.

7. Set the `XAD_C_ADAPTER_LOGFLAGS` variable to the error logging level you want.

   You can choose any level between `0` and `5`. For example, to set error logging to `4`, set the variable as follows:

   ```
   XAD_C_ADAPTER_LOGFLAGS=/EL4
   ```

   Table 6-1 describes the levels of error logging. The logging levels are cumulative. For example, level `2` includes level `1` error messages, and level `5` includes error messages from all logging levels.

Table 6-1 Error Logging Levels

Logging Level	Description
0	Logging is turned off.
1	Logs adapter start up information. Prints the initial setting for the administrative commands, for example: the package name, adapter name, adapter status (enabled or disabled), XML dialect.
2	Logs the request and response XML document.
3	Logs the request and response operation information.
4	Logs the main flow of the adapter.
5	Logs the detailed flow of the adapter. Starts from the `XADserver` to the wrapper and back.

Running the Adapter

You are now ready to run the adapter. This section describes how to run a Java adapter and a C adapter.

➤ To run a Java adapter

NOTE Execute all of the scripts and commands in this procedure from the same command line.

1. From a command line, navigate to *XAD_ROOT*`/adapters/`*adapterName*`/java` and run the `setadapterenv` script (where *adapterName* is the name of your adapter).

   You must have already copied and modified this script. For information on copying the script, see "To set up a directory structure for a Java adapter" on page 28. For information on modifying the script, see "To configure the runtime environment for a Java adapter" on page 82.

2. Compile the adapter by executing the `xad_mk_java_adapt` script.

   The executable files are written to the *XAD_ROOT*`/adapters/`*adapterName*`/java/bin` directory.

3. If the RMI registry is not already running on the machine with the adapter, start it. Use one of the following commands:

   On a Solaris platform:

   ```
   rmiregistry&
   ```

   On a Windows platform:

   ```
   start rmiregistry
   ```

   If you are using an LDAP server as your JNDI provider, start the server for that system.

4. Run the adapter by executing the `xad_run_java_adapt` script.

   You are now ready to set up your transport driver so that your adapter can communicate with your client application. For information on this topic, see Chapter 5, "Communicating With the Adapter: HTTP and JMS."

➤ To run a C adapter

NOTE Execute all of the scripts and commands in this procedure from the same command line.

1. From a command line, navigate to *XAD_ROOT*`/adapters/`*adapterName*`/c` and run the `setadapterenv` script (where *adapterName* is the name of your adapter).

   You must have already copied and modified this script. For information on copying the script, see "To set up a directory structure for a C adapter" on page 29. For information on modifying the script, see "To configure the runtime environment for a C adapter" on page 83.

2. Compile the adapter by executing the `xad_mk_c_adapt` script.

   The executable file is written to the *XAD_ROOT*`/adapters/`*adapterName*`/c/bin` directory.

3. Run the adapter by executing the `xad_run_c_adapt` script.

NOTE If you want to start multiple instances of your adapter for load balancing purposes, start the instances from a command line instead running this script. See "Load Balancing" on page 97 for more information.

You are now ready to test your adapter with the test client.

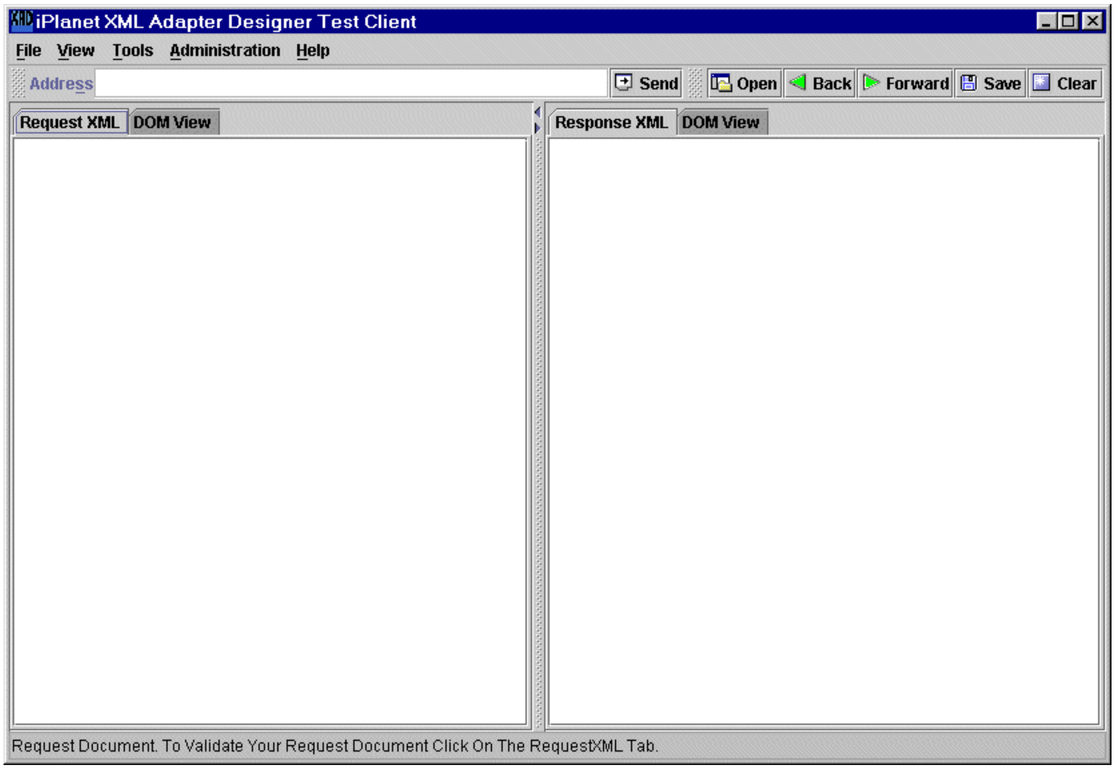
Running the XAD Adapter Test Client

The XAD adapter test client allows you to test adapters by sending request messages to the adapter through HTTP or JMS and displaying messages from the adapter.

➤ **To run the client**

1. Run the `xad_run_testclient` script in the *XAD_ROOT*`\install\bin` directory.

   The client window opens.



Request Document. To Validate Your Request Document Click On The RequestXML Tab.

Working with XML Documents

You can import, view and edit sample XML request documents using the XAD test client.

➤ To open an XML request document

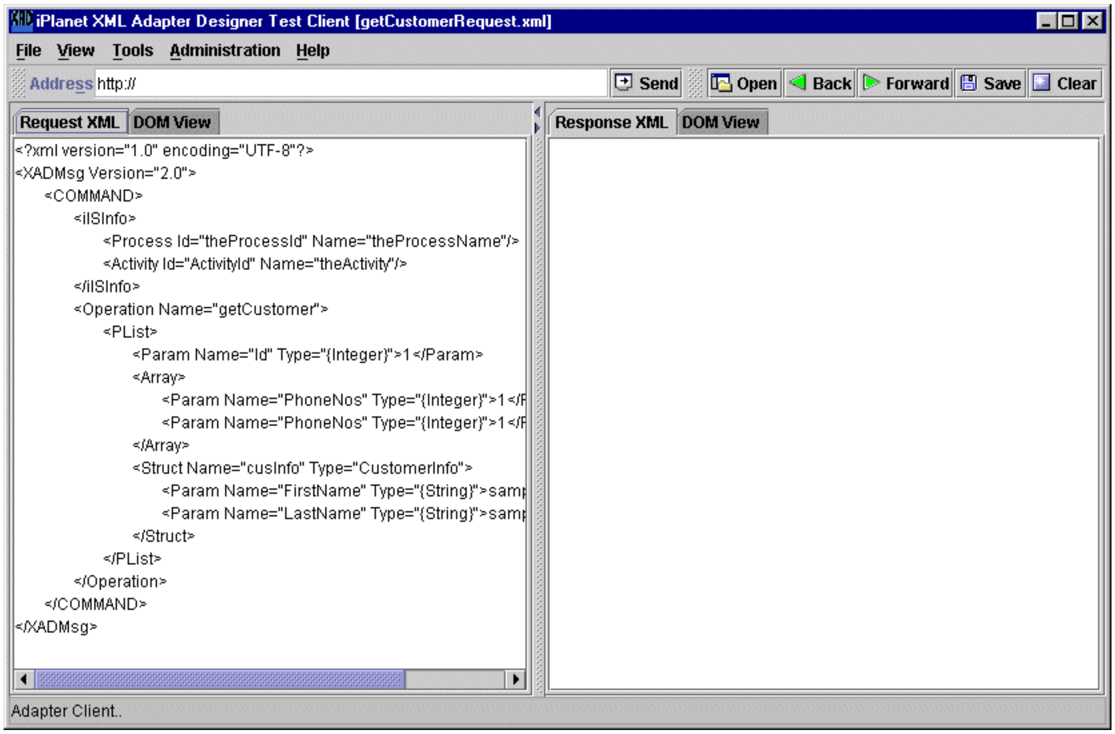
1. Choose File > Open and navigate to the directory containing a request document and select a file, such as `getCustomerRequest.xml`.

2. Click Open.

   The document opens on the Request XML tab. You can view the document as a tree by clicking the DOM View tab.

➤ To edit an XML request document

1. Select the XML Request tab and make any changes to the document in the text area.


iPlanet XML Adapter Designer Test Client [getCustomerRequest.xml]
File View Tools Administration Help
Address http://
Send
Open
Back
Forward
Save
Clear
Request XML
DOM View
Response XML
DOM View
Adapter Client..

```
<?xml version="1.0" encoding="UTF-8"?>
<XADMsg Version="2.0">
    <COMMAND>
        <ilSInfo>
            <Process Id="theProcessId" Name="theProcessName"/>
            <Activity Id="ActivityId" Name="theActivity"/>
        </ilSInfo>
        <Operation Name="getCustomer">
            <PList>
                <Param Name="Id" Type="{Integer}">1</Param>
                <Array>
                    <Param Name="PhoneNos" Type="{Integer}">1</P
                    <Param Name="PhoneNos" Type="{Integer}">1</P
                </Array>
                <Struct Name="cusInfo" Type="CustomerInfo">
                    <Param Name="FirstName" Type="{String}">samp
                    <Param Name="LastName" Type="{String}">samp
                </Struct>
            </PList>
        </Operation>
    </COMMAND>
</XADMsg>
```

2. Select File > Save Request and a browser prompts you to save the file.

3. Click Save.

You can page through the history of sent requests by clicking on the Back and Forward buttons.

Sending Requests to an Adapter

You can send requests from the test client to a generated adapter over HTTP or JMS. This section explains how to:

- Set the router type

  This setting controls whether the test client sends requests over HTTP or JMS.

- Configure the HTTP connection settings
- Configure the JMS connection settings

The instructions in this section assume that you have already set up the listener servlet or JMS transport listener as described in Chapter 5, "Communicating With the Adapter: HTTP and JMS."

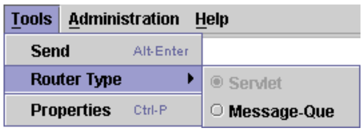
➤ To set the router type for the test client

1. Select Tools > Router Type.

   A submenu opens containing two radio-button options:

   ❍ Servlet

   ❍ Message-Que



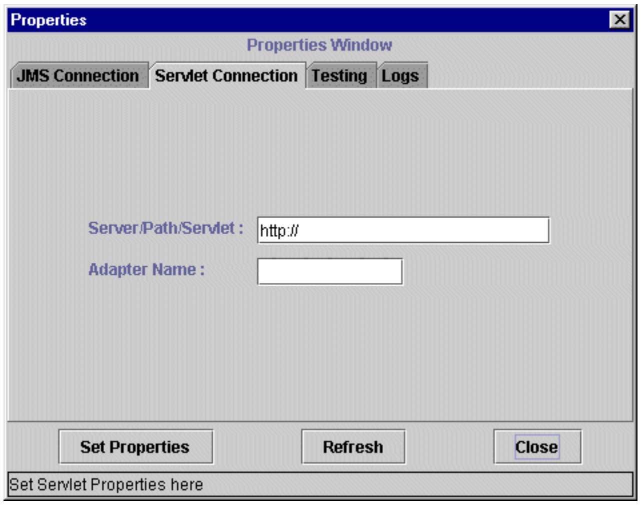
2. Choose Servlet if you want to send requests over HTTP; choose Message-Que if you want to send requests over JMS.

➤ To send a request over HTTP

1. Ensure that Router Type is set to Servlet.
2. If the Properties Window is not already open, open it by choosing Tools > Properties.

   Click the Servlet Connection tab if it is not already displayed.

   If the Set Properties button is disabled, the Router Type is not set to the correct value.



Server/Path/Servlet : http://

Adapter Name :

Set Servlet Properties here

3. In the Server/Path/Servlet field, type the URL of the listener servlet.

   The URL contains the web server DNS name, the virtual path to the servlet, and the servlet name, for example:

   ```
   http://rex.varrius.com/adapter/XADServlet
   ```

   The virtual path and servlet name are specified in the web server. See "Installing the Listener Servlet" on page 73 for more information on this topic.
4. In the Adapter Name field, type the name of your generated adapter.

   This field is case sensitive.
5. Click Set Properties.

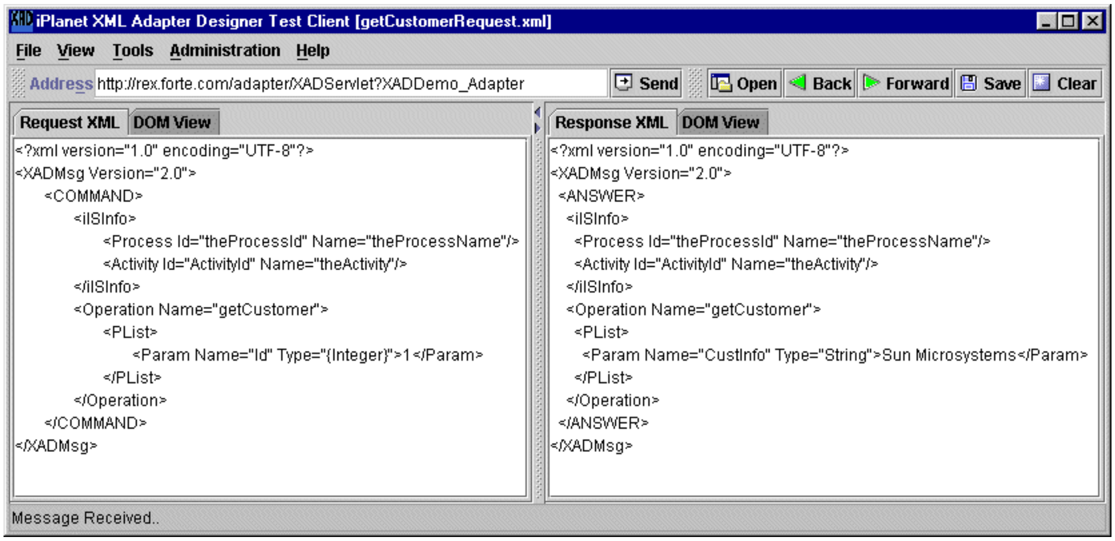
   The complete address appears in the address field.

6. Open a request document (see "To open an XML request document" on page 88).

7. Select Action > Send or click Send.

   The Response window displays either the adapter's response or an error message if the URL or message content contains an error.

   You can save the response as an XML file by choosing File > Save Response.



Address: http://rex.forte.com/adapter/XADServlet?XADDemo_Adapter

```
<?xml version="1.0" encoding="UTF-8"?>
<XADMsg Version="2.0">
    <COMMAND>
        <iISInfo>
            <Process Id="theProcessId" Name="theProcessName"/>
            <Activity Id="ActivityId" Name="theActivity"/>
        </iISInfo>
        <Operation Name="getCustomer">
            <PList>
                <Param Name="Id" Type="{Integer}">1</Param>
            </PList>
        </Operation>
    </COMMAND>
</XADMsg>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<XADMsg Version="2.0">
 <ANSWER>
  <iISInfo>
   <Process Id="theProcessId" Name="theProcessName"/>
   <Activity Id="ActivityId" Name="theActivity"/>
  </iISInfo>
  <Operation Name="getCustomer">
   <PList>
    <Param Name="CustInfo" Type="String">Sun Microsystems</Param>
   </PList>
  </Operation>
 </ANSWER>
</XADMsg>
```

Message Received..

➤ To send a request over JMS

1. Ensure that Router Type is set to Message-Queue.

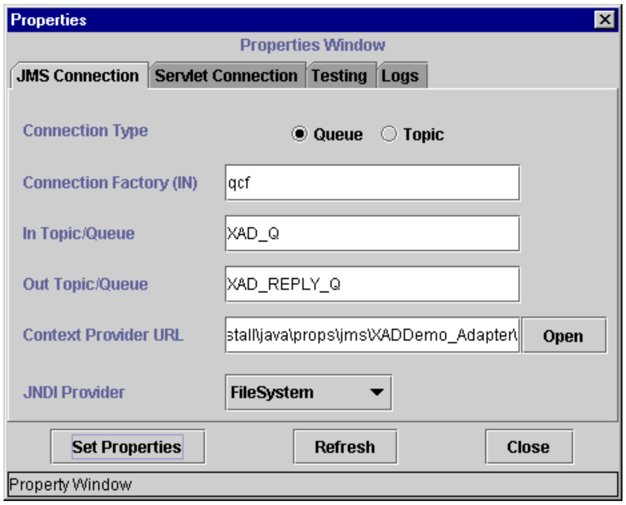
2. If the Properties Window is not already open, open it by choosing Tools > Properties.

   Click the JMS Connection tab if it is not already displayed.

   If the Set Properties button is disabled, the Router Type is not set to the correct value.

3. Enter values for the fields on the tab from the properties set in the JMS Transport Driver's properties file. If they are not exactly the same, the test client will not be able to send the message. All the fields are required.

   If you select FileSystem as your JNDI Provider click on Open to navigate to the `.bindings` file generated during the registration process. It should be in the same directory as the `.properties` file.



Connection Type: (● Queue) ○ Topic
Connection Factory (IN): qcf
In Topic/Queue: XAD_Q
Out Topic/Queue: XAD_REPLY_Q
Context Provider URL: stall\java\props\jms\XADDemo_Adapter\
JNDI Provider: FileSystem

4. When you have finished entering the information, click Set Properties. The Address field displays `JMS://Messaging`.

5. Send a request to the adapter by opening an XML request document and clicking Send. The XML response will be displayed in the Response XML window.

Adapter Administration Tools

The XAD Adapter Client contains a number of administrative commands you can use to manage and test the adapter. The client sends them to the administrative engine component of a generated adapter. The administrative engine executes the request and returns the response to the adapter engine.

The supported commands are listed in Table 6-2.

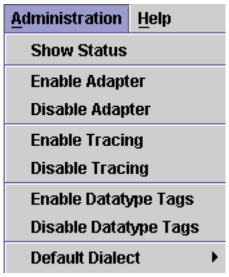
Table 6-2 Administrative Commands

Command	Function
Enable Tracing	Issues an XML command to the adapter administrative engine, instructing it to turn on the internal tracing. For a C adapter, this command sets tracing (logging) to the level specified by the `XAD_C_ADAPTER_LOGFLAGS` variable. This variable is set by the `setadapterenv` script. Table 6-1 on page 84 describes the logging levels to which you can set this variable. The default level is `0`, which is equivalent to turning tracing off. For Java adapters, tracing is either on or off.
Disable Tracing	Issues an XML command to the adapter administrative engine, instructing it to turn off the internal tracing.
Show Status	Issues an XML command to the adapter administrative engine to retrieve and display the status of all the adapter instruments.
Enable/Disable the Adapter	Issues an XML Trace message to the adapter administrative engine instructing it to turn the adapter on or off. An adapter that has been turned off is still an active process capable of processing system messages and trace information but unable to execute XML requests. Can be used when the system is undergoing maintenance.
Enable/Disable Datatype Tags	Issues an XML command to the adapter administrative engine instructing it not to return the data type of parameters in the XML response message. By default, the output of data types is enabled.
Default Dialect	Specifies the default XML dialect of requests and responses.

➤ **To send an administrative command to the adapter**

1. Make sure the adapter is running and the transport information (servlet or JMS Transport Driver) is set up correctly.

2. Select Administration from the menu. The commands in the table are displayed as menu items.



   Select the desired command.

3. The XAD test client sends the message to the adapter administrative engine which returns an XML response. The contents of this response are displayed in a separate Adapter Status dialog box.

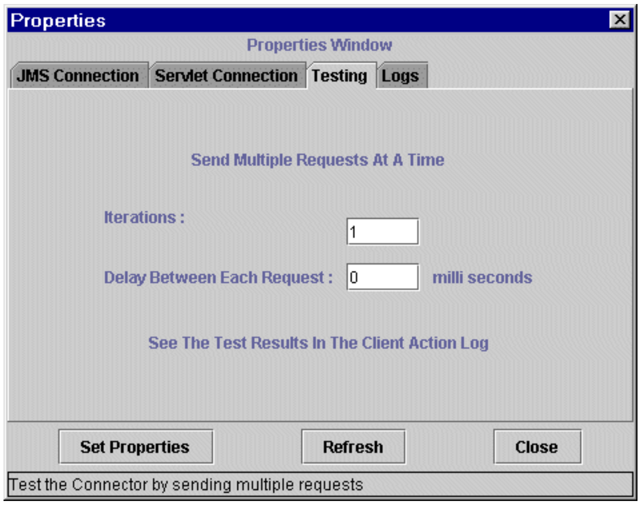
Stress Testing an Adapter

You can use the test client to stress test your adapter by automating the sending of a sample request. You can configure the:

- Total number of times the test client sends the request
- The length of time between requests

➤ To stress test the adapter

1. Choose Tools > Properties and the click the Testing tab.



Send Multiple Requests At A Time

Iterations : 1

Delay Between Each Request : 0 milli seconds

See The Test Results In The Client Action Log

Test the Connector by sending multiple requests

2. In the Iteration field, type the number of times you want your request sent to the adapter.
3. In the Delay Between Each Request field, type the time in milliseconds that you want the test client to wait before repeating the request.
4. Click Set Properties.
5. If Action Logging for the test client is disabled, a dialog box opens containing the text, "Action Logging has been switched off. Enable the Action Logging to view the test results."
   a. Click OK.

      The Logs tab is displayed.
   b. Enable the Action Logs field.
6. Open a request document and click on Send.
7. The information regarding the number of requests sent, the time required for each request, and the average time of all requests is written to the action file specified on the Logs tab.

Setting Logging in the XAD Test Client

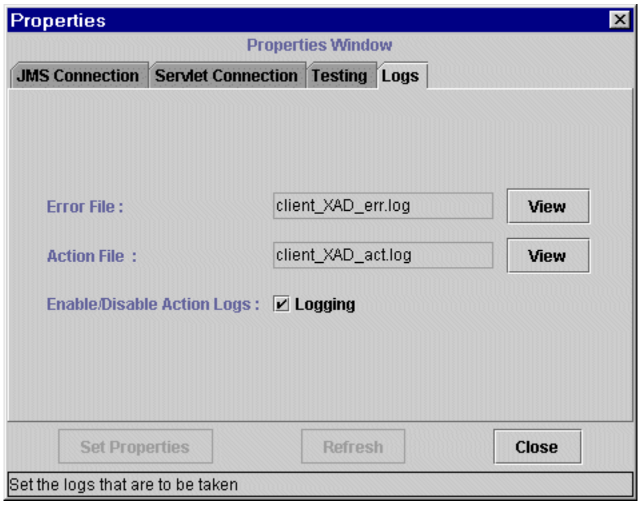
You can view the XAD test client's log files and enable or disable logging.

➤ **To view the XAD test client's log files**

1. Select Tools > Properties and click on the Logs tab. The tab displays the names of the test client's action and error log files. Click View to display the contents of either log file.

2. Select or deselect the Logging checkbox to enable or disable logging.

   By default, action logging is disabled. Error logging is always enabled.



Error File : client_XAD_err.log

Action File : client_XAD_act.log

Enable/Disable Action Logs : ✓ Logging

Set the logs that are to be taken

Load Balancing

This section explains how to provide load balancing between your adapter and an iIS client. It also provides general information on load balancing that is relevant to other types of clients.

XAD adapters are capable of maintaining multiple simultaneous client sessions. How you configure your application to achieve this functionality depends on the kind of adapter you build—a Java adapter or C adapter—and on whether you use HTTP or JMS for transport.

Java Adapters

XAD Java adapters and the XAD servlet listener are both multi threaded. If you are using a Java adapter and communicating over HTTP, you do not have to do any configuration for load balancing on the adapter side. The only configuration you have to perform is for the iIS proxy (see “Configuring the iIS Proxy” on page 101).

The XAD JMS listener is single threaded. If you want to load balance a Java adapter and you are using JMS, you must start multiple instances of the JMS listener (see “Starting Multiple Instances of the JMS Listener” on page 99). Each JMS listener instance communicates with the same Java adapter instance (which is multi threaded).

You then enable load balancing on either the iIS proxy or a JMS queue. As shown in Figure 6-1, if you enable load balancing on the iIS proxy, you must configure *n* destinations in the proxy and *n* queues in the JMS broker.

If you enable load balancing on a JMS queue, you configure a single destination in the iIS proxy and a single queue in the broker. This model is shown in Figure 6-2.

For information on load balancing the iIS proxy, see “Configuring the iIS Proxy” on page 101. For information on load balancing a JMS queue, consult the documentation for your JMS product.

Figure 6-1 JMS Load Balancing Using Round Robin on the iIS Proxy

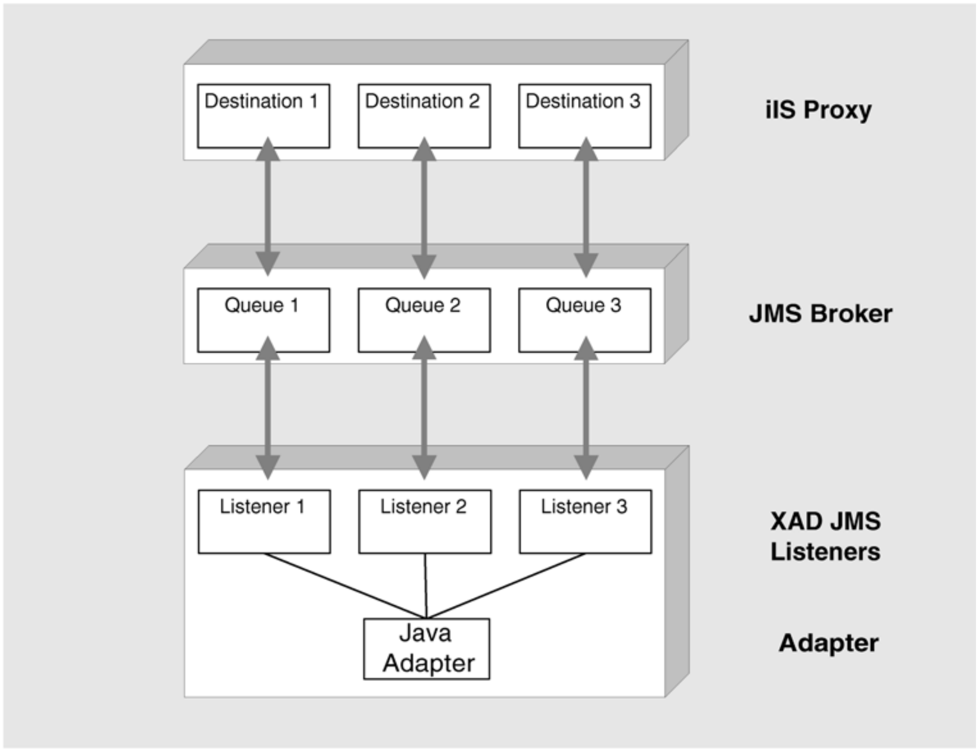
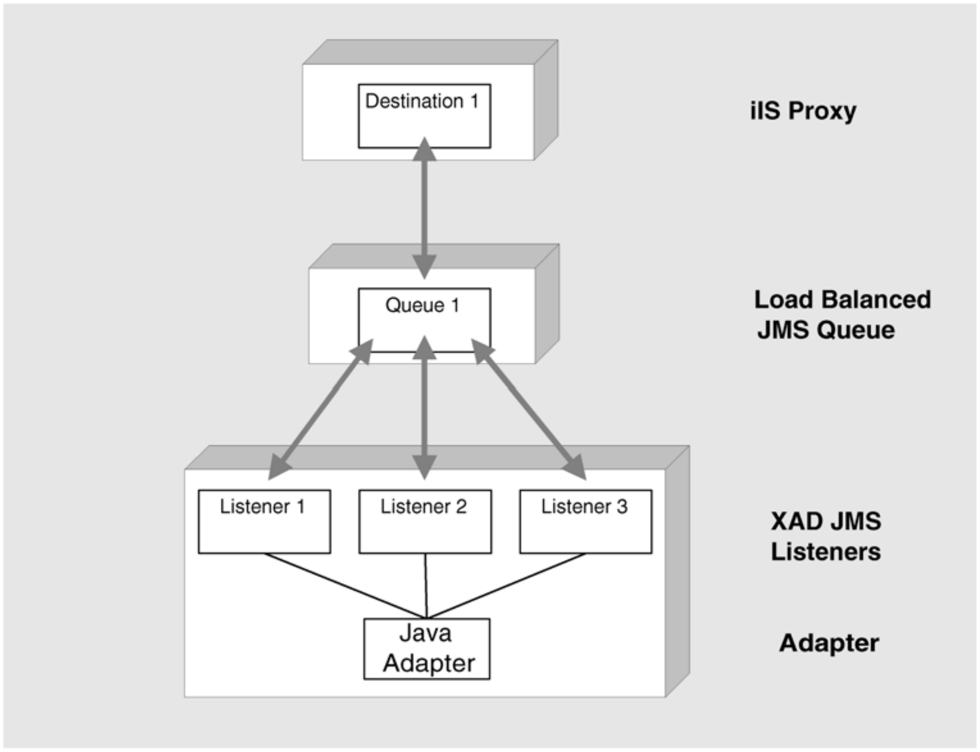


Figure 6-2 JMS Load Balancing Using Round Robin on a JMS Queue



Starting Multiple Instances of the JMS Listener

The procedure you follow to start multiple instances of the JMS listener depends on whether you load balance the iIS proxy or a JMS queue.

➤ **To start multiple instances of the JMS listener for a load balanced JMS queue**

1. Ensure that you have a JMS queue configured for round robin load balancing.
2. Run the `xad_run_jms` script once for each listener instance you want to start.

➤ **To start multiple instances of the JMS listener for a load balanced iIS proxy**

1. Create a separate properties file for each listener instance you want to start by copying the `XADSamples_Java.properties` file located in the *XAD_ROOT*`/install/java/props/jms` directory.

   Give each file a unique name. You can place them in the same directory if you want. For example:

   - `listener1.properties`
   - `listener2.properties`
   - `listener3.properties`

2. Edit each of the properties files by giving the `Queue` and `ReplyQueue` properties unique values, for example:

   -
     ```
     Queue = XAD_Q1
     ReplyQueue = XAD_REPLY_Q1
     ```

   -
     ```
     Queue = XAD_Q2
     ReplyQueue = XAD_REPLY_Q2
     ```

   -
     ```
     Queue = XAD_Q3
     ReplyQueue = XAD_REPLY_Q3
     ```

3. Open the `xad_run_jms` script located in the *XAD_ROOT*`/install/bin directory` and locate the line that executes the `com.iplanet.xad.jms.XADJMSRouter` class.

   On Solaris platforms, the line looks like this:

   ```
   ${JAVA_HOME}/bin/java -DXAD_ROOT=${XAD_ROOT} -Xmx64m
   com.iplanet.xad.jms.XADJMSRouter -properties
   ${XAD_JMS_PROP_FILE}
   ```

   On Windows platforms, the line looks like this:

   ```
   %JAVA_HOME%\bin\java -DXAD_ROOT=%XAD_ROOT% -Xmx64m
   com.iplanet.xad.jms.XADJMSRouter -properties %XAD_JMS_PROP_FILE%
   ```

4. Using this line as a model, execute this class once for each properties file you created in Step 1.

   Substitute the appropriate values for the `JAVA_HOME` and `XAD_ROOT` variables. The `XAD_JMS_PROP_FILE` variable specifies the path to the properties file. Each time you execute the class, change the value of the `properties` argument so that you point to each of the properties files in turn.

C Adapter

XAD C adapters are single threaded. To provide load balancing, you must start multiple instances of your adapter. Each C adapter instance has its own integrated and dedicated HTTP server. You must configure the HTTP server for each adapter instance to listen on a different port. You should also direct logging messages for each adapter instance to a separate log file.

Code Example 6-1 shows an example of how to start an instance of a C adapter from a Solaris command line. In this example, the adapter is named `myAdapter`. The command sets the port number to `8001`, sets the logging level to `EL4`, and names the log file `adapterInst1.log`. *XAD_ROOT* represents the XAD installation directory.

Code Example 6-1 Starting a C adapter from a Solaris Command Line

```
XAD_ROOT/bin/myAdapter /P:8001 /EL4 > XAD_ROOT/log/adapterInst1.log
```

Code Example 6-2 shows the same example executed from a Windows command line.

Code Example 6-2 Starting a C adapter from a Windows Command Line

```
"XAD_ROOT\bin\myAdapter.exe" /P:8001 /EL4 > "XAD_ROOT\log\adapterInst1.log"
```

Configuring the iIS Proxy

This section explains how to configure your iIS proxy for load balancing under the following scenarios:

- Java adapter over HTTP
- C adapter over HTTP

For further information on load balancing XAD adapters, including how to configure an iIS proxy for load balancing a Java adapter over JMS, refer to iPlanet Knowledge Base article 7781 located at:

`http://knowledgebase.iplanet.com/ikb/kb/articles/7781.html`

For general information on working with iIS proxies, refer to the *iIS Backbone System Guide*.

Java Adapter Over HTTP

If your iIS proxy communicates with a Java adapter over HTTP, your only load balancing issue is to configure the number of outbound sessions the iIS proxy runs. Use the FNscript command `SetSessionMaximum` to set the maximum number of sessions for your proxy. For example, the following command sets the maximum number of outbound sessions to 5:

```
SetSessionMaximum out 5
```

Because both the Java adapter and servlet listener are multi threaded, you need to add only one destination to the proxy (the destination points to the servlet listener). The FNscript command for adding a destination is `AddAplURL`, for example:

```
AddAplURL http://rex.varrius.com:8000/adapter/XADServlet?adaptername
```

C Adapter Over HTTP

For a C adapter, the procedure is more complicated than for a Java adapter. Because each C adapter instance listens on a different port, you have to add multiple destinations to your iIS proxy. For example, if you start three C adapter instances listening on ports `8000`, `8001`, and `8002` on the machine `rex.varrius.com`, you would add three destinations as follows:

```
AddAplURL http://rex.varrius.com:8000

AddAplURL http://rex.varrius.com:8001

AddAplURL http://rex.varrius.com:8002
```

NOTE You can use the command `ClearAplURL` before the `AddAplURL` command to clear all previously added destinations.

The `xadbackbone_inst_http.scr` script, located in the *XAD_ROOT*`/install/forte/scripts` directory, is provided as part of the XAD iIS client example. This script contains commented out lines that are useful for various load balancing scenarios. You can copy and edit this script and use it to configure your iIS proxy.

Chapter 7

Working With Data Types

This chapter explains important information about working with data types in XAD. The situations in which you work with data types are:

- Defining a target API in the Interface Editor.

  As part of defining your target API, you specify the data types of the parameters passed between your adapter and the API (as explained in "Creating Parameters" on page 33).

- Creating XSLT rules for an iIS client.

  After generating XSLT rules (as explained in "Generating XSLT Rules" on page 47) you must edit the rules to ensure that the data types of your iIS activity attributes map correctly to the operation parameters of your adapter.

Specifying Data Types in the API Definition

As you define your target API, you add operations and parameters to the definition. XAD allows you to specify the data type of each parameter from one of two categories of data types.

- Language-specific data types
- Abstract data types

If you specify your parameters as language-specific data types, you can use your API definition to generate an adapter only for that specific language. For example, if you specify your parameters as Java scalars and classes, you can use your API definition to generate a Java adapter but not a C adapter.

XAD provides a more general way to specify data types through the use of abstract data types. If you specify your parameters as abstract data types, you can use the same API definition to generate a Java adapter or a C adapter. Using abstract data types also reduces the amount of editing required in generated XSLT rules (see "Mapping iIS Data Types to XAD Data Types" on page 105).

Abstract data types are always enclosed in curly braces, for example `{Integer}` or `{String}`.

Conversion of Abstract Data Types

When you generate an adapter, all the abstract data types specified in your API definition are converted to data types that are specific to the language in which you generate the adapter.

Table 7-1 lists the abstract data types supported by XAD and the corresponding language-specific data types to which they are converted.

Table 7-1 Abstract Data Types and Their Corresponding Language-Specific Data Types

Abstract Data Type	Java	C
`{Integer}`	`int`	`int`
`{String}`	`java.lang.String`	`char*`
`{Sized String}`	`java.lang.String`	`char [`*dimension*`]`
`{Char}`	`char`	`char`
`{Boolean}`	`boolean`	`int`
		The C language does not provide a boolean type. The {Boolean} abstract data type accepts the value `true` or `false` (case insensitive) and translates it to `1` or `0`. (Actually, anything other than `true` translates to `0`).
`{Double}`	`double`	`double`
`{Float}`	`float`	`float`
`{Date Time}`	`java.util.Date`	`struct tm`
`{Long}`	`long`	`long int`
`{Short}`	`short`	`short int`

Table 7-1 Abstract Data Types and Their Corresponding Language-Specific Data Types

Abstract Data Type	Java	C
`{Unsigned Int}`	`int`	`unsigned int`
`{Unsigned Long Int}`	`int`	`unsigned long int`
`{Unsigned Short Int}`	`short`	`unsigned short int`
`{Unsigned Char}`	`char`	`unsigned char`

Mapping iIS Data Types to XAD Data Types

An iIS client uses outbound XSLT rules to transform attribute data types from types understood by iIS to types understood by your adapter. Inbound XSLT rules perform the reverse function—transforming adapter parameter data types into iIS attribute data types. As explained in "Generating XSLT Rules" on page 47, you can generate outbound and inbound XSLT rules from your target API definition.

At the time of generation, the XAD Generator knows the parameter data types used in the adapter because you specified these when you created your API definition. However, the XAD Generator has no knowledge of the data types used in the iIS client, so it uses a default mapping scheme to generate the rules. It maps the parameter data types specified in the API definition to default iIS data types.

Table 7-2 describes the default scheme used to map some of the XAD abstract data types to iIS data types.

,

Table 7-2 Default Scheme Used to Map Some Abstract Data Types to iIS Data Types

Abstract Data Type	iIS Data Type
{Boolean}	BooleanData
{DateTime}	DateTimeData
{Float}	DecimalData
{Double}	DoubleData
{Integer}	IntegerData
{String}	TextData

Note: There is no default mapping to the iIS data type `IntervalData`.

All other data types are mapped to themselves. For example, `int` is mapped to `int`; `{Short}` is mapped to `{Short}`; and a user-defined class named, for example, `myClass` is mapped to `myClass`.

Arrays are mapped according to the data type they contain. For example, an array of `{String}` data types is mapped to a `TextData` data type (following the mapping scheme described in Table 7-2). Similarly, an array of `int` data types is mapped to an `int` data type.

In some cases, you need to edit the iIS data types in your generated XSLT rules. The following section describes when and how to make these edits.

NOTE XAD adapters use the date-time format `dd-mmm-yyyy hh:mm:ss`. If the iIS proxy is configured to use a different date-time format, you must add a custom XSLT rule to the generated inbound and outbound rules to transform between the two date-time formats.

Editing Outbound Rules

The outbound XSLT rule prepares the iIS client's XML request to be sent to the adapter. You need to edit the iIS data types in the generated outbound rule only for user-defined data types (Java classes and C structures) and arrays.

Mismatches between all other data types are ignored because the outbound rule matches the iIS attribute to the adapter parameter by name only. At runtime, the adapter is already generated and knows the data type of the parameter.

The following code examples illustrate how to edit outbound rules generated for user-defined data types and arrays. Code Example 7-1 shows an outbound rule generated for a parameter named `cusInfo` of type `CustomerInfo` that is mapped to an iIS process attribute named `c`.

Code Example 7-1 Outbound Rule Before Editing

```
<xsl:template
match="FNProcessAttributeList/FNProcessAttribute[@Name='c']">
  <Param>
    <xsl:attribute name="Name">cusInfo</xsl:attribute>
    <xsl:attribute name="Type">CustomerInfo</xsl:attribute>
    <xsl:value-of select="."/>
  </Param>
</xsl:template>
```

Code Example 7-2 shows the outbound rule after editing.

Code Example 7-2 Outbound Rule After Editing

```
<xsl:template
match="FNProcessAttributeList/FNProcessAttribute[@Name='c']">
  <xsl:copy-of select="*"/>
</xsl:template>
```

Editing Inbound Rules

The inbound XSLT rule maps the attribute data types in the XML response to the attribute data types in the iIS process definition. If your API definition specifies data types other than those listed in Table 7-2, you must edit your generated inbound XSLT rule to ensure that the attribute data types map correctly.

For example, Code Example 7-3 shows a portion of code from a generated inbound XSLT rule. This code passes the adapter's return parameter `uiReturn` as an iIS attribute named `uiPReturn`. In the API definition, `uiReturn` is specified as an `{Unsigned Int}` data type. Therefore, according to the default mapping scheme, the data type in the generated rule is also `{Unsigned Int}`.

Code Example 7-3 Inbound Rule for an {Unsigned Int} Data Type Before Editing

```
<xsl:when test="@Name='uiReturn'">
  <FNProcessAttribute>
    <xsl:attribute name="Name">uiPReturn</xsl:attribute>
    <xsl:attribute name="Type">{Unsigned Int}</xsl:attribute>
    <xsl:value-of select="."/>
  </FNProcessAttribute>
</xsl:when>
```

Let us assume for this example that the `uiPReturn` attribute is specified in the iIS process definition as an `IntegerData` data type. The code must therefore be modified as shown in Code Example 7-4 (the modification is shown in bold).

Code Example 7-4 Inbound Rule for an {Unsigned Int} Data Type After Editing

```
<xsl:when test="@Name='uiReturn'">
  <FNProcessAttribute>
    <xsl:attribute name="Name">uiPReturn</xsl:attribute>
    <xsl:attribute name="Type">IntegerData</xsl:attribute>
    <xsl:value-of select="."/>
  </FNProcessAttribute>
</xsl:when>
```

In iIs, user-defined types and arrays are stored as an `xmlData` data type. The entire XML body that represents the parameter value is stored in the process attribute before and after the activity. As an example for a user-defined data type, the process attribute `c` (from Code Example 7-1) might have a value similar to that shown in Code Example 7-5.

Code Example 7-5 A User-Defined Data Type

```
<Struct Name="cusInfo" Type="CustomerInfo">
  <Param Name="FirstName" Type="{String}">sample string</Param>
  <Param Name="LastName" Type="{String}">sample string</Param>
</Struct>
```

A process attribute representing an array might have a value similar to that shown in Code Example 7-6.

Code Example 7-6 Inbound Rule for an Array Data Type Before Editing

```
<Array Name="sInput">
  <Param Name="sInput" Type="{String}">sample string</Param>
  <Param Name="sInput" Type="{String}">sample string</Param>
  <Param Name="sInput" Type="{String}">sample string</Param>
</Array>
```

Suppose you wanted to store a structure parameter named `cusInfoReturn` in the `c` process attribute. The generated inbound XSLT rule contains lines shown in Code Example 7-7 within the `<xsl:choose>` element of the `<xsl:template match="Param">` element.

Code Example 7-7 Inbound Rule for User-Defined Data Type Before Editing

```
<xsl:when test="@Name='cusInfoReturn'">
  <FNProcessAttribute>
    <xsl:attribute name="Name">c</xsl:attribute>
    <xsl:attribute name="Type">CustomerInfo</xsl:attribute>
    <xsl:value-of select="."/>
  </FNProcessAttribute>
</xsl:when>
```

You would edit this rule by deleting the entire text shown in Code Example 7-7 and adding the text shown in Code Example 7-8.

Code Example 7-8 Inbound Rule for User-Defined Data Type After Editing

```
<xsl:template match="Struct">
  <xsl:choose>
    <xsl:when test="@Name='cusInfoReturn'">
      <FNProcessAttribute>
        <xsl:attribute name="Name">c</xsl:attribute>
        <xsl:attribute name="Type">xmlData</xsl:attribute>
        <xsl:copy-of select="."/>
      </FNProcessAttribute>
    </xsl:when>
  </xsl:choose>
</xsl:template>
```

In the case of an array, you would add the text shown in Code Example 7-9 instead.

Code Example 7-9 Inbound Rule for an Array Data Type After Editing

```
<xsl:template match="Array">
  <xsl:choose>
    <xsl:when test="@Name='sReturn'">
      <FNProcessAttribute>
        <xsl:attribute name="Name">strArray</xsl:attribute>
        <xsl:attribute name="Type">xmlData</xsl:attribute>
        <xsl:copy-of select="."/>
      </FNProcessAttribute>
    </xsl:when>
  </xsl:choose>
</xsl:template>
```

Array and Pointer Data Types in C Adapters

The XAD distribution contains a C adapter example in *XAD_ROOT*`/adapters/XADSamples_C`. This example includes the API interface definition and API implementation. The API interface definition is located at `def/XADSamples_C.xml` relative to the sample directory. The API implementation is in `c/src/sample.h` and `sample.c`. Examine these files for details on how to use arrays in C adapters.

Using Arrays

There are two approaches for using arrays in XAD:

- Wrap the array in a `struct` in the API implementation.
- Create a `typedef` that points to the array.

Wrapping the Array in a struct in the API Implementation

The following is an example of how to use an array of 2 integers.

```
typedef struct arrayint {
  int i[2];
} ArrayInt;
```

From this point you can use `ArrayInt`.

Input and Return Mechanism

Implementation: `ArrayInt * testIntArray_in (ArrayInt * integer_in);`

In the Interface Editor: Define a group parameter each for input and return both of type `ArrayInt *`

Output Mechanism

Implementation: `void testIntArray_out (ArrayInt ** integer_out);`

In the Interface Editor: Define a group parameter for output of type `ArrayInt *`

InOut Mechanism

Implementation: `void testIntArray_inout (ArrayInt ** integer_inout);`

In the Interface Editor: Define a Group parameter for inout of type `ArrayInt *`

Creating a typedef that Points to the Array

This example shows how to use an array of 5 integers by creating a `typedef` that points to the array. This example is taken from the example C adapter in *XAD_ROOT*`/adapters/XADSamples_C`.

```
typedef int myArray[5];
typedef myArray *myArrayPtr;
```

From this point you can use `myArrayPtr` in the API implementation.

Return Mechanism

Implementation: `myArrayPtr FunctionNoPList();`

Interface Definition: Create a collection parameter for return of type `int *` and size `5`.

Using Pointers

This section provides tips on using pointers in a C adapter.

Output and InOut Types

In the case of output and inout types, if you are using a pointer to a pointer in the API implementation, define them as pointers only in the interface.

Returning a Pointer

In return mechanism, when returning a pointer from the API implementation, you must ensure that memory is allocated to the pointer in the implementation. If the implementation tries to return the same pointer as its input pointer, the adapter will fail because it tries to free the memory twice—once for the input pointer and again for the return pointer, both of which point to the same location. For example, the following function will fail under these circumstances:

```
int * intInputAndReturn(int *p){
  return p;
}
```

The following example shows the correct way to code such a function:

```
int * intInputAndReturn(int *p){
int * j = (int*) malloc(sizeof(int));
*j = *p;
  return j;
}
```

Appendix A

Running the iIS Client Example

The XAD software distribution provides an iIS example application that you can optionally install as part of your XAD installation. This document explains how to set up and run this example application as a client to an example adapter provided in the XAD installation.

To perform the procedures described in this document, you should be familiar with the basic concepts and tools of iIS and UDS.

Overview

The example described in this appendix demonstrates the runtime operation of an iIS client application communicating with an XAD generated adapter. The XAD distribution provides both a Java adapter and a C adapter for use in the example. You can configure the communication between the client and adapter to use one of the following combinations of message format and transport mechanism:

- XAD dialect of XML over HTTP
- SOAP over HTTP (*Java adapter only*)
- XAD dialect of XML over JMS (*Java adapter only*)

The JNDI provider for JMS can be either an LDAP server or the filesystem.

The iIS Client

The iIS client application implements a simple customer accounting process consisting of three activities. In the first activity, a customer name is retrieved by providing a customer ID. The second activity transfers money from one account to another. The third activity summarizes the customer’s transaction.

For the purpose of this example, the iIS process definition (the process development plan `XADDemo_PD`) has been kept very simple. The information supplied to the process definition includes a user validation, user profile, assignment rules, and the application dictionary, as indicated in the following table.

Table 7-3 Process Definition for the iIS Example Application

Process Development Plan	**Description**
User Validation (`XADDemo_UV`)	The iIS process engine checks the user validation to determine that the user has access. In this example, all users are granted access.
User Profile (`XADDemo_UP`)	The user profile is the default plan, with no additional changes.
Assignment Rules (`XADDemo_AR`)	The iIS process engine uses assignment rules to determine the sessions to which activities are offered. This example contains two roles, with associated assignment rules, to assign the `CustomerInfo` and `TransferFunds` activities.
Application Dictionary (`XADDemo_AD`)	Application dictionaries describe the properties of applications, and are not registered with the iIS process engine. The example process definition requires this application dictionary to define properties for the process activities.
Process Definition (`XADDemo_PD`)	Contains the iIS activities.

The Example XML Adapters

The application adapters provided with your distribution are located at *XAD_ROOT*`/adapters`. The Java adapter is located in the *XAD_ROOT*`/adapters/XADDemo_Adapter/java` directory. The C adapter is located in the *XAD_ROOT*`/adapters/XADDemo_Adapter/c` directory.

XSLT Stylesheets

There are two XSLT stylesheets for this example:

- `XADDemoIn`
- `XADDemoOut`

The XAD installer uses the `xad_demo_install` script to import the process development plans and these stylesheets into your specified repository and register them with the backbone and an iIS process engine. `XADDemoIn` processes messages from the adapter to the iIS proxy. `XADDemoOut` processes messages from the iIS proxy to the adapter.

Setup Script

The example setup script is located at *XAD_ROOT*`/install/bin/xad_demo_install`. This script is called by the XAD installer to configure the example. You can also edit and run this script after installation to reconfigure the example. Table A-1 describes the variables that this script sets.

Table A-1 Variables Set by the `xad_demo_install` Script

Variable	Description
`XAD_DEMO_TRANSPORT`	Transport mechanism used by the iIS proxy to communicate with the adapter's listener. Default value is `HTTP`. Valid values are `HTTP`, `JMS`, and `SOAP` (for a C adapter, the value must be `HTTP`).
`XAD_REPOS`	(Set during installation.) Name of the UDS repository into which the plans for the iIS example application are imported. This value must match the value in the Repository Name field of the UDS Control Panel, for example, `bt:c:\forte\repos\xad`.
`XAD_WORKSPACE`	(Set during installation.) The name of the UDS workspace in which the plans for the iIS example application are saved, for example, `xaddemo`.
`FNTEST_URL`	(Set during installation.) DNS name for the machine on which the adapter's listener is located. (The DNS name is the host portion of the machine's URL, for example, `titan.varrius.com`.)

Table A-1 Variables Set by the `xad_demo_install` Script *(Continued)*

Variable	Description
`FNTEST_NODE_NAME`	(Set during installation.) The name of the UDS node on which the proxy for the iIS example application runs.
`FNTEST_ENGINE_NAME`	(Set during installation.) The name of the iIS process engine used to run the iIS example application, for example, `XADDemoEng`. This value is case sensitive.
`XAD_ADAPT_RULE_LOC`	The location of the XSLT rules for use by the iIS proxy (not needed for SOAP). The default location is *XAD_ROOT*`\adapters\XADDemo_Adapter\iis`.

Additional Scripts

Table A-2 describes the scripts called by the `xad_demo_install` setup script.

Table A-2 Scripts Called by the `xad_demo_install` Script

Script	Description
`importplans.scr`	*XAD_ROOT*`\install\forte\scripts \importplans.scr` Imports the iIS plans into the repository and registers them with the process engine. The iIS engine and the repository server should be running when this script is invoked.
`xad_jmq2register`	*XAD_ROOT*`\install\bin\xad_jmq2register` Sets up JMS environment (if the transport is JMS) and registers the queue or topic names with iMQ. This script calls the `xad_jms_env` script to set up the environment.
`xad_jms_env`	*XAD_ROOT*`\install\bin\xad_jms_env` Sets up the JMS environment. Specifies values for Q name, Reply Q name, Adapter name, JMS broker name, JNDI provider type (file/ldap) and others. These variables are needed by the XAD JMS transport and the iIS backbone.

`xadbackbone_inst_http`
`xadbackbone_inst_jms`
`xadbackbone_inst_soap`
	XAD_ROOT`/install/forte/scripts/`
	
	Scripts that create and configure the backbone for the example.

Running the Example

The following overview outlines the steps required to set up and run the example. Subsequent subsections provide detailed instructions on how to perform each step.

➤ **To setup and run the example adapter and iIS client**

1. Ensure that the example iIS application and example adapter are installed.

   The iIS application and adapter must be installed using the XAD software installer. See the *XAD Installation Guide* for installation instructions.

2. Ensure that the transport mechanism is correctly configured (see "Configuring the Transport Mechanism").

3. Start the adapter (see "Starting the Adapter" on page 123).

4. Set up and start the transport listener (see "Deploying the XAD Listener Servlet" on page 124 or "Starting Up a JMS Listener" on page 124).

5. Test the adapter and transport driver (see "Testing the Adapter and Transport Driver" on page 125).

6. Run the adapter with the iIS client (see "Running the Adapter With the iIS Client" on page 129).

Configuring the Transport Mechanism

By default, the example is configured to use the example adapter, `XADDemo_Adapter`. Communication between the iIS proxy and adapter uses the XAD XML dialect over HTTP. If this is the configuration in which you want to run the example, (and the example's scripts have not previously been edited) you can skip this section.

If you want to use the example C adapter rather than the Java adapter, JMS rather than HTTP, or SOAP rather than the XAD XML dialect, you must reconfigure the transport mechanism accordingly.

There are two aspects to configuring the transport mechanism:

- If you want to use the C adapter, you must reconfigure the iIS proxy. By default the proxy is configured to communicate with the Java adapter. This procedure is described in "To configure the iIS proxy to communicate with the C adapter."
- If you want to use JMS or SOAP, you must reconfigure the transport and message protocols. This procedure is described in "To configure the transport and message protocols" on page 119.

Note that if you configure the example to run the C adapter, you must use the XAD XML dialect over HTTP. Also, you can send SOAP messages over HTTP only, not JMS.

➤ To configure the iIS proxy to communicate with the C adapter

1. Use the following `fnscript` commands to set the URL used by the iIS proxy to communicate with the C adapter.

   These commands assume that the `FNTEST_URL` variable is set for your command line. If it isn't set, substitute the appropriate value for `{$FNTEST_URL}`. See Table A-1 on page 115 for a description of this variable.

   ```
   FindBackbone XADDemoBB

   FindProxy XADDemoPxy

   Shutdown

   ClearAplURL

   AddAplURL http://{$FNTEST_URL}:12345/

   Startup
   ```

   NOTE If you intend to configure the transport and message protocols, as described in the next section ("To configure the transport and message protocols"), you can edit the `xadbackbone_inst_http` script as an alternative to using these `fnscript` commands.

   In this case, edit the script by uncommenting the lines:

   ```
   # ClearAplURL

   # AddAplURL http://${FNTEST_URL}:12345/
   ```

   Then comment out the previous `AddAplURL` command.

   See Table A-2 on page 116 for more information on the `xadbackbone_inst_http` script.

➤ To configure the transport and message protocols

1. Edit the `xad_demo_install` script located in the *XAD_ROOT*`/install/bin` directory.

   Set the `XAD_DEMO_TRANSPORT` variable to the appropriate value. Valid values are `http`, `jms`, and `soap` (for C adapters the value must be `http`). The default value is `http`.

   `http` – Use this value to send XAD dialect XML messages over HTTP.

   `jms` – Use this value to send XAD dialect XML messages over JMS.

   `soap` – Use this value to send SOAP messages over HTTP.

NOTE Comment out the line that calls *FORTE_ROOT*`\install\bin\cscript -i importplans.scr` if the plans for the iIS example application were imported into the repository during the XAD installation.

If you want to import the plans when you run the `xad_demo_install` script, do not comment out this line. But in this case, make sure that you start the iIS process engine and create the workspace and repository as specified in the *XAD Installation Guide*.

   If you set `XAD_DEMO_TRANSPORT` to `http` or `soap`, skip to Step 9.

2. If you are using JMS, edit the following variables in the `xad_jms_env` script located in the *XAD_ROOT*`/install/bin` directory:

Variable	**Description**
`JMS_DOMAIN`	Specifies the type of destination to which JMS messages are sent. The value `PublishSubscribe` specifies a topic destination. The value `PointToPoint` specifies a queue destination. For this example, ensure that this variable is set to `PointToPoint`.
`JNDI_PROVIDER`	Specifies whether the transport driver should look up queues and topics in the file system or in an LDAP server. Valid values are: • `file` • `ldap` The default value is `file`.

Variable	Description
`JNDI_URL`	Specifies the location of the JNDI provider. If the JNDI provider is the file system, this variable must point to the JMS `.bindings` file. The default value of this variable points to the default location of the `.bindings` file. If the JNDI provider is an LDAP server, this variable points to the LDAP server, for example: `ldap://fthiro.varrius.com:389/o=NetscapeRoot`

3. If you are using JMS, start the iMQ broker.
4. If you are using JMS, edit the following properties in the `XADDemo_Adapter.properties` file located in the *XAD_ROOT*`/install/java/props/jms/XADDemo_Adapter` directory:

Table A-3 JMS Properties for the iIS Example Application

Property	Description

`JNDIContextJMS`	If your `JNDI_PROVIDER` variable is set to `file`, this property must be set to:
	
	`com.sun.jndi.fscontext.RefFSContextFactory`
	
	This is the default value of this property.
	
	If your `JNDI_PROVIDER` variable is set to `ldap`, this property must be set to:
	
	`com.sun.jndi.ldap.LdapCtxFactory`

Table A-3 JMS Properties for the iIS Example Application *(Continued)*

Property	Description

`JNDIProviderJMS`	If your `JNDI_PROVIDER` variable is set to `file`, this property must be set to:
	`file:`*XAD_ROOT*`/install/java/props/jms/XADDemo_Adapter`
	where *XAD_ROOT* denotes the root directory of your XAD software. For example, if your XAD software is installed on a Windows platform in `C:\xad`, set this property to:
	`file:c:/xad/install/java/props/jms/XADDemo_Adapter`
	If your `JNDI_PROVIDER` variable is set to `ldap`, set this property to the URL of your LDAP server using the following syntax:
	`ldap://`*machineName.domain*:*portNumber*/o=*organization*
	For example:
	`ldap://sunflower.varrius.com:389/o=varrius.com`

`ModeOfCommunication`	Specifies the type of destination to which JMS messages are sent. For this example, ensure that this property is set to `ptp`.

`QueueConnectionFactory`	If your `JNDI_PROVIDER` variable is set to `file`, this property must be set to:
	`qcf`
	This is the default setting of this property.
	If your `JNDI_PROVIDER` variable is set to `ldap`, this property must be set to:
	`cn=qcf,ou=XADDemo_Adapter`

`Queue`	If your `JNDI_PROVIDER` variable is set to `file`, this property must be set to: XAD_Q This is the default setting of this property. If your `JNDI_PROVIDER` variable is set to `ldap`, this property must be set to: `cn=XAD_Q,ou=XADDemo_Adapter`

Table A-3 JMS Properties for the iIS Example Application *(Continued)*

Property	Description
`ReplyQueue`	If your `JNDI_PROVIDER` variable is set to `file`, this property must be set to: XAD_REPLY_Q This is the default setting of this property. If your `JNDI_PROVIDER` variable is set to `ldap`, this property must be set to: `cn=XAD_REPLY_Q,ou=XADDemo_Adapter`

5. From a command line, navigate to either *XAD_ROOT*`\adapters\XADDemo_Adapter\java` or *XAD_ROOT*`\adapters\XADDemo_Adapter\c`, depending on whether you want to run the Java or C adapter.
6. From the same command line, run the `setadapterenv` script.
7. If you are using JMS, run the `xad_jmq2register` script located in the *XAD_ROOT*`/install/bin` directory. Run this script from the same command line you used to run the `setadapterenv` script.

   The `xad_jmq2register` script registers queues or topics with the iMQ broker. It creates a bindings file named `.bindings` in *XAD_ROOT*`/install/java/props/jms/XADDemo_Adapter` if the `JNDI_PROVIDER` variable is set to `file`. If the `JNDI_PROVIDER` variable is set to `ldap`, the script creates entries in the LDAP server.
8. If you are using LDAP for registration, define an organizational unit (`ou`) with the name of the adapter (`XADDemo_Adapter`) in the LDAP server.
9. Run the `xad_demo_install` script.
10. Check the engine registrations and backbone configurations.
    - **a.** Use the iIS Console to see if the process engine contains registrations such as `XADDemo_PD`.
    - **b.** Use the following `fnscript` commands to check the backbone configuration settings:

      ```
      FindBackbone XADDemoBB
      ```

      ```
      showconf
      ```

Starting the Adapter

➤ **To start the adapter**

1. From a command line, navigate to either *XAD_ROOT*`\adapters\XADDemo_Adapter\java` or *XAD_ROOT*`\adapters\XADDemo_Adapter\c`, depending on whether you want to run the Java or C adapter.

2. Run the `setadapterenv` script.

3. *For the Java adapter only*. From the same command line, start the RMI registry as a background process.

   On a Windows operating system, use the command:

   ```
   start rmiregistry
   ```

   On a Solaris operating system, use the command:

   ```
   rmiregistry&
   ```

   On a Windows platform, the RMI registry process starts in a separate window. Minimize this window after it starts.

4. From the same command line and in the same directory, start either the Java adapter by executing the `xad_run_java_adapt` script, or start the C adapter by executing the `xad_run_c_adapt` script.

   When the Java adapter starts, the following message is displayed:

   ```
   registered successfully
   Starting logging services..
   ```

   When the C adapter starts, a message is displayed that gives the location of the adapter's log file.

   After the adapter starts, proceed to "Deploying the XAD Listener Servlet" on page 124 or "Starting Up a JMS Listener" on page 124 depending on the transport mechanism you are using.

Deploying the XAD Listener Servlet

If you are using HTTP to pass messages between the iIS client and the adapter, you must deploy the XAD listener servlet. Deploying the listener servlet entails:

- Configuring the servlet's properties file
- Installing the servlet on the web server
- Configuring the servlet attributes on the web server

For this example, the servlet's properties file is preconfigured. For instructions on installing the servlet on the web server and configuring the servlet attributes, see "Installing the Listener Servlet" on page 73.

Starting Up a JMS Listener

If you are using JMS to pass messages between the iIS client and the adapter, you must start a JMS listener.

➤ **To start a JMS listener**

1. Ensure iMQ Broker is running.
2. If you are using LDAP, ensure that iPlanet Directory Server is running.

   Also, make sure you have created an organizational unit (ou), as described in "Configuring the Transport Mechanism" on page 117.
3. Navigate to *XAD_ROOT*`/adapters/XADDemo_Adapter`.
4. Run the `setAdapterEnv` script.
5. Navigate to *XAD_ROOT*`/install/bin`.
6. Run the `xad_run_jms` script.

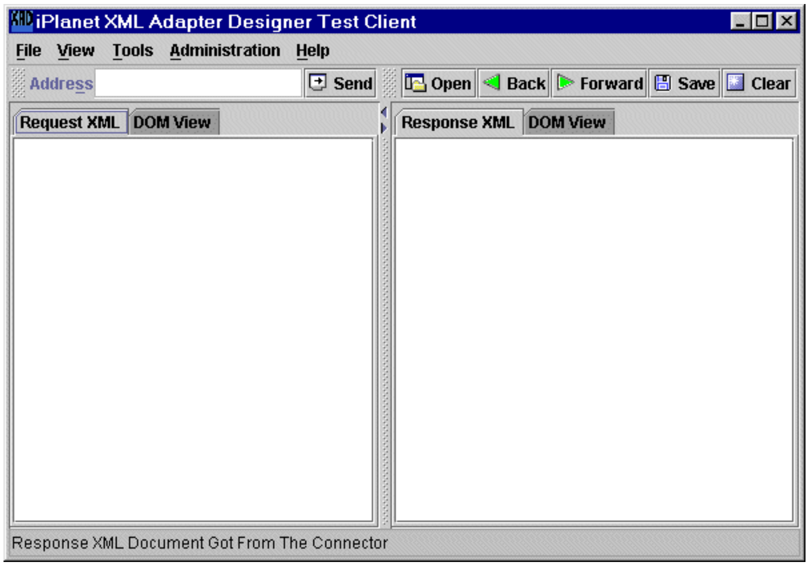
Testing the Adapter and Transport Driver

The XAD software includes a Java client for testing adapters. You use the test client to send request XML messages to an adapter and to display response XML messages returned by the adapter. The test client communicates with the adapter using the transport driver you set up in"Configuring the Transport Mechanism" on page 117 and "Deploying the XAD Listener Servlet" on page 124 or "Starting Up a JMS Listener" on page 124. The test client enables you to test the functionality of the adapter and the transport driver.

➤ **To run the test client and send a request XML message to the adapter**

1. Run the `xad_run_testclient` script in the *XAD_ROOT*`\install\bin` directory.

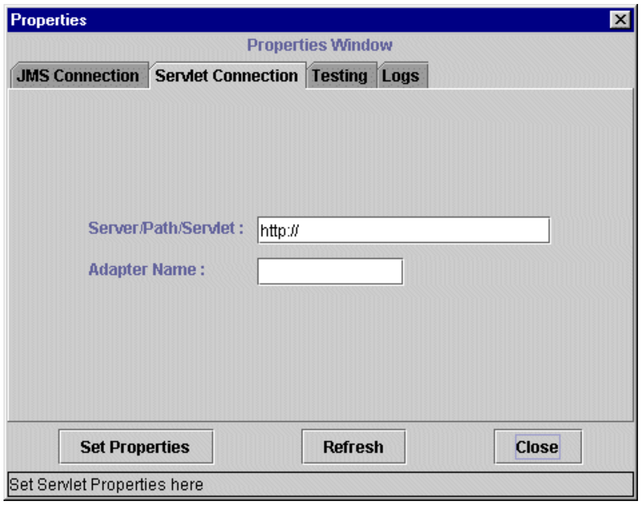
   The test client window opens.



Response XML Document Got From The Connector

2. Choose Tools > Properties.

   The Properties window opens.



Server/Path/Servlet : http://

Adapter Name :

Set Servlet Properties here

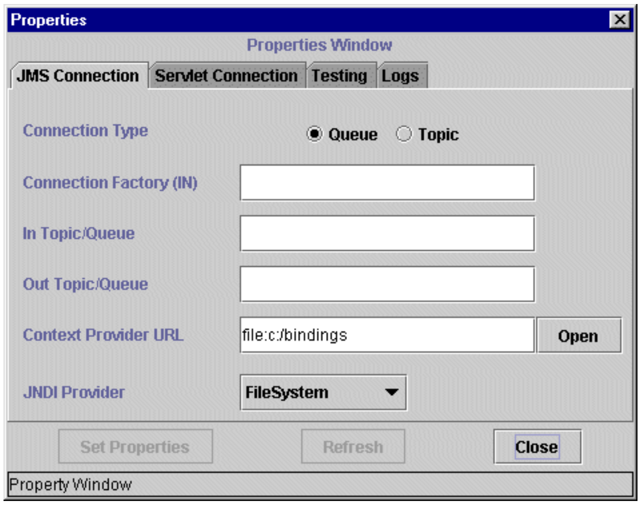
3. Ensure that the test client's Router Type is set correctly.

   Set it to Servlet if you are using HTTP. Set it to Message-Queue if you are using JMS. See "To set the router type for the test client" on page 89 for more information on this topic.

4. If your transport is HTTP, fill out the fields on the Servlet Connection tab using information from the following table.

Field	Value
Server/Path/Servlet	URL of the listener servlet. The URL contains the web server DNS name, the virtual path, and the servlet name, for example: `http://rex.varrius.com/adapter/XADServlet`
Adapter Name	Type `XADDemo_Adapter`. This field is case sensitive.

5. If your transport is JMS, click the JMS Connection tab and fill out the fields using information from the following table.



Connection Type: ● Queue ○ Topic
Connection Factory (IN):
In Topic/Queue:
Out Topic/Queue:
Context Provider URL: file:c:/bindings
JNDI Provider: FileSystem
Property Window

Field	Value
Connection Type	The type of destination to which JMS messages are sent. For this example, set this field to Queue.
Connection Factory (IN)	If the JNDI Provider field is set to Filesystem, set this field to: `qcf`
	If the JNDI Provider field is set to LDAP, set this field to: `cn=qcf,ou=XADDemo_Adapter`
In Topic/Queue	If the JNDI Provider field is set to Filesystem, set this field to: `XAD_Q`
	This is the default setting of this property.
	If the JNDI Provider field is set to LDAP, set this field to: `cn=XAD_Q,ou=XADDemo_Adapter`

Field	Value

Out Topic/Queue	If the JNDI Provider field is set to Filesystem, set this field to:
	`XAD_REPLY_Q`
	This is the default setting of this property.
	If the JNDI Provider field is set to LDAP, set this field to:
	`cn=XAD_REPLY_Q,ou=XADDemo_Adapter`

Context Provider URL	Specifies the URL of the JNDI provider.
	If the JNDI Provider field is set to Filesystem, set this field to the location of the JMS `.bindings` file. You must include the name of the file (`.bindings`) in the URL. The default location of this file is in the `install/java/props/jms/XADDemo_Adapter` directory of your XAD installation. For example, on a Windows platform, the URL could be:
	`file:c:/xad/install/java/props/jms/XADDemo_Adapter/.bindings`
	If the JNDI Provider field is set to LDAP, set this field to the URL of the LDAP server, for example:
	`ldap://hiro.varrius.com:389/o=NetscapeRoot`

JNDI Provider	Choose Filesystem if the transport driver should look up queues and topics in the filesystem. Choose LDAP if the transport driver should look up queues and topics in an LDAP server.

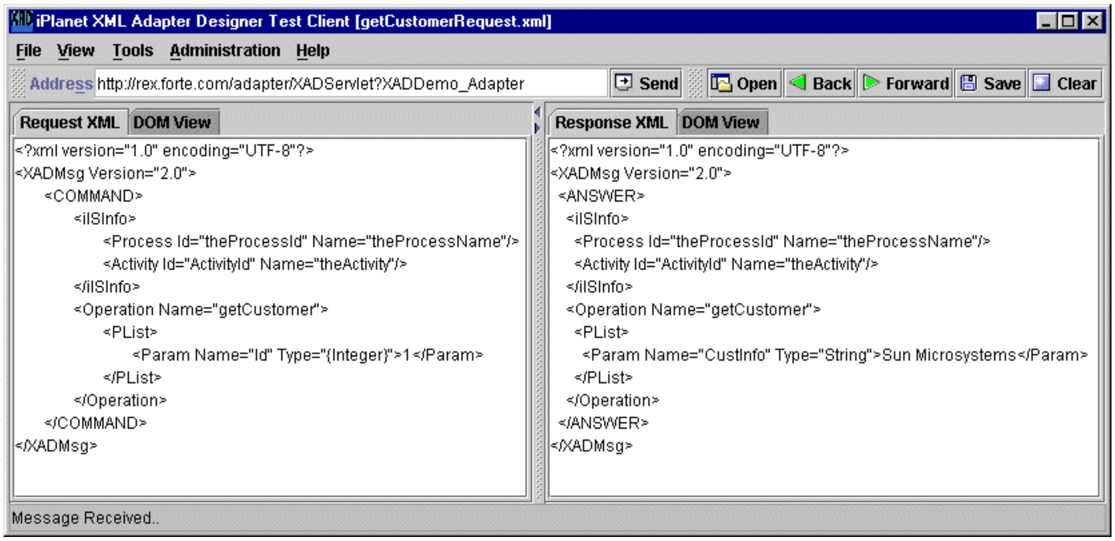
6. Click Set Properties.
7. Choose a request XML message to send to the adapter.
   a. Choose File > Open and then navigate to the *XAD_ROOT*`\adapters\XADDemo_Adapter\xml` directory.
   b. Select a file, such as `getCustomerRequest.xml`.

      To test SOAP messages, select a file such as `getCustomerRequestSOAP.xml`.
   c. Click Open.

      The request XML message is displayed in the Request XML window.

8. Click Send.

The request XML message is sent to the adapter and the adapter's XML response message is displayed in the Response XML window.


iPlanet XML Adapter Designer Test Client [getCustomerRequest.xml]
File View Tools Administration Help
Address http://rex.forte.com/adapter/XADServlet?XADDemo_Adapter
Send Open Back Forward Save Clear
Request XML DOM View
Response XML DOM View

Request XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<XADMsg Version="2.0">
    <COMMAND>
        <iISInfo>
            <Process Id="theProcessId" Name="theProcessName"/>
            <Activity Id="ActivityId" Name="theActivity"/>
        </iISInfo>
        <Operation Name="getCustomer">
            <PList>
                <Param Name="Id" Type="{Integer}">1</Param>
            </PList>
        </Operation>
    </COMMAND>
</XADMsg>
```

Response XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<XADMsg Version="2.0">
 <ANSWER>
  <iISInfo>
   <Process Id="theProcessId" Name="theProcessName"/>
   <Activity Id="ActivityId" Name="theActivity"/>
  </iISInfo>
  <Operation Name="getCustomer">
   <PList>
    <Param Name="CustInfo" Type="String">Sun Microsystems</Param>
   </PList>
  </Operation>
 </ANSWER>
</XADMsg>
```

Message Received..

9. Click File > Exit to close the test client.

Running the Adapter With the iIS Client

If you installed the iIS example application included with the XAD software distribution, you can run the example adapter using the iIS application as a client. Running the example tests the functionality of:

- The XSLT rules that were generated by the XAD generator along with the example adapter
- The iIS proxy definition
- The iIS engine process definition

➤ **To run the example iIS client application with the example adapter**

1. Ensure that:

   - The example adapter is running
   - The iIS application's process engine is running (warm start the engine)
   - The transport driver is set up and running

2. Use the following `fnscript` commands to start the backbone manager and proxy:

   ```
   findbackbone XADDemoBB
   ```

   ```
   startup
   ```

3. Open the workspace in which your iIS application plans are defined.

   This workspace is named using the value you supplied for the Workspace field during installation. The default value is `XADDemo`.

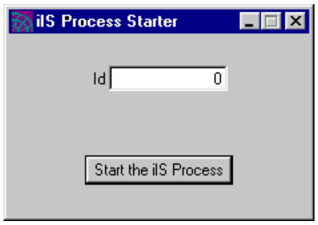
4. Open the `XADDemoClient` project.

5. Ensure that the constant `K_ENGINENAME_WHEN_NOENWAR` is set to the value you supplied for the Engine Name field during installation.

   The default value is `XADDemoEng`.

6. From your open workspace, run the `XADDemoClient` project.

   The iIS Process Starter window opens.

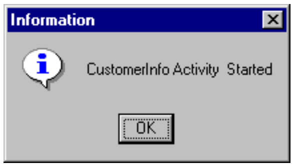


7. Set the value of the ID field to `1`.

8. Click the button labeled Start the iIS Process.

The client starts the iIS process and sets the `ID` process attribute to the value in the customer ID field.

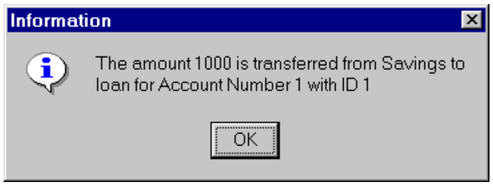
A message dialog box opens to indicate that the `CustomerInfo` Activity has started.



CustomerInfo Activity Started

9. Click OK, and continue clicking OK as more message dialog boxes open.

The last dialog box to open indicates that a sum of money was transferred from one account to another. The process is finished.



The amount 1000 is transferred from Savings to loan for Account Number 1 with ID 1

You can repeat the process using another ID number, such as `2` or `3`.

When you are finished, you can shut down the backbone and the process engine.

XAD Example Adapters

This appendix describes how to configure and run the example adapters provided with the XAD distribution.

Java Adapter Example

This section describes how to configure and run the `XADSamples_Java` example adapter. This example illustrates language-specific issues with Java adapters but does not show integration with iIS.

About the Java Adapter Example

This example demonstrates the use of complex data types, such as arrays and user-defined classes, with Java adapters. The example utilizes three components:

- An example EIS (Enterprise Information System) application
- A Java adapter designed for the EIS application interface
- A test client

The example is installed as part of your XAD installation in the *XAD_ROOT*`/adapters/XADSamples_Java` directory. Within this directory, the adapter is located at:

```
java/src/XADSamples_Java.java
```

The EIS application is located at:

```
java/src/XADDemo_ImpClass.java
```

Running the Java Adapter Example

➤ To run the Java adapter example

1. Start the adapter (see "To start the adapter").
2. Start the transport listener.

   For HTTP transport, start, the listener servlet (see "To configure and start the listener servlet").

   For JMS transport, start the JMS listener (see "To configure and start the JMS listener").
3. Run the test client (see "To run the test client" on page 135).

➤ To start the adapter

1. Navigate to *XAD_ROOT*`/adapters/XADSamples_Java/java`.
2. Run the `setAdapterEnv` script or batch file.
3. Start `rmiregistry`.
4. Navigate to *XAD_ROOT*`/install/bin`.
5. Run the `xad_run_java_adapt` script or batch file.

➤ To configure and start the listener servlet

1. Ensure iPlanet Web Server is configured with `XADServlet` as described in "Running the iIS Client Example" on page 113 and running.

➤ To configure and start the JMS listener

1. Open the `XADSamples_Java.properties` file located in the *XAD_ROOT*`/install/java/props/jms/XXADSamples_Java` directory and locate the line that sets the `JNDIProviderJMS variable`.
2. Uncomment the line and edit it so that it points to your XAD installation directory (the unedited value points to a default installation location).
3. Ensure iMQ Broker is running.
4. If you are using LDAP, ensure that iPlanet Directory Server is running.
5. Navigate to *XAD_ROOT*`/adapters/XADSamples_Java`.

6. Run the `setadapterenv` script or batch file.
7. Navigate to *XAD_ROOT*`/install/bin`.
8. Run the `xad_run_jms` script or batch file.

➤ **To run the test client**

1. Navigate to *XAD_ROOT*`/install/bin`.
2. Run the `xad_run_testclient` script or batch file.

   The test client starts.
3. Choose File > Open and load a sample XML request from *XAD_ROOT*`/adapters/XADSamples_Java/xml`.
4. Choose Tools > Properties and set the transport properties as follows:

   a. *For HTTP transport,* set the following fields on the Servlet Connection tab:

      Server/Path/Servlet: *webServerName*`:`*port*`/adapter/XADServlet`

      (For example, `localhost:80/adapter/XADServlet`)

      Adapter Name: `XADSamples_Java`

   b. *For JMS transport,* set the following fields on the JMS Connection tab:

      Connection Type: `q`

      Connection Factory (IN): `qcf`

      In Topic/Queue: `XAD_Q`

      Out Topic/Queue: `XAD_REPLY_Q`

      JNDI Provider: Choose Filesystem to look up topics in the filesystem. Choose LDAP to look up topics in an LDAP server.

      Context Provider URL: If the JNDI Provider is Filesystem, set to the location of the JMS `.bindings` file, for example:

      XAD_ROOT`/install/java/props/jms/XADDemo_Adapter/.bindings`

      If the JNDI Provider field is LDAP, set this field to the URL of the LDAP server, for example:

      `ldap://fthiro.varrius.com:389/o=NetscapeRoot`

5. Click send.

   The request is sent and the response is displayed in the test client's Response XML pane.

For more information, refer to "Deploying and Testing an Adapter" on page 81.

C Adapter Example

This section describes how to configure and run the example C adapter. This example illustrates language specific issues with C adapters but does not show integration with iIS.

About the C Adapter Example

This example demonstrates the use of complex data types, such as arrays and user-defined structures, with C adapters. The example utilizes the following components:

- An example EIS (Enterprise Information System) application
- A C adapter designed for the EIS application interface
- A test client

The example is installed as part of your XAD installation in the *XAD_ROOT*`/adapters/XADSamples_C directory`. Within this directory, the adapter is located at `c/src/XADSamples_C.c`, and the EIS application is located at `c/src/Sample.c`.

Running the C Adapter Example

NOTE For C adapters the HTTP listener is included within the adapter. The port of the HTTP listener is set to 12345 in the `setadapterenv` script.

➤ **To run the C adapter example**

1. Start the adapter (see "To start the adapter").
2. Run the test client (see "To run the test client").

➤ **To start the adapter**

1. Navigate to *XAD_ROOT*`/adapters/XADSamples_C/c`.
2. Run the `setadapterenv` script or batch file.
3. Navigate to *XAD_ROOT*`/install/bin`.
4. Run the `xad_run_c_adapt` script or batch file.

➤ **To run the test client**

1. Navigate to *XAD_ROOT*`/install/bin`.
2. Run the `xad_run_testclient` script or batch file.

   The test client starts.
3. Choose File > Open and load a sample XML request from *XAD_ROOT*`/adapters/XADSamples_C/xml`.
4. Choose Tools > Properties, and on the Servlet Connection tab, set the transport properties as follows:

   Server/Path/Servlet: *webServerName*`:12345`

   (For example, `localhost:12345`)

   Adapter Name:

   (Leave this field blank.)
5. Click send.

   The request is sent and the response is displayed in the test client's Response XML pane.

For more information, refer to "Deploying and Testing an Adapter" on page 81.

Appendix C

XAD Adapters at Runtime

This appendix describes how XAD adapters process requests at runtime. Java and C adapters are discussed separately.

XAD Java Adapter at Runtime

1. The client sends an XML request to the transport driver.

   XAD provides two transport drivers for Java adapters, a listener servlet for communicating over HTTP and a JMS listener for communicating over JMS.

2. Using RMI, the transport driver forwards the request as a string to the `XADRemoteServer` class in the adapter engine.

   The adapter engine consists of two classes, `adapterEngine` and `XADRemoteServer`. The `XADRemoteServer` class interacts with the rest of the Java environment. Generated adapters subclass the `XADRemoteServer` class to process requests to execute specific methods and their parameters. The `adapterEngine` class is responsible for maintaining a list of dialect engines and communicating with them as well as communicating with the administrative engine.

3. The adapter engine queries the available dialect engines and sends the request to the correct dialect engine.

4. The dialect engine converts the XML document to an XML dialect-neutral representation (an `InteractionSpec` object).

5. The adapter engine sends this `InteractionSpec` object to the generated adapter.

6. The adapter executes the API method call and uses the results of the method call to build an output `InteractionSpec` object, which it returns to the adapter engine.

7. The adapter engine forwards the output `InteractionSpec` object to the dialect engine.

8. The dialect engine builds the XML response from the output `InteractionSpec` object and returns it to the client via the transport driver.

9. The adapter engine intercepts any errors and generates a standard XML error document. It also starts the administrative engine, which executes and responds to administrative commands sent to the adapter.

XAD C Adapter at Runtime

1. The adapter engine, has hooks to the generated code by means of function pointers. One such hook is the function `FAD__InitFunction` in the generated code. This function is called once, when the adapter starts up.

2. The adapter's integrated HTTP server listens at the port specified by the adapter startup argument `/p:<port-no>`.

   By default this value is set to `12345` by the `setadapterenv` script.

3. The adapter engine receives an XML request. The administration engine, which performs the system management tasks, verifies the request.

4. The adapter engine identifies the dialect and sends the request to the correct dialect engine.

   If the target dialect engine is not found, an error XML response is built.

5. The dialect engine parses the XML document and builds an XML dialect-neutral representation that contains the details of the request parameters, including their values.

6. The dialect engine calls the `ProcessRequest` function in the generated code, which then executes the target API function.

7. The adapter fills the XML dialect-neutral representation with the results of the function call.

8. The dialect engine builds the XML response from the intermediate structure.

9. The XML response is passed to the adapter engine, which returns it to the client.

Index