SYBASE®

Java Support for e-Biz Impact

e-Biz Impact™

5.4.5

DOCUMENT ID: DC26541-01-0545-01

LAST REVISED: July 2005

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

About This Book

Audience

This book is for system and application administrators and application developers who want to create Java applications for use with e-Biz Impact™ 5.4.5.

How to use this book

This book contains these chapters:

- Chapter 1, “Java Support,” describes the Java support available in e-Biz Impact version 5.4.5, including access to Web services using Simple Object Access Protocol (SOAP).
- Chapter 2, “Secure TCP Socket Connections,” explains how to implement secure or standard (plaintext) TCP socket connections in an e-Biz Impact implementation.
- Chapter 3, “Java Support Samples,” provides codes samples that illustrate how to implement e-Biz Impact Java support.
- Chapter 4, “Web Services Templates,” provides sample bridge templates that demonstrate how to have Java-based acquisition and delivery AIMs (the modules that deliver or receive data to and from end points) interoperate with SOAP calls.

Related documents

e-Biz Impact documentation The following documents are available on the Sybase Getting Started CD in the e-Biz Impact 5.4.5 product container:

- The e-Biz Impact installation guide explains how to install the e-Biz Impact software.
- The e-Biz Impact release bulletin contains last-minute information not documented elsewhere.

e-Biz Impact online documentation The following e-Biz Impact documents are available in PDF and DynaText format on the e-Biz Impact 5.4.5 Sybooks CD:

- The *e-Biz Impact Application Guide* provides information about the different types of applications you create and use in an e-Biz Impact implementation.

- The *e-Biz Impact Authorization Guide* explains how to configure e-Biz Impact security.
- *e-Biz Impact Command Line Tools* describes how to execute e-Biz Impact functionality from a command line.
- The *e-Biz Impact Configurator Guide* explains how to configure e-Biz Impact using the Configurator.
- The *e-Biz Impact Feature Guide* describes new features, documentation updates, and fixed bugs in this version of e-Biz Impact.
- The *e-Biz Impact Getting Started Guide* provides information to help you quickly become familiar with e-Biz Impact.
- The *Monitoring e-Biz Impact* explains how to use the Global Console, the Event Monitor, and alerts to monitor e-Biz Impact transactions and events. It also describes how e-Biz Impact uses the standard Simple Network Management Protocol (SNMP).
- *Java Support in e-Biz Impact* (this book) describes the Java support available in e-Biz Impact 5.4.5.
- The *e-Biz Impact MSG-IDE Guide* describes MSG-IDE terminology and explains basic concepts that are used to build Object Definition Language (ODL) applications.
- The *e-Biz Impact ODL Guide* provides a reference to Object Definition Language (ODL) functions and objects. ODL is a high-level programming language that lets the developer further customize programs created with the IDE tools.
- The *e-Biz Impact TRAN-IDE Guide* describes how to use the TRAN-IDE tool to build e-Biz Impact production objects, which define incoming data and the output transactions produced from that data.

Note The *e-Biz Impact ODL Application Guide* has been incorporated into the *e-Biz Impact ODL Guide*.

The *e-Biz Impact Alerts Guide*, the *e-Biz Impact SNMP Guide*, and the *e-Biz Impact Global Console Guide* have been combined into a new guide—*Monitoring e-Biz Impact*.

Adaptive Server Anywhere documentation The e-Biz Impact installation includes Adaptive Server® Anywhere, which is used to set up a Data Source Name (DSN) used with e-Biz Impact security and authorization. To reference

Adaptive Server Anywhere documentation, go to the Sybase Product Manuals Web site at Product Manuals at http://www.sybase.com/support/manuals/, select SQL Anywhere Studio from the product drop-down list, and click Go.

Note the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.

Other sources of information

Use the Sybase Getting Started CD, the SyBooks CD, and the Sybase Product Manuals Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.
- The SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format.

  Some documentation may be provided in PDF format, which you can access through the PDF directory on the SyBooks CD. To read or print the PDF files, you need Adobe Acrobat Reader.

  Refer to the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.
- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Sybase Product Manuals Web site, go to Product Manuals at http://www.sybase.com/support/manuals/.

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ **Finding the latest information on product certifications**

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Select Products from the navigation bar on the left.

3 Select a product name from the product list and click Go.

4 Select the Certification Report filter, specify a time frame, and click Go.

5 Click a Certification Report title to display the report.

❖ **Creating a personalized view of the Sybase Web site (including support pages)**

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

❖ **Finding the latest information on EBFs and software maintenance**

1 Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.

2 Select EBFs/Maintenance. Enter user name and password information, if prompted (for existing Web accounts) or create a new account (a free service).

3 Select a product.

4 Specify a time frame and click Go.

5 Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Conventions

The syntax conventions used in this manual are:

Key	Definition
commands and methods	Command names, command option names, utility names, utility flags, Java methods/classes/packages, and other keywords are in lowercase Arial font.

Key	Definition
variable	Italic font indicates: • Program variables, such as *myServer* • Parts of input text that must be substituted, for example: *Server*.log • File names
File | Save	Menu names and menu items are displayed in plain text. The vertical bar shows you how to navigate menu selections. For example, File | Save indicates "select Save from the File menu."
`package 1`	Monospace font indicates: • Information that you enter in a graphical user interface, at a command line, or as program text • Sample program fragments • Sample output fragments

Accessibility features

This document is available in an HTML version that is specialized for accessibility. You can navigate the HTML with an adaptive technology such as a screen reader, or view it with a screen enlarger.

For information about how Sybase supports accessibility, see Sybase Accessibility at http://www.sybase.com/accessibility. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

CHAPTER 1

Java Support

This chapter provides an overview of the Java support available in e-Biz Impact 5.4.5.

Introduction

Java support for e-Biz Impact provides a practical way for users to interact with e-Biz Impact from a Java application by providing a set of Java classes, interfaces, and exceptions that allow you to:

- Extend native Java 2 Platform, Standard Edition (J2SE) and Java 2 Platform, Enterprise Edition (J2EE) services to e-Biz Impact
- Send Distributed Function Calls (DFCs) to e-Biz Impact from a Java client application
- Send command and control (CNC) requests to e-Biz Impact from a Java client application
- Receive or distribute transactions through Secure Sockets Layer (SSL) to e-Biz Impact
- Provide Web services access to and from e-Biz Impact
- Plug Java applications into e-Biz Impact

Java support for e-Biz Impact falls into two categories:

- **Remote access** Allows access to e-Biz Impact from an external Java process.
- **Plug-in access** Support for Java plug-in applications hosted directly by the e-Biz Impact runtime engine.

A message layer offers containers for the definition of DFC and CNC messages, arguments, and argument wrappers.

Note e-Biz Impact Java support requires JRE 1.4.2_05 or later.

When you run Java plug-in applications, run the cluster with the same version of JVM that you used to compile your plug-in applications, and keep the JVM and JDK versions in sync.

For example:

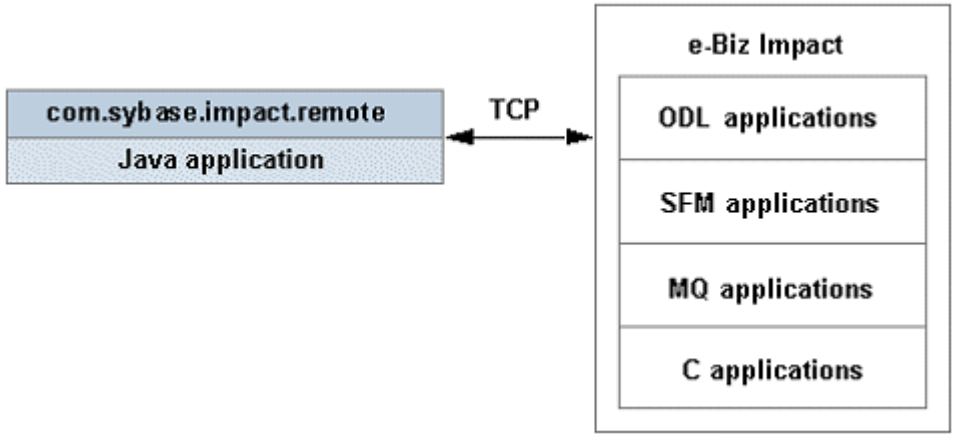
- Classes compiled with JDK 5 do not run properly if the cluster is started with JVM client version 1.4.
- Classes compiled with JDK 5 run properly if the cluster is started with JVM client version 5.
- Classes compiled with JDK 1.4 run properly if the cluster is started with JVM client version 1.4.

Remote access

Remote access provides a set of Java classes that allow an external Java application to:

- Exchange function calls with other e-Biz Impact applications
- Control the e-Biz Impact runtime engine by issuing command and control (CNC) requests

Figure 1-1: Remote Java support



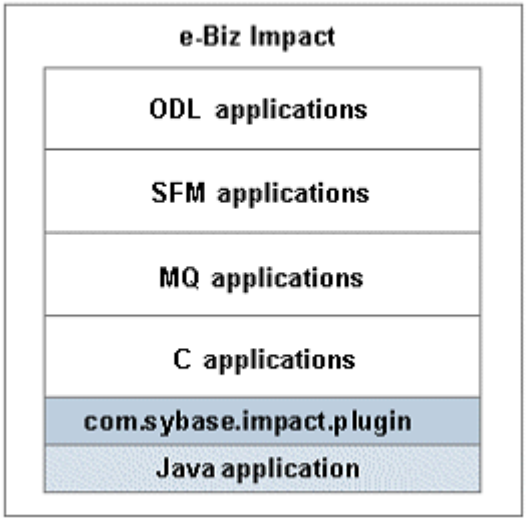
Remote Java support allows you to:

- Extend native J2SE and J2EE services to e-Biz Impact
- Collect transaction data from a Web application and send the transaction into an e-Biz Impact Store and Forward Manager (SFM) application
- Automate e-Biz Impact CNC operations natively in Java
- Receive function calls from e-Biz Impact for transaction delivery to Java applications

Plug-in access

Plug-in Java support enables Java plug-in applications to be hosted directly by the e-Biz Impact runtime engine. A toolkit allows developers to construct Java applications that can receive or send distributed function calls, access application log files, change an application's status, and so on.

Figure 1-2: Plug-in Java support



Plug-in Java support allows you to:

- Extend native J2SE and J2EE services to e-Biz Impact
- Access Web services to and from e-Biz Impact
- Receive or distribute transactions through SSL

You control and configure Java plug-in applications using existing e-Biz Impact console tools—the Configurator, the Global Console, and the Authorization Console.

Web services

e-Biz Impact 5.4.5 includes sample bridge templates that demonstrate how to have Java-based acquisition and delivery AIMs (the modules that deliver or receive data to and from a Store and Forward Manager) that are interoperable with SOAP calls.

Typically, a SOAP acquisition AIM is a Java-based class deployed as a service in a SOAP engine (for example, Sybase Web Services in EAServer). That class also uses the new e-Biz Impact Java support remote classes, allowing the acquisition AIM to function as a SOAP/DFC bridge.

Conversely, a SOAP delivery AIM is a Java-based DFC/SOAP bridge that operates as an e-Biz Impact Java plug-in. The class exposes a DFC interface to other e-Biz Impact applications, but upon a DFC call, internally issues a synchronous SOAP call.

Use the template samples provided in Chapter 4, “Web Services Templates,”to create your own bridge that extends the reach of the e-Biz Impact environment out to Web services or SOAP.

Secure TCP support

e-Biz Impact version 5.4.5 provides secure or standard (plaintext) socket communication using support classes that are included in the com.sybase.impact.tcp package. Interaction with secure or standard sockets is performed using the TcpProxy and TcpService classes.

For more information, see Chapter 2, “Secure TCP Socket Connections,” for instructions.

Javadoc

The Javadoc APIs are provided with the Java support functionality for e-Biz Impact.

Samples

Sample code, which demonstrate how to use the new library, is provided with the Java support for e-Biz Impact. See Chapter 3, “Java Support Samples.”

Configuring Java support

To add Java applications that you have created, open the e-Biz Impact Configurator, right-click a Controller in the tree view, and select New | Java from the menu.

To modify a new Java application, double-click the Java icon in the right pane to bring up the Java Properties window.

Java application properties

Complete these fields below when you create or modify a Java application in the Configurator.

General tab

Java Application

- Debug – enables debugging, which logs messages to the console output, error stream, and temporary files.
- Active – have e-Biz Impact start this application, which allows you to configure a Java application without putting it into production.
- Tracking – set the application's working directory automatically, which dynamically keeps the working directory the same as the application's name.
- Name – enter the name of the Java application. The name must begin with a letter and contain only letters, numbers, and underscores (_).
- Working Directory – enter the path to the Java application working directory, which is where the configuration file associated with this Java application resides.
- Plug-in Class Name – enter the name of the class that extends the application; for example, myBob.class or myBob. You enter the location of this class in the Class Path Items list box. See "Plug-in Class Name and Class Path Items field entries" on page 8 for more information about this field.
- Command Line – enter any command line arguments used by the Java application. This option can be left blank.
- Log Limit (K) – enter the size of the Java application log file. The default is 1000, which allows a balance of storage and disk usage. This field is optional.
- Depth – defines the number of copies of the log file that e-Biz Impact maintains. The default setting of 2 indicates that e-Biz Impact creates an additional file when the main activity log reaches its maximum size. If the second file reaches the limit, e-Biz Impact rolls back to the first file, overwriting the content.

- Class Path Items – allows you to list the path to the class, JAR, or package for this Java application to execute.
  - Name – enter the name of the JAR file, which can include the path.
  - Click Add to add the new JAR file, click Delete to remove the file, and click Modify to change the selected JAR file's name.

  See "Plug-in Class Name and Class Path Items field entries" on page 8 for more information about this field.
- Instances –
  - Min – enter the minimum number of instances of this Java application that e-Biz Impact will spawn. This value must be at least "1".
  - Max – enter the maximum number of instances of this Java application that e-biz Impact will spawn. This value must be greater than or equal to the value specified for Min.
- Custom Keys – this option allows you to insert custom key values into the Java configuration file. Custom keys are defined in groups, and each group is configured through the Custom Keys Info form.

  Note Configuring custom keys is advanced functionality. Usually, this option remains blank. For more information, see the *e-Biz Impact Configurator Guide*.

  - New – click New to display the form that allows you to add and configure a new custom key group.
    - Name – enter the name of the new custom key group.
    - Key – enter the name of one specific key that belongs to this group.
    - Value – enter the value of the key.

    Click Add to add the new custom key group, click Delete to remove a group, and click Modify to edit an existing group.
  - Delete – remove the selected custom key group.
  - Edit – modify the configuration of the selected custom key group. Double-clicking a custom key group also displays the reconfiguration form.

Plug-in Class Name and Class Path Items field entries

When you specify the class that extends the application (Plug-in Class Name) and the path to that class (Class Path Items), your entries fall into one of four categories:

Package, No; JAR, No The class is not part of a package or a JAR file.

- Plug-in Class Name – enter the class name; for example, myBob.class.
- Class Path Items – enter the path to the class; for example *D:\working*.

Package, No; JAR, Yes The class is not part of a package, but is part of a JAR file.

- Plug-in Class Name – enter the class name; for example, myBob.class.
- Class Path Items – enter the path to the JAR file that contains the class; for example *D:\working\MyJar.jar*.

Package Yes; JAR, No The class is part of a package file, but not part of a JAR file.

- Plug-in Class Name – enter the class name; for example, myBob.class.
- Class Path Items – enter the path to the package file; for example, *D:\working*.

Package, Yes; JAR, Yes The class is part of a package and a JAR file.

- Plug-in Class Name – enter the class name; for example, com.foo.myBob.
- Class Path Items – enter the path to the class; for example *D:\working\com\foo\MyJar.jar*.

Java package names consists of a series of alphanumeric characters separated by periods. Java expects one-to-one mapping of the package name and the file system directory structure; that is, put the file in the directory structure that mirrors the package name. For example, the Java classes that belong to the package "scheme" should be in a directory also named "scheme." *D:\working\com\foo\myBob.class*, the package name would be com.foo.myBob.

DFC tab

- Distributed Functions – list the DFC entry points into the Java application. Frequently, these are servproc and servayt.
- Name – enter the distributed function's name, which should match the function name override specified in the SFM.

- Flavor – enter the function's flavor, which should match the destination flavor defined in the SFM's properties on the Routing tab. See the *e-Biz Impact Configuration Guide* and the *e-Biz Impact Getting Started Guide*, Chapter 6, "Configuring e-Biz Impact," for more information about SFM routing destinations.
- Timeout – enter the amount of time, in milliseconds, that the source waits for a response from the receiving application before timing out.
- Availability – lets you define when a function is available to be called. This is optional advanced functionality.

Click Add. The function appears in the Distributed Functions list.

Click Modify to edit a function that is selected in the list.

Click Delete to remove a selected function.

Click Clear to remove all of the functions listed.

Notes tab

Select Description and enter an optional description or note about this Java application. This information is only for reference and does not affect the e-Biz Impact environment.

When you finish your entries, click Apply, then click OK to close the window.

See *e-Biz Impact Getting Started Guide* for detailed instructions on the remaining e-Biz Impact work flow.

Implementation

Java support for e-Biz Impact uses the following package structure:

```
com.sybase.impact.<package>
```

where *<package>* is any of the following:

- remote – services for access to a remote instance of e-Biz Impact.
  - remote.spi – service provider implementation for remote support.
- messaging – core elements that define message containers and arguments.
- plugin – services for building Java applications for e-Biz Impact

  - plugin.spi – service provider implementation for plug-in support.
- utils – utility classes for data conversion, storage, and so on.

Note See the Javadocs for detailed information about the API. See Chapter 3, “Java Support Samples,” for Java support code samples.

Remote layer

Remote Java support provides:

- Remote access to e-Biz Impact from an external Java process
- Message reception and distribution

The underlying interprocess communication (IPC) uses socket connections.

DfcProxy

The DfcProxy class allows a Java client application to connect to an e-Biz Impact server and make generic function calls into a remote instance of e-Biz Impact:

1 Use the connect function, specifying the host and the TCP port of an e-Biz Impact manager process, to establish a session with e-Biz Impact.

2 Use the SendDfc function to make distributed function calls into e-Biz Impact.

3 Use disconnect to terminate the session.

SfmDfcProxy

The SfmDfcProxy class offers easy access to an SFM application that resides in a remote instance of e-Biz Impact. SfmDfcProxy allows you to send transactions into a SFM and specify advanced options for processing those transactions.

DfcService

The DfcService class implements the ServerListner interface, which allows a Java client application to start a TCP server and listen for remote function calls from e-Biz Impact. When the service receives a function call, it invokes the function implementation of the custom Java client application.

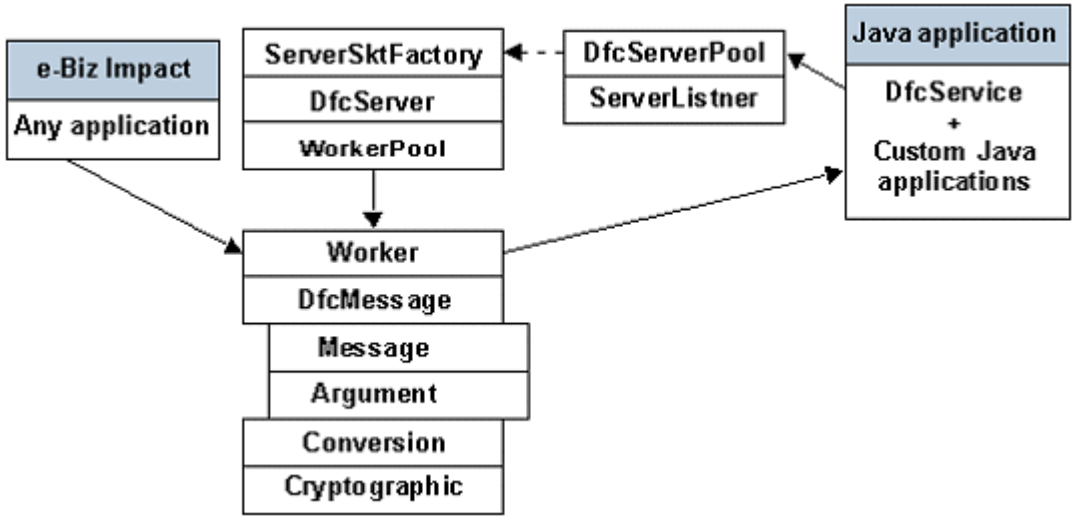
DfcService is a resource that can be shared across threads within the same process. Internally, the service makes use of a class factory. Also, the service has the capacity to adjust to the workload associated with incoming client requests.

- Use setMaxWorkerThreadCount to limit the number of concurrent threads available to fulfill client requests.
- Use addApplication or removeApplication to add or remove Java client applications.

When adding an application, the service looks into each application and registers all functions. The service uses the same flavor for all application functions.

DfcService creates an instance of DfcServer, then registers itself for notifications of DfcMessages.

Figure 1-3: DfcService application



DfcService starts the DfcServer. The DfcServer starts a server socket channel on a specified TcpPort. This channel receives requests from an instance of e-Biz Impact.

DfcService allows for the registration of applications. Upon registration, the DfcService registers each function into the DfcServer.

When the DfcServer receives a message from e-Biz Impact, it requests a worker from a WorkerPool. The size of the WorkerPool is dynamic and adjusts to the workload by the DfcServer. The maxWorkerThreadCount property defines the maximum number of worker threads.

The DfcServer delegates the work to the worker thread. This worker thread:

- Drains the channel
- Serializes the incoming raw data
- Analyses the message type, function name, and flavor
- Locates a registered ServerListner that provides an implementation for the received function

The worker then invokes the processMessage of the ServerListner. The DfcMessage is passed to the DfcService. The DfcService takes the first function that matches by name and looks for the function argument data type. If the type is ArgumentsInterface, the DfcService composes an argument wrapper. The DfcService invokes the function in the consumer application. After invocation, the DfcService sets the return argument of the incoming message, cleans up input-only arguments, and sends the response message back to e-Biz Impact. When a worker is finished fulfilling a request, it returns itself to the pool.

CncProxy

The CncProxy class allows a Java client application to connect to and issue CNC requests to a remote instance of e-Biz Impact.

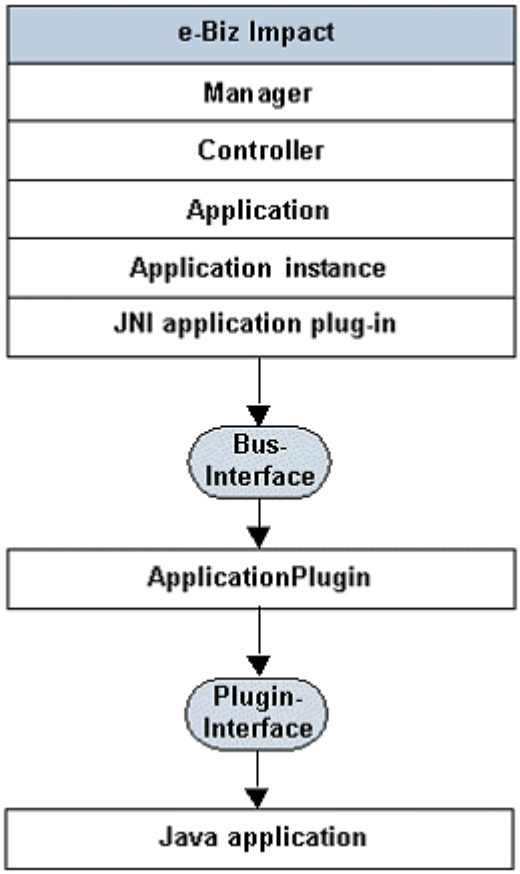
- Use the connect function, specifying the host and the TCP port of an e-Biz Impact manager process, to establish a session with e-Biz Impact.
- Use the login function, specifying the user name and password of an authorized user, to authenticate the session.
- Use any of the functions to execute command and control requests.
- Use logout to invalidate the session.
- Use disconnect to terminate the session.

Plug-in layer

Application

A javasrv application is a Java Native Interface (JNI) plug-in to the e-Biz Impact bus that hosts a Java application plug-in written by the user.

Figure 1-4: Java application



At runtime when the JNI plug-in receives a DFC and passes the DFC into an ApplicationPlugin Java class. The ApplicationPlugin invokes the corresponding function from a user-defined Java application.

The user-defined application extends Application to allow:

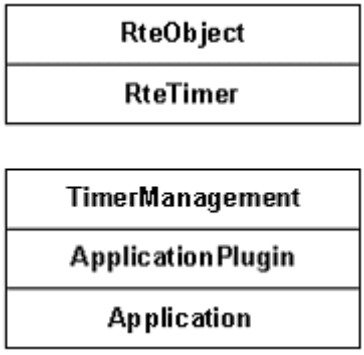
- Receiving function calls from the bus

- Sending function calls through the bus into another e-biz Impact application
- Setting the application instance status code and description
- Logging information into the application *xlog*
- Defining and accessing RteObjects like timers (see RteObject below)
- Suspending and resuming activity
- Getting configuration keys

RteObject

The runtime engine is built to deliver asynchronous notifications. e-Biz Impact version 5.4.5 offers RteTimer.

Figure 1-5: RteObject and RteTimer



A RteTimer can be created or killed through Application. Timers invoke a callback function when the timer expires. Timers can be defined as single events or repeatable.

Message layer

Interfaces

You can write a Java application to be a e-Biz Impact acquisition or delivery AIM that implements AcqInterface and DelInterface. When you have a Java application that functions as an AIM, SfmInterface simulates the function entry points for a remote Store and Forward Manager (SFM) application.

Consumer applications can also implement ArgumentsInterface to build function argument wrappers. Argument wrappers allow you to define output or input-output Java arguments with simple immutable or primitive data types.

DfcMessage and *CncMessage*

DfcMessage and CncMessage are the basic containers for the definition of a DFC or a CNC request. Users do not directly interact with these two containers.

DfcMessage requires properties such as function name, timeout, and flavor. A DfcMessage can have one or many arguments.

CncMessage requires command properties such as command name, object name, and subject name. CncMessage can have one or many arguments

Arguments

An argument is a polymorphic object containing a byte array that can be converted to basic primitive data types.

An argument has a default audiotape that allows for the serialization and de-serialization of its content. An argument audiotape can be:

- an array of char
- an array of int
- an array of short
- int
- unsigned int
- char
- unsigned char
- short
- unsigned short
- long
- float
- double
- string
- blob

- unsigned long
- map of (string, string)

To set the argument payload, use the setData( ) method.

To define an unsigned argument, use setUnsigned(true) prior to using setData( ). By default, the argument is signed.

Note In Java, all data types are signed, unlike C, C++, and ODL applications.

To serialize an argument, use the getAsBytes( ) function.

To display all known argument information, use printInfo( ).

CHAPTER 2

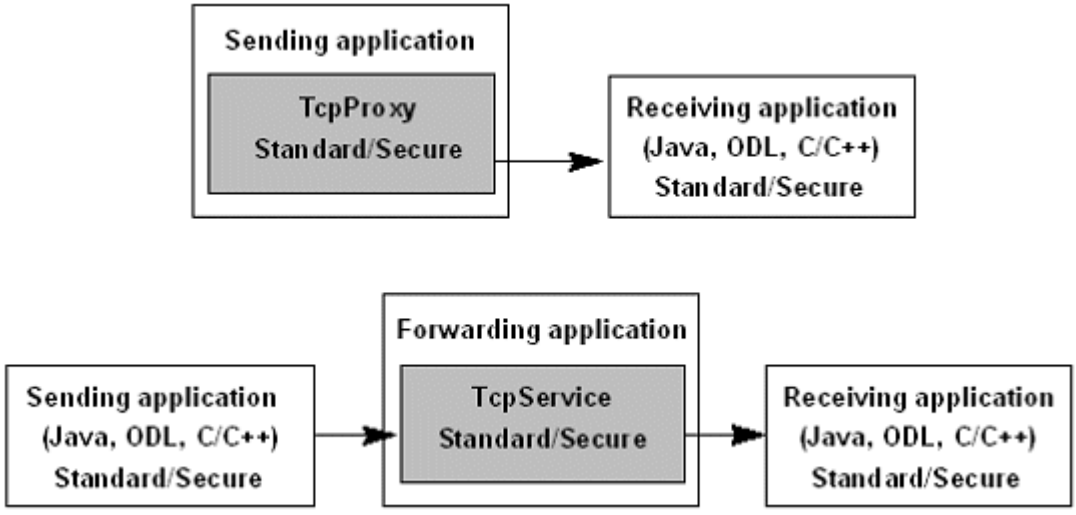
Secure TCP Socket Connections

This chapter explains how to implement secure or standard (plaintext) TCP socket connections in an e-Biz Impact implementation.

Introduction

e-Biz Impact provides secure or standard (plaintext) socket communication using support classes that are included in the com.sybase.impact.tcp package. Interaction with secure or standard sockets is performed using the TcpProxy and TcpService classes.

Figure 2-1: Secure and standard socket interaction



TcpProxy is a helper class that supports a secure or standard TCP socket connection. Once connected, an application can send and receive data using TcpProxy send() and receive() methods.

Internally, TcpService starts a listener that accepts secure or standard TCP connections from a client and routes all data to a secure or standard target connection. Externally, TcpService provides redirection that is intended to redirect secure connection data to a plaintext connection or vice-versa.

Typically, you have an application redirect secure connection data to a plaintext connection where a handler, such as an ODL socket AIM, uses the plaintext data. Secure-to-plaintext and plaintext-to-secure functionality allows ODL AIMs to operate either as servers or as clients in protocol implementations.

Securing socket connections with TcpProxy or TcpService requires that you use keystores and truststores. See the references at the end of this chapter for additional information on creation and maintenance of key stores and trust stores.

Standard JavaDocs are provided for TcpProxy and TcpService.

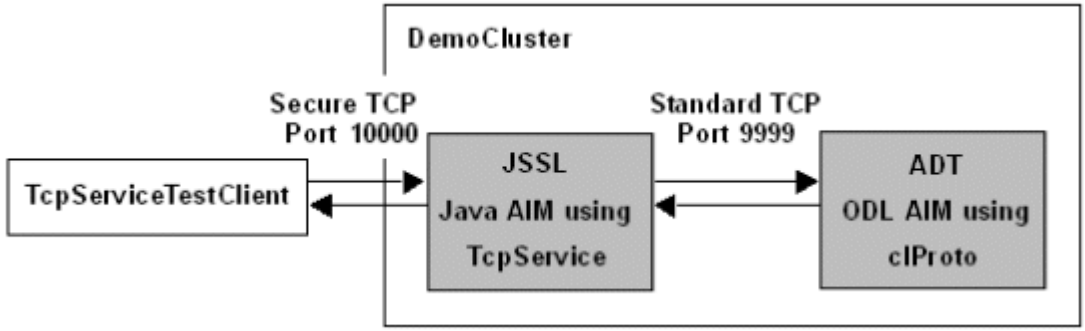
Sample implementations

To illustrate TCP secure socket connection functionality, e-Biz Impact provides *sslPluginSample*, located in *Sybase\ImpactServer-5_4\samples\Java*, which runs a cluster with two AIMs.

- JSSL is a Java AIM that uses TcpService.
- ADT is an standard ODL acquisition AIM.

The sample is provided with a *TcpServiceTestClient.cmd* client.

Figure 2-2: TcpServiceTestClient sample implementation



Code sample – JSSL plug-in

This sample demonstrate how to start TCPService, which forwards data from a secure socket to a non-secure socket running in an ODL AIM.

```
/*
* PluginApplicationSample.java
*
* Created on June 18, 2004, 10:47 AM
*/
package samples.plugin;

import com.sybase.impact.plugin.Application;
import com.sybase.impact.tcp.TcpService;
import com.sybase.impact.tcp.SSLStoreObject;
import com.sybase.impact.tcp.TcpFlowListner;
import com.sybase.impact.tcp.TcpEventListner;

/**
* This class is an example of a Java application that is powered by the e-Biz
* impact runtime engine. Any custom function that is defined public can be
* accessed by the e-Biz Impact bus. This sample demonstrate how to start a
* TcpService which will forward data from a Secured Socket to a nonsecured
* socket running in an ODL aim
*/
public class PluginSslAimSample extends Application implements TcpFlowListner,
     TcpEventListner;
{

/**
* Private Tcp Service instance
```

```
*/
private TcpService m_TCP =null;

/**
* Creates a new instance of <code>PluginApplicationSample.</code>
*/
public PluginSslAimSample() {
     // Create and configure new TCP service
     m_TCP= new TcpService();

     // enable debug
     m_TCP.setDebugEnabled(true);

     // Willing to receive notification of openFlow and closeFlow events
     m_TCP.registerFlowListner(this);

     // Willing to receive notification runtime and debugging events
     m_TCP.registerEventListner(this);
}

/**
* Standard entry points used by Impact bus to initialize the application.
* Implementing this function is optional.
*
* @return Positive or null for success.
*/
public int initialize(){
     erm("Acquisition Aim Initializing");
     int i=0;
     try{

     // Compose keystore used for secured client
     SSLStoreObject oKeyStore = new SSLStoreObject("keystore.txt");
     oKeyStore.setPassword("password",false);

     // Declare trust for clients authorized to connect
     SSLStoreObject oTrustStore = new SSLStoreObject("truststore.txt");
     oTrustStore.setPassword("trustword",false);
     m_TCP.setServerSecurity(oKeyStore,oTrustStore);
     m_TCP.setServerSecurityEnabled(true);

     // Set the server TCP Port to listen for secured client connections.
     // Warning, for secured clients, always use setServerInfo()
     // After setServerSecurity() and setServerSecurityEnabled() , otherwise,
     // The server will start listening for standard connections instead.
     // In this sample, the server listens for secured connections on port 10000
```

```
     // At this time the server will bound its socket on the specified port.
     m_TCP.setServerInfo(10000, 0);

     // This server will forward data to non secured connections on port 9999
     m_TCP.setTargetSecurityEnabled(false);
     m_TCP.setTargetInfo("localhost",9999);

     // Start the TcpService on non blocking mode to continue execution
     //of the application
     m_TCP.setServerBlockingEnabled(false);
     m_TCP.startServer();
     i = 1;

}
catch (Exception e){
     i =0;
     erm(e);
  }
     return i;
}

/**
* Standard entry points uses by the bus to de-initialize the application.
* Implementing this function is optional.
*
* @return Positive or null for success.
*/
public void deinitialize(){
     erm("Acquisition Aim is Deinitializing");
     try{
        // Stop server, because it is still running on non blocking mode
        // also explicitely indicate to release the socket
        m_TCP.stopServer();
        m_TCP.stopListening();
        m_TCP.unregisterFlowListner(this);

        // Release reference to Tcp service instance forces garbage collect
        m_TCP=null;
     }
     catch(Exception e){
        erm(e);
     }
}

/**
* Method inherited from interface TcpFlowListner notifies when
* flows are open with source and target protocol objects
```

```
* @param sFlowName is a unique socket name
* @return True to allow communication to be open,
* False to prevent the communication from been opened
*/
public boolean openFlow(String sFlowName){
     erm("openFlow:" + sFlowName);
     return true;
}

/**
* Method inherited from interface TcpFlowListner notifies when
* flows are closed with source and target protocol objects
* @param sFlowName is a unique socket name
*/
public void closeFlow(String sFlowName){
     erm("closeFlow:" + sFlowName);
}

/**
* Method inherited from interface TcpEventListner notifies
* of the TcpService's runtime events
* @param sDescription sDescription Description of the event
* @param sDescription sDescription Severity of the event,
* ex: INFO, WARNING, ERROR, etc.
*/
public void processEvent(String sDescription, String sSeverity){
     erm(sSeverity + " " + sDescription);
     }
}
```

Code sample – TcpProxy

```
int     serverPort              = 10000;
String  serverHost              = "localhost";
String  sampleData              = new String("AAFirst|Last|M|40|CABB");
String  sTrustStoreFileName     = "truststore.txt";
String  sTrustStorePassword     = "trustword";

// Create new TCP proxy instance
m_tcp = new TcpProxy();

// enable debug
m_tcp.setDebugEnabled(true);

// connect to a secure socket
```

```
m_tcp.connectSecure(serverHost, serverPort, sTrustStoreFileName,
     sTrustStorePassword);
//if (!m_tcp.isConnected())
//    throw new Exception("Failed to connect to target TcpService")

// Prepare a sample data to send
byte[] bDat = sampleData.getBytes();
byte[] bRes = null;

// Sending data
System.out.println("Sending " + bDat.length + " bytes");
m_tcp.send(bDat);

// Receiving data
bRes = m_tcp.receive();
if (bRes != null)
     System.out.println( new String(bRes));
else
bRes=null;
System.out.println("complete.");

// disconnect
m_tcp.disconnect();
m_tcp = null;
```

Code sample – TcpService

```
// Create new TCP service
TcpService TCP= new TcpService();

// enable debug
TCP.setDebugEnabled(true);

// Declare security for connection with secure clients.
// Keystore info need to be changed to custom values.
SSLStoreObject oKeyStore = new SSLStoreObject("keystore.txt");
oKeyStore.setPassword("password", false);

// Declare trust for clients authorized to connect
// truststore info need to be changed to custom values.
SSLStoreObject oTrustStore = new SSLStoreObject("truststore.txt");
oTrustStore.setPassword("trustword",false);

// Set security key and trust for server
TCP.setServerSecurity(oKeyStore,oTrustStore);
```

```
TCP.setServerSecurityEnabled(true);

// Set the server TCP Port to listen for client connections.
// Warning, for secured clients, always use setServerInfo()
// After setServerSecurity() and setServerSecurityEnabled() , otherwise,
// The server will start listening for standard connections instead.
// In this sample, the server will listen for secured connections on port 10000
// At this time the server will bound its socket on the specified port.
TCP.setServerInfo(10000, 0);

// Set the target TCP port to forward data to.
// In this sample the server forwards data to standard connections on port 9999
// Target can be secured using setTargetSecurity(oTrustStore)
TCP.setTargetSecurityEnabled(false);
TCP.setTargetInfo("polaris",9999);

// Start the server on blocking mode (default)
TCP.startServer();

// Stop the server.
TCP.stopServer();
```

More information

For additional information, see:

- The *Java Support for e-Biz Impact* API documentation.
- Sun's documentation:
  - The *Java Secure Socket Extension (JSSE) Reference Guide for the Java 2 SDK, Standard Edition, v 1.4.2* at http://java.sun.com/j2se/1.4.2/docs/guide/security/jsse/JSSERefGuide.html.
  - *JSSE for the Java 2 SDK* at http://java.sun.com/products/jsse/index-14.html.
  - *Creating a Keystore to Use with JSSE* at http://java.sun.com/j2se/1.4.2/docs/guide/security/jsse/JSSERefGuide.html#CreateKeystore.

CHAPTER 3

Java Support Samples

The sample code in this chapter demonstrates how to use the libraries that provide Java support for e-Biz Impact. A simplified class diagram is also included for your reference.

Creating Java applications

Creating a plug-in application

This is a sample of a Java application powered by the e-Biz Impact runtime engine. Any custom function that is defined as public can be accessed by the e-Biz Impact bus. This sample implements DelAimInterface that defines the servproc and servayt functions, called by SFM to deliver transactions to an end point.

```
/*
* PluginApplicationSample.java
*
* Created June 18, 2004, 10:47 AM
*/

package samples.plugin;

import com.sybase.impact.messaging.*;
import com.sybase.impact.plugin.*;
```

```
/**
* This class is an example of a Java application that is powered by the e-Biz
* Impact runtime engine. Any custom function that is defined public can be
* accessed by the e-Biz Impact bus. This sample implements DelAimInterface that
* defines the servproc and servayt functions, called by sfm for delivering
* transactions to an end point.
*
* @author hlebegue
*
* @version 1.0
*
* @since 1.0
*/
public class PluginDelAimSample extends Application implements DelAimInterface{

/**
* Creates a new instance of <code>PluginDelAimSample.</code>
*/
public PluginDelAimSample() {
}

/**
* Standard entry points used by the bus to initialize the application.
* Implementing this function is optional.
*
* @return Positive or null for success.
*/
public int initialize(){
    erm("Delivery Aim is Initialized");
    return 1;
}

/**
* Standard entry points uses by the bus to Initialize the application.
* Implementing this function is optional.
*
* @return Positive or null for success.
*/
public void deinitialize(){
    erm("Delivery Aim is Deinitialized");
}

/**
* Called by an SFM application. A consumer is expected to return 1
* if ready; otherwise negative or null. A consumer is also expected to set up
* the last transaction ID into <code>sAytStat</code> to inform SFM of the last
```

```
* known good transaction serial number.
*
* @param oArguments A server argument wrapper that contains all the function
* arguments.
*
* @return Value greater that one if success; otherwise null or negative.
*/
public int servayt(ServAytArguments, Arguments) {
    erm("\n" + Arguments.toString());
    return 1;
}

/**
* Called by an SFM application and correspond to the server processing a
* transaction. The consumer is expected to return positive for success, -1 for
* failure, request a new retry, or -999 to cancel this transaction and move on
* to the next. The consumer is expected to dispatch the transaction data
* buffer and if it succeeds, save the transaction serial number into a file to
* allow <code>servayt</code> to read the number.
*
* @param oArguments A <code>servproc</code> argument wrapper that contains all
* the function arguments.
*
* @return Value greater that one if success; otherwise null or negative for
* failure. Special consideration on -999 for cancel or -998 for skip.
*/
public int servproc(ServProcArguments oArguments) {
    erm("\n" + oArguments.toString());
    Arguments.errorText="kjkjbfds";
    return 1;
    }
}
```

Creating a Java application using the Configurator

1 Create Java functions by adding this code to your application:

```
/** Called by ODL and receives a custom argument wrapper.
* It <code>erms</code> out the field value.
*/
public int myJavaFunctionCalled(PluginSampleArgumentsWrapper Args){
    super.erm("Function myJavaFunctionCalled received: " + Args.field);
    return 1;
}
```

2 Start the Configurator.

3 Right-click a Controller an select New | Java from the menu.

4 Double-click the new Java icon in the right pane. When the Java Properties window displays, select the General tab, then select Debug and Active. Leave the Tracking option unselected.

5 Complete the remaining options on the General tab:

- Name – `JAVA`
- Working Directory – enter a dot (.) for the current directory.
- Plugin Class Name – `samples.plugin.PluginDelAimSample`
- Command Line – leave this field blank.
- Log Limit – `2048`
- Depth – `1`
- Class Path items – enter "`../classes`" in the Name field, then click Add.
- Instances – enter `1` for Minimum and Maximum.
- Custom Keys – leave this option blank.

Click Apply.

6 Select the DFC tab in the Java Properties window and add these two functions:

1 servproc

- Name – `servproc`
- Flavor – `6`
- Timeout – `10`

Click Add.

2 servayt

- Name – `servayt`
- Flavor – `6`
- Timeout – `10`

7 Click Add, then click OK.

Any e-Biz Impact application can call the added functions.

Using *DfcProxy* and *DfcSfmProxy* for remote access

Creating *DfcProxy*

```
DfcProxy impactClient = new DfcProxy();
```

Connecting to e-Biz Impact

```
impactClient.connect("localhost" /*host or ip adress*/ , 1234 /*tcp port*/);
```

Checking connection status

```
if (impactClient.isConnected()){
   System.out.println("Connected and ready for Dfc");
}
```

Making a distributed function call

```
byte[] bData = "Bart|Simpson|M|8|IL".getBytes();

// One possible way using the Sfm proxy
iRet = impactClient.route_veng("sourcename", "Eng1", bData);

// Another way creating an argument wrapper
RouteArguments argWrapper = new RouteArguments ();
argWrapper.source="AcqAimSample";
argWrapper.route="ENGINE";
argWrapper.data=bData;
iRet = impactClient.sendDfc("route_veng",50,30,argWrapper);

// Can also use the RouteRecxArguments wrapper for advanced options
// on the route call

// Printing result
System.out.println("   AcqAimSample::Send transaction(2) return " + iRet);

// More function through the SfmDfcProxy
// Other ways to send data to an sfm using the DfcSfmProxy
int iRet = impactClient.route_recx("source1",
   "ENGINE",
   new String("Bart|Simpson|M|8|IL").getBytes(),
```

```
   256,
   (char)20,
   "serialkey123");
int iRet = impactClient.route_vrec("source1",
   "ENGINE",
   new String("Bart|Simpson|M|8|IL").getBytes(),
   );
int iRet = impactClient.route_veng("source1",
   "ENGINE",
   new String("Bart|Simpson|M|8|IL").getBytes(),
   );
int iRet = impactClient.route_vprod("source1",
   "prod1",
   new String("Bart|Simpson|M|8|IL").getBytes(),
   );
```

Disconnecting from e-Biz Impact

```
impactClient.disconnect();
```

Using arguments

```
// This sample prints information about an Argument.

public void PrintArgumentInfo(Argument lArg) {
   try{
      System.out.println("Arg Name:" + lArg.getName());
      int iType = lArg.getType();
      int iMode = lArg.getMode();
      int iSize = lArg.getSize();
      String sType = lArg.getTypeAsString();

      System.out.println("  Arg Type:" + iType + " (" +  sType +")");
      System.out.println("  Arg Mode:" + iMode + " (" + lArg.getModeAsString()
         +")" );
      System.out.println("  Arg Size:" + lArg.getSize());

      if (iType == eArgType.Arg_Int)
         System.out.println("  Arg Value[int]:" + lArg.toInt() + "\n");
      else if (iType == eArgType.Arg_Blob){
         if (iSize < 1000){
            System.out.println("  Arg Value[blob]:" + new String
```

```
                (lArg.toByteArray()) + "\n");
            }
         }
         else if (iType == eArgType.Arg_Char)
            System.out.println("  Arg Value[char]:" + lArg.toChar()+ "\n");
         else if (iType == eArgType.Arg_Short)
            System.out.println("  Arg Value[short]:" + lArg.toShort()+ "\n");
         else if (iType == eArgType.Arg_String){
            if (iSize < 1000){
               System.out.println("  Arg Value[String]:" + lArg.toString()+ "\n");
            }
         }
         else if (iType == eArgType.Arg_Double)
            System.out.println("  Arg Value[Double]:" + lArg.toDouble()+ "\n");
         else if (iType == eArgType.Arg_Array){
            try{
            System.out.println("  Array of type:" + lArg.getArrayType() + " (" +
lArg.getArrayTypeAsString() +")" );
            System.out.println("  Array Size:" + lArg.getArraySize());
            if(lArg.getArrayType() == eArgType.Arg_Char){
               char [] s = lArg.toCharArray();
               System.out.print("    Arg Value[char array]:");
               for (int k=0; k< lArg.getArraySize();k++)
                  System.out.print("\n      a[" + k + "]='" + s[k] + "'");
            System.out.println("\n");
         }
         else if(lArg.getArrayType() == eArgType.Arg_Short){
            short[] s = lArg.toShortArray();
            System.out.print("    Arg Value[short array]:");
            for (int k=0; k< lArg.getArraySize();k++)
               System.out.print("\n      a[" + k + "]='" + s[k] + "'");
            System.out.println("\n");
         }
         else if(lArg.getArrayType() == eArgType.Arg_Int){
            int[] s = lArg.toIntArray();
            System.out.print("    Arg Value[int array]:");
            for (int k=0; k< lArg.getArraySize();k++)
               System.out.print("\n      a[" + k + "]='" + s[k] + "'");
            System.out.println("\n");
         }
         else if(lArg.getArrayType() == eArgType.Arg_Double){
            double[] s = lArg.toDoubleArray();
            System.out.print("    Arg Value[double array]:");
            for (int k=0; k< lArg.getArraySize();k++)
               System.out.print("\n      a[" + k + "]='" + s[k] + "'");
            System.out.println("\n");
```

```
      }
      else if(lArg.getArrayType() == eArgType.Arg_Unset){
         System.out.println("    Arg Value[Unset]: Empty array"+ "\n");
      }
      else{
         System.out.println("    Arg Value[Not implemented]:" +
lArg.toString()+ "\n");
         }
      }
         catch(Exception e){e.printStackTrace();
      }
   }
      else if (iType == eArgType.Arg_Unset){
         System.out.println("  No Arg Value (Arg_Unset)");
      }
      else
         System.out.println("  No Arg Value (Not implemented)");
      //System.out.println("  Binary dump:" + lArg.printAsBinary());
   }
   catch(Exception e){
      System.out.println("  Can not display arguments");
      e.printStackTrace();
   }
}
```

Using *DfcService with remote access*

DfcService controls one or more applications. The samples in this section illustrate how to:

- Build a Java acquisition AIM that acquires transactions from a source endpoint and sends the transactions to a SFM in e-Biz Impact.
- Build a Java delivery AIM that receives transactions from an e-Biz Impact SFM and delivers the transactions to a destination endpoint application.

Creating a Java application for data acquisition

```
/** An acquistion AIM acquires data and makes a distributed function call into
a remote application (here an SFM) using the Java DfcSfnmProxy.

 * This application implements the AcqAimInterface, which defines the default
```

entry points of the application (initialize, deinitialize, ping). When the SFM references this application as one of its sources, the SFM periodically calls the ping function. Here the ping function sends a transaction back to the SFM

```
 * @see AcqAimInterface
 * @see DfcSfmProxy
 * @author  hlebegue
 * @since 1.0
 * @version 1.0
 */
public class AcqAimSample implements AcqAimInterface {

/** DfcSfmProxy allows the aim to deliver data to a remote SFM*/
private DfcSfmProxy m_Sfm=null;

/** Remote host when Impact 5.x is running*/
private String m_sRemoteImpactHost="localhost";

/** Remote Tcp port on which Impact 5.x is accepting dfc*/
private int m_sRemoteImpactTcpPort=2255;

/** Creates a new instance
 * The aims creates a new Sfm Proxy
 */
public AcqAimSample() {
    m_Sfm = new DfcSfmProxy(50);
}
/** This function is inherited from the AcqAimInterface
 * The aims connects to Impact using the Sfm Proxy
 * @see AcqAimInterface
 */
public int initialize() {
    System.out.println("-->AcqAimSample::Called on initialize()");
    try{
        m_Sfm.connect(m_sRemoteImpactHost, m_sRemoteImpactTcpPort);
    }
    catch (Exception e){}
return 1;
}

/** This function is inherited from the AcqAimInterface
 * The aims disconnect from Impact using the Sfm Proxy
 * @see AcqAimInterface
 */
public void deinitialize() {
    System.out.println("-->AcqAimSample::Called on deinitialize()");
    if (m_Sfm.isConnected()){
        try{
```

```
            m_Sfm.disconnect();
        }
        catch(Exception e){}
    }
}

/** This function is inherited from the AcqAimInterface
 * For an acq application, SFM will periodically call this method in order
 * to identify the status of the source
 * @return negative to indicate that source application is not ready
 * @see AcqAimInterface
 */
public int ping(int iAimFlavor, int iCallerFlavor) {
    System.out.println("-->AcqAimSample::Called on ping()");
    System.out.println("   Flavor:" + iAimFlavor + " CallerFlav:"+
iCallerFlavor);

    // Any logic to determine if sending application is ready here,
    // eventually reattempt to connect if not.
    int iRet=-1;
    if (!m_Sfm.isConnected()){
        try{
            m_Sfm.connect(m_sRemoteImpactHost, m_sRemoteImpactTcpPort);
        }
        catch (Exception e){}
    }

    // Here, for this sample, we will send a transaction to
    // Sfm every time Sfm pings the source
    iRet = sendToSfmEngine("Bart|Simpson|M|8|IL".getBytes());

    return iRet;
}

/** This function is user defined. Consumer application calls it to send data
 * to Sfm @param bData The transaction data
 */
private int sendToSfmEngine(byte [] bData){
    int iRet=-1;
    if (m_Sfm.isConnected()){
        try{

            // one way to send a transaction to Sfm
            iRet = m_Sfm.route_veng("me_myself_and_i", "Eng1", bData);
            System.out.println("AcqAimSample::Send transaction(1) rc=" +
iRet);
```

```
            // another way using an argument wrapper
            RouteVRecArguments argWrapper = new RouteVRecArguments ();
            argWrapper.source="AcqAimSample";
            argWrapper.route="ENGINE";
            argWrapper.data=bData;
            iRet = m_Sfm.sendDfc("route_veng",50,30,argWrapper);
            System.out.println("AcqAimSample::Send transaction(2) rc=" +
iRet);
            argWrapper =null;

        }
        catch (Exception e){}
    }
    return iRet;
}
```

Creating a Java application for data delivery

```
/** A delivery AIM receives a transaction from a SFM and dispatches the data to
a receiving end point.
 * This application implements the DelAcqAimInterface which defines the default
entry points of the application (initialize, deinitialize, servproc and
servayt).
 * @see DelAimInterface
 * @see DfcSfmProxy
 * @since 1.0
 * @version 1.0
 */
public class DelAimSample implements DelAimInterface {

/** Utility variable to store last transaction serial number received by SFM*/
private String m_sLastSerial="";\

/** Utility variable used for sucessfull return value*/
private int iRet = 1;

/** Creates a new instance */
public DelAimSample() {
}

/** This function is inherited from the DelAimInterface
* The delivery application is given a chance to initialize the connection to
an end point for delivery
```

```
 * @return Positive for success
 * @see DelAimInterface
 */
public int initialize() {
System.out.println("-->DelAimSample::Called on initialize()");
return 1;
}

/** This function is inherited from the DelAimInterface
 * SFM will call this function in order to check if the application is ready to
deliver data.
 * User code is responsible for setting up the value of the outpout argument
lastSerial to the last known good transaction serial number
 * @return Positive for ready to deliver, else negative or zero.
 * @see DelAimInterface
 */

public int servayt(ServAytArguments oArguments) {
   System.out.println("-->DelAimSample::Called on servayt()");
   System.out.println("   oArgs.flavor     ="  + oArguments.flavor);
   System.out.println("   oArgs.lastSerial ="  + oArguments.lastSerial);

// More elegant solutions will look for the serial number into a lastID file or
repository.
oArguments.lastSerial=m_sLastSerial;
return iRet;
}

/** This function is inherited from the DelInterface
 * This function is called by Sfm to deliver the transaction. The
ServProcArguments wrapper contains all of the specific information for the
transaction.
 * @return Positive for sucess, -1 for failure, -999 for Cancel transaction upon
return, -998 for Skip transaction upon return
 * @see DelAimInterface
 */
public int servproc(ServProcArguments oArguments){
   System.out.println("-->DelAimSample::Called on servproc()");
   System.out.println("   oArgs.delFlavor  ="  + oArguments.delFlavor);
   System.out.println("   oArgs.serial     ="  + oArguments.serial);
   System.out.println("   oArgs.source     ="  + oArguments.source);
   System.out.println("   oArgs.data       ="  + new String(oArguments.data));
   System.out.println("   oArgs.route      ="  + oArguments.route);
   System.out.println("   oArgs.errorText  ="  + oArguments.errorText);
   System.out.println("   oArgs.fkey       ="  + oArguments.fkey);
   System.out.println("   oArgs.options    ="  + oArguments.options);
```

```
   System.out.println("   oArgs.priority   ="  + oArguments.priority);
   System.out.println("   oArgs.sfmFlavor  ="  + oArguments.sfmFlavor);
   m_sLastSerial=oArguments.serial;
   oArguments.errorText="Hello from the delivery app!";
   return iRet;
}

/** This function is inherited from the DelAimInterface
 * The delivery application is given a chance to close the connection with an
end point application
 * @see DelAimInterface
 */
public void deinitialize() {
   System.out.println("-->DelAimSample::Called on deinitialize()");
   }
}
```

Defining a DfcService

```
DfcService ctrl  = new DfcService(12345 /*TCP port number used to listen
    for incoming functions*/);
```

Adding an application to a controller

```
AcqAimSample acq = new AcqAimSample();
ctrl.addApplication(acq,65 /*flavor used for all public functions
    for AcqAimSample*/);

DelAimSample del = new DelAimSample();
ctrl.addApplication(del,66 /*flavor used for all public functions
    for DelAimSample*/);
```

Starting the service

```
m_ctrl.enable();
```

Shutting down the service

```
ctrl.shutdown();
```

Using *CncProxy* for remote access

Creating a *CncProxy*

```
CncProxy rC = new CncProxy ("localhost"/*host or ip*/, 1234/*manager port
number*/);
```

Logging in

```
if (!rC.login("system","manager")){System.out.println("User is not
   authenticated! ");
return;
}
```

Sending a command

```
String sUnprocTxList = rC.getSfmTxList("SFM_LXB",0, 1,50);
int iStatus = rC.getControllerStatus("Controller2","Cluster2");
```

Getting error and return code information

```
Int iErrCode = rC.getErrorNo();
Int iRetCode = rC.getRetCode ();

String sErrText = rC.getErrorString());
```

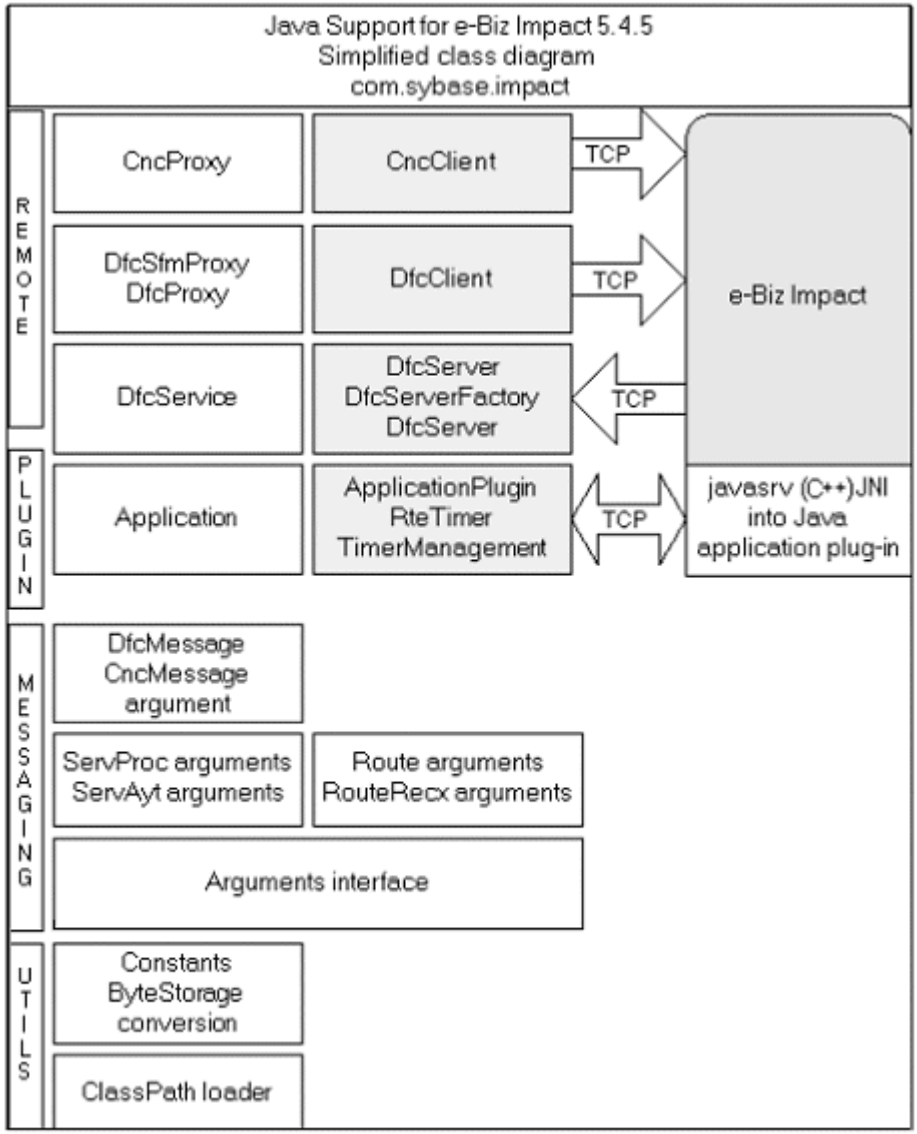
Logging out

```
if (rC.isLoggedIn()){
    System.out.println("Logging out");
    rC.logout();
}
```

Class diagram

Figure 3-1is a simplified class diagram of e-Biz Impact Java support. The diagram illustrates the three main packages—remote, plug-in, and messaging. Both the remote and plug-in packages use the messaging package.

Figure 3-1: Java support class diagram



CHAPTER 4

Web Services Templates

This chapter provides two sample templates that allow you to create a bridge from e-Biz Impact to a SOAP service.

SOAP client bridge sample

This example illustrates how to use a Java-based DFC/SOAP bridge that runs as an e-Biz Impact plug-in to a create a bridge from an ODL DFC client to a SOAP-based service. The ODL client calls the bridge through DFC links, and the bridge in turn calls the SOAP service.

Four different DFC functions are implemented within the bridge and defined within the cluster:

- soapHello1 – demonstrates the ArgumentsInterface method of DFC argument handling.
- soapHello2 – demonstrates the DfcMessage method of DFC argument handling in conjunction with static SOAP stub bindings.
- soapHello3 – demonstrates the DfcMessage method of DFC argument handling in conjunction with dynamic SOAP bindings.
- soapHello4 – defined as a valid DFC in a cluster, but unimplemented in the bridge plug-in.

The files used for this sample are located in *Sybase\ImpactServer-5_4\samples\Java\soapClientBridge*, and listed in Table 4-1.

Table 4-1: SOAP client bridge sample files

File	Description
_clean.bat	Cleans up and removes log files
_readme.txt	Duplicates the instructions in this section

File	Description
_readmeEAS500.txt	Duplicates the procedures for running the SOAP client bridge sample with Sybase EAServer 5.0
_start.cmd	Sets up the Java runtime path, CLASSPATH, and runs the sample cluster
HelloArgsWrapper.java	DFC argument handling support for *SoapHelloBridge.java*
HelloBridgeEAS500.jar	A complete e-Biz Impact Java plug-in to use with EAServer 5.0
HelloClient.odl	ODL-based DFC client that calls the bridge
odl2soap.bridge.cfg	e-Biz Impact application configuration for the Java-based DFC/SOAP bridge
odl2soap.client.cfg	e-Biz Impact application configuration for the ODL-based DFC client
odl2soap.xml	Primary e-Biz Impact cluster configuration file for this sample
SoapHello.java	A simple Java class to deploy a Web service
SoapHelloBridge.java	e-Biz Impact plug-in that operates as a DFC/SOAP bridge

Requirements

To use the e-Biz Impact SOAP client bridge sample, you must have the following installed on your system:

- e-Biz Impact 5.4.5
- J2SE 1.4.2_05 or later
- A SOAP engine, such as Sybase EAServer 5.0 or later

Running the sample

❖ **Using the SOAP client bridge sample**

This sample uses e-Biz Impact and EAServer 5.0 or later with Sybase Web Services Toolkit and Eclipse.

Note If you are using EAServer 5.0., skip steps 1 through 3 and use *HelloBridgeEAS500.jar*.

1 Install and set up EAServer.

a Install EAServer 5.0 or later. See the EAServer installation guide for your platform at http://sybooks/onlinebooks/group-eag/.

Note Eclipse is installed as part of the standard EAServer installation when you install Web Services. To run Eclipse, you must have a complete JDK installation (*jdk1.4* or later), which is not installed as part of the standard EAServer installation.

b Start EAServer.

c Start Eclipse and connect to EAServer.

UNIX From the command line in the *Shared/eclipse* subdirectory, enter:

```
./starteclipse.sh
```

Windows From the command line in the *Shared\eclipse* subdirectory, enter:

```
starteclipse.bat
```

or select Start | Run | Programs | Sybase | EAServer 5.0.0 | Eclipse.

d In Eclipse, select Window | Show View | Other.

e In the Show View window, expand the Sybase Web Services folder, select Sybase Web Services in the tree, and click OK.

"Sybase Web Services Servers" displays in the Eclipse left pane with a server profile beneath it.

f Right-click the server profile (for example, "Default on <*localhost*>:*8080*") and select Connect.

Eclipse is now connected to the EAServer SOAP engine.

2 Compile and deploy the *SoapHello.java* Web service.

a Select File | New | Project.

b Select Sybase Web Services and click Next.

c Enter the project name `SoapHelloService` and click Finish.

d Select the Package Explorer tab below the left pane, then right-click SoapHelloService and select Import.

e Select File System and click Next.

f Use Browse to navigate to the *soapClientBridge* directory (for example, *\ImpactServer-5_4\samples\Java\soapClientBridge*), and click OK.

g Select *SoapHello.java* and click Finish.

h From the Package Explorer, expand the SoapHelloService tree until you see *SoapHello.java*,

i Right-click *SoapHello.java* and select Quickly Deploy As Web Service.

j You see a message that the Web service *ws/SoapHello* has been successfully created and deployed. Click OK.

3 Create SOAP client stubs and artifacts.

a Create a new project to hold artifacts and Impact DFC/SOAP bridge code.

1 Select File | New | Project.

2 Select Sybase Web Services and click Next.

3 Enter the project name `SoapHelloBridge` and click Finish.

b Generate client artifacts.

1 Click the Sybase Web Services tab at the bottom of the Eclipse upper-left pane.

2 Expand the Web Services icon.

3 Expand the WS icon.

4 Right-click SoapHello and select Create Web Services Client.

5 Select SoapHelloBridge and click Next.

6 Leave the package name blank and click Next twice.

7 Click Finish.

8 You see a message that the generation was successful. Click OK.

The SoapHelloBridge project now contains all required SOAP client stubs and artifacts for the SoapHello service.

4 Compile the DFC/SOAP bridge plug-in.

a Select the Package Explorer tab below the left pane, then select SoapHelloBridge.

b Select Project | Rebuild All to compile the bridge.

c Right-click SoapHelloBridge and select Import.

d Select File system and click Next.

e Use Browse and navigate to the *soapClientBridge* directory (for example, *ImpactServer-5_4\samples\Java\soapClientBridge*) and click OK.

f Select *SoapHelloBridge.java* and *HelloArgsWrapper.java* and click Finish.

g Right-click SoapHelloBridge and select Properties.

h In the Properties window, select Java Build Path in the left pane, then select the Libraries tab in the right pane.

i Click Add External JARs.

j Navigate to the e-Biz Impact Java runtime directory (for example, *\ImpactServer-5_4\classes*), select *ims54rt.jar*, and click Open.

k Click OK.

l Select Project | Rebuild All to compile the bridge.

The e-Biz Impact bridge code and the SOAP client stubs are now compiled.

5 Package the bridge plug-in.

a Right-click the default package icon beneath SoapHelloBridge and select Export.

b Select JAR file and click Next.

c Select the export destination. Use Browse to navigate to the *soapClientBridge* directory (for example, *\ImpactServer-5_4\samples\Java\soapClientBridge*).

d Select `HelloBridgeEAS500.jar` for the File Name and click Save. If you are prompted to save over an existing file, answer yes and continue.

e Click Finish in the JAR Export dialog box.

6 Select File | Exit to close Eclipse.

7 In a terminal window or at a command line, enter `_start.cmd` from the *soapClientBridge* directory.

8 Inspect the logs *odl2soap.client.xlog1*, *odl2soap.bridge.xlog1*, and the SOAP engine log for success indications.

Typical output for *odl2soap.client.xlog1*

```
2004-09-09 12:12:31.453 | ims54cluster | 4012 | 4052 | client.1 |
   :clinit: hello
2004-09-09 12:12:31.453 | ims54cluster | 4012 | 4052 | client.1 |
   :callSoapHello: call soapHello1 with: Hello from ODL.
2004-09-09 12:12:32.828 | ims54cluster | 4012 | 4052 | client.1 |
   :callSoapHello: soapHello returns: 1, Hello from SOAP service!
2004-09-09 12:12:32.828 | ims54cluster | 4012 | 4052 | client.1 |
   :callSoapHello: call soapHello2 with: Hello from ODL.
2004-09-09 12:12:32.875 | ims54cluster | 4012 | 4052 | client.1 |
   :callSoapHello: soapHello returns: 1, Hello from SOAP service!
2004-09-09 12:12:32.875 | ims54cluster | 4012 | 4052 | client.1 |
   :callSoapHello: call soapHello3 with: Hello from ODL.
2004-09-09 12:12:32.890 | ims54cluster | 4012 | 4052 | client.1 |
   :callSoapHello: soapHello returns: 1, Hello from SOAP service!
2004-09-09 12:12:32.890 | ims54cluster | 4012 | 4052 | client.1 |
   :callSoapHello: call soapHello4 with: Hello from ODL.
2004-09-09 12:12:32.906 | ims54cluster | 4012 | 4052 | client.1 |
   :callSoapHello: soapHello DFC error:
2004-09-09 12:12:40.140 | ims54cluster | 4012 | 4052 | client.1 |
   IpcRequest()received deinit message
2004-09-09 12:12:40.140 | ims54cluster | 4012 | 4052 | client.1 |
   :cldeinit:good-bye
```

Typical output from *odl2soap.bridge.xlog1*

```
2004-09-09 12:12:31.625 | ims54cluster | 4012 | 4008 | bridge.1 |
   :SoapHelloBridge:initialize
2004-09-09 12:12:31.640 | ims54cluster | 4012 | 4008 | bridge.1 |
   : soapHello called with argument: Hello from ODL.
2004-09-09 12:12:32.828 | ims54cluster | 4012 | 4008 | bridge.1 |
   : hello returns: Hello from SOAP service!
2004-09-09 12:12:32.828 | ims54cluster | 4012 | 4008 | bridge.1 |
   : soapHello returning argument: Hello from SOAP service!

2004-09-09 12:12:32.843 | ims54cluster | 4012 | 4008 | bridge.1 | : process
called...

2004-09-09 12:12:32.843 | ims54cluster | 4012 | 4008 | bridge.1 | : Function
Name:soapHello2Function Flavor:0Function Timeout:0sMessage Type: requestArg
Name:Arg0  Arg Mode: 3 (Input, Output)  Arg Type: 10 (String)  Arg Size: 15  Arg
```

```
Value: Hello from ODL.

2004-09-09 12:12:32.859 | ims54cluster | 4012 | 4008 | bridge.1 | : hello
returns: Hello from SOAP service!

2004-09-09 12:12:32.875 | ims54cluster | 4012 | 4008 | bridge.1 | : process
returning...2004-09-09 12:12:32.875 | ims54cluster | 4012 | 4008 | bridge.1 |
: Function Name:soapHello2Function Flavor:0Function Timeout:0sMessage Type:
responseArg Name:Arg0  Arg Mode: 3 (Input, Output)  Arg Type: 10 (String)  Arg
Size: 24  Arg Value: Hello from SOAP service!Arg Name:ReturnArg  Arg Mode: 3
(Input, Output)  Arg Type: 3 (Int)  Arg Size: 1  Arg Value: 1

2004-09-09 12:12:32.875 | ims54cluster | 4012 | 4008 | bridge.1 | : process
called...

2004-09-09 12:12:32.875 | ims54cluster | 4012 | 4008 | bridge.1 | : Function
Name:soapHello3Function Flavor:0Function Timeout:0sMessage Type: requestArg
Name:Arg0  Arg Mode: 3 (Input, Output)  Arg Type: 10 (String)  Arg Size: 15  Arg
Value: Hello from ODL.

2004-09-09 12:12:32.875 | ims54cluster | 4012 | 4008 | bridge.1 | : Connecting
to: http://gradyxp:8080/ws/services/SoapHello

2004-09-09 12:12:32.890 | ims54cluster | 4012 | 4008 | bridge.1 | : hello
returns: Hello from SOAP service!

2004-09-09 12:12:32.890 | ims54cluster | 4012 | 4008 | bridge.1 | : process
returning...

2004-09-09 12:12:32.890 | ims54cluster | 4012 | 4008 | bridge.1 | : Function
Name:soapHello3Function Flavor:0Function Timeout:0sMessage Type: responseArg
Name:Arg0  Arg Mode: 3 (Input, Output)  Arg Type: 10 (String)  Arg Size: 24  Arg
Value: Hello from SOAP service!Arg Name:ReturnArg  Arg Mode: 3 (Input, Output)
Arg Type: 3 (Int)  Arg Size: 1  Arg Value: 1

2004-09-09 12:12:32.906 | ims54cluster | 4012 | 4008 | bridge.1 | : process
called...

2004-09-09 12:12:32.906 | ims54cluster | 4012 | 4008 | bridge.1 | : Function
Name:soapHello4Function Flavor:0Function Timeout:0sMessage Type: requestArg
Name:Arg0  Arg Mode: 3 (Input, Output)  Arg Type: 10 (String)  Arg Size: 15  Arg
Value: Hello from ODL.

2004-09-09 12:12:32.906 | ims54cluster | 4012 | 4008 | bridge.1 | : process:
unsupported DFC: soapHello4
```

```
2004-09-09 12:12:32.906 | ims54cluster | 4012 | 4008 |
bridge.1 | : process returning...

2004-09-09 12:12:32.906 | ims54cluster | 4012 | 4008 |
bridge.1 | : Function Name:soapHello4Function Flavor:0Function
Timeout:0sMessage Type: responseArg Name:Arg0  Arg Mode: 3
(Input, Output)  Arg Type: 10 (String)  Arg Size: 0  Arg Value: Arg
Name:ReturnArg  Arg Mode: 3 (Input, Output)  Arg Type: 3 (Int)  Arg Size: 1  Arg
Value: -9999

2004-09-09 12:12:40.140 | ims54cluster | 4012 | 4008 | bridge.1 | IpcRequest()
received deinit message

2004-09-09 12:12:40.140 | ims54cluster | 4012 | 4008 | bridge.1 | :
SoapHelloBridge:deinitialize

2004-09-09 12:12:40.140 | ims54cluster | 4012 | 3192 | bridge.1 | Finalize()
In plugin!!!

2004-09-09 12:12:40.140 | ims54cluster | 4012 | 3192 | bridge.1 | Finalize()
Done
```

Typical output from *Sybase\EAServer\bin\Jaguar.log*

```
Sep 09 12:12:32 2004: SoapHello:Hello() called with input: Hello from ODL.

Sep 09 12:12:32 2004: SoapHello:Hello() returning value: Hello from SOAP
service!

Sep 09 12:12:32 2004: SoapHello:Hello() called with input: Hello from ODL.

Sep 09 12:12:32 2004: SoapHello:Hello() returning value: Hello from SOAP
service!

Sep 09 12:12:32 2004: SoapHello:Hello() called with input: Hello from ODL.

Sep 09 12:12:32 2004: SoapHello:Hello() returning value: Hello from SOAP
service!
```

SOAP service bridge sample

This example shows how to access an e-Biz Impact service using a Java-based SOAP/DFC bridge that runs as a SOAP service in a SOAP engine from a SOAP client.

The files used for this sample are located in *\ImpactServer-5_4\samples\Java\soapServiceBridge*, and listed in Table 4-2.

Table 4-2: SOAP service bridge sample files

File	Description
_clean.bat	Cleans up and removes log files
_readme.txt	Duplicates the instructions in this section
_readmeEAS500.txt	Duplicates the procedures for running the SOAP service bridge sample with Sybase EAServer 5.0
_start.cmd	Sets up the Java runtime path, classpath, and runs the sample cluster
Hello.odl	A simple ODL-based DFC service
ImpactHello.java	Java module that operates as a SOAP/DFC bridge
soap2odl.hello.cfg	e-Biz Impact application configuration for the ODL-based DFC service
soap2odl.xml	The primary e-Biz Impact cluster configuration file for this sample

Requirements

To use the e-Biz Impact SOAP service bridge sample, you must have the following installed on your system:

- e-Biz Impact 5.4.5
- J2SE 1.4.2_05 or later
- A SOAP engine, such as Sybase EAServer 5.0 or later

Running the sample

❖ **Using the SOAP service bridge sample**

This sample uses e-Biz Impact 5.4.5 and EAServer 5.0 or later with Sybase Web Services Toolkit and Eclipse.

Note Eclipse is installed as part of the standard EAServer installation when you install Web Services. To run Eclipse, you must have a complete JDK installation (*jdk1.4* or later), which is not installed as part of the standard EAServer installation.

---

1 Start and set up EAServer.

a Install EAServer 5.0 or later. See the EAServer installation guide for your platform at http://sybooks/onlinebooks/group-eag/.

b In a text editor, modify *\Sybase\EAServer\bin\setenv.bat* to point to JRE 1.4.2_05 or later; for example:

```
REM set JAGUAR_JDK14=D:\Sybase\Shared\jdk1.4.1_03
```

c Set the variable `JAGUAR_JDK14=D:\Java\j2sdk1.4.2_05`.

d In a text editor, modify *Sybase\eclipse\starteclipse.bat* to point to JRE 1.4.2_05 or later; for example:

```
rem eclipse -vm "D:\Sybase\Shared\jdk1.4.1_03\jre\bin\java"
eclipse -vm "D:\Java\j2sdk1.4.2_05\jre\bin\java"
```

e Start EAServer.

2 Deploy *ImpactHello.java* as a Web service using the SOAP engine:

a Start Eclipse and connect to EAServer.

UNIX From the command line in the *Shared/eclipse* subdirectory, enter:

```
./starteclipse.sh
```

Windows From the command line in the *Shared\eclipse* subdirectory, enter:

```
starteclipse.bat
```

or select Start | Run | Programs | Sybase | EAServer 5.0.0 | Eclipse.

b In Eclipse, select File | New | Project.

c Select Sybase Web Services and click Next.

d Enter the project name `ImpactHello` and click Next.

e Click the Libraries tab, then click Add External JARs.

f Use Browse to navigate to the *\ImpactServer-5_4\classes* directory, select *ims54rt.jar*, then click Open.

g Click Finish.

h Right-click the new ImpactHello icon in the Package Explorer pane and select Import.

i Select File System and click Next.

j Use Browse to navigate to the *\ImpactServer-5_4\samples\Java* directory, highlight *soapServiceBridge*, and click OK.

k Select *ImpactHello.java* and click Finish.

l Expand the new default package icon beneath the ImpactHello package icon.

m Right-click *ImpactHello.java* and select Quickly Deploy As A Web Service.

3 Generate, compile, and test client stubs and artifacts for the SOAP client using the SOAP engine.

a Generate client stub artifacts.

1 In Eclipse, select File | New | Project to create a new project to hold artifacts and the e-Biz Impact DFC/SOAP bridge code.

2 Select Sybase Web Services and click Next.

3 Enter the project name `ImpactHelloClient` and click Finish.

b Generate client artifacts.

1 Click the Sybase Web Services tab at bottom of the Eclipse upper-left pane.

2 Expand the Web Services icon.

3 Expand the WS icon.

4 Right-click ImpactHello and select Create Web Services Client.

5 Select ImpactHelloClient and click Next.

6 Leave the package name blank and click Next twice.

7 Click Finish.

The ImpactHelloClient project now contains all required SOAP client stubs and artifacts for the ImpactHello service.

c Adjust and test the client.

1 Expand the ImpactHelloClient icon in the Package Explorer pane.

2 Expand the default package icon.

3 Open (double-click) the *ImpactHello_ServiceTestClient.java* test client.

4 Replace the line "`client.main(null);`" with:

```
System.out.println("Result: " +
client.odlHello("Hello from soap client"));
```

5 Select File | Save.

6 Select Run | Run As | Java Application. The console should display "Result: Hello from ODL."

4 Inspect the logs.

Typical output from \Sybase\EAServer\bin\Jaguar.log

Note The Java-based SOAP/DFC bridge.

```
Aug 30 10:42:56 2004:   odlHello()...
Aug 30 10:42:56 2004:   DfcClient connected to localhost:31010
Aug 30 10:42:56 2004:   setup odlHello() arguments
Aug 30 10:42:56 2004:   call odlHello() with message...
Aug 30 10:42:56 2004:   Function Name:odlHello
Aug 30 10:42:56 2004:   Function Flavor:1
Aug 30 10:42:56 2004:   Function Timeout:30s
Aug 30 10:42:56 2004:   Message Type: request
Aug 30 10:42:56 2004:   Arg Name:sInOut
Aug 30 10:42:56 2004:   Arg Mode: 3 (Input, Output)
Aug 30 10:42:56 2004:   Arg Type: 10 (String)
Aug 30 10:42:56 2004:   Arg Size: 22
Aug 30 10:42:56 2004:   Arg Value: Hello from soap client
Aug 30 10:42:56 2004:   odlHello() returns message
Aug 30 10:42:56 2004:   Function Name:odlHello
Aug 30 10:42:56 2004:   Function Flavor:1
Aug 30 10:42:56 2004:   Function Timeout:30s
Aug 30 10:42:56 2004:   Message Type: response
```

```
Aug 30 10:42:56 2004:   Arg Name:sInOut
Aug 30 10:42:56 2004:   Arg Mode: 3 (Input, Output)
Aug 30 10:42:56 2004:   Arg Type: 10 (String)
Aug 30 10:42:56 2004:   Arg Size: 15
Aug 30 10:42:56 2004:   Arg Value: Hello from ODL.
Aug 30 10:42:56 2004:   Arg Name:Return
Aug 30 10:42:56 2004:   Arg Mode: 2 (Output)
Aug 30 10:42:56 2004:   Arg Type: 3 (Int)
Aug 30 10:42:56 2004:   Arg Size: 1
Aug 30 10:42:56 2004:   Arg Value: 1
Aug 30 10:42:56 2004:   odlHello() return: Hello from ODL.
```

Typical output from *soap2odl.xlog1*

Note *soap2odl.xlog1* is a e-Biz Impact ODL-based DFC service.

```
2004-08-30 10:42:56.937 | ims54cluster | 800 | 1288 | hello.1 | :odlHello:
called with: Hello from soap client

2004-08-30 10:42:56.937 | ims54cluster | 800 | 1288 | hello.1 | :odlHello:
returning: Hello from ODL.
```

Index

J

L

M

O

P

R

S

W