SUN SEEBEYOND

eWAY™ DEVELOPMENT KIT USER'S GUIDE

Release 5.1.2

Sun microsystems

Part Number: 819-7384-10

Version 20061006165502

Contents

Chapter 4

Chapter 5

Chapter 1

Introducing the eWay Development Kit

Welcome to the *eWay™ Development Kit User's Guide*. This document includes information about installing, configuring, and using the Sun SeeBeyond eWay Development Kit, also referred to as the eDK throughout this guide.

What's in this Chapter

1.1 About the eWay Development Kit

The eWay Development Kit is a development tool for creating Sun SeeBeyond eWay™ components that work with the Sun Java Composite Application Platform Suite, version 5.1.2.

The eWay Development Kit includes the following:

- A design-time build tool to define the eWay Interfaces (both inbound and outbound)
- A code generation tool to allow eWay code generation. This code works in both Java Collaborations and eInsight Business Processes (BPEL).

The eDK is designed to alleviate many of the tedious development steps required to build eWays by automating the standard build process. eWays created with the eDK are built, packaged, installed, and executed the same way as standard Sun SeeBeyond eWays.

Using the eDK does not require any specialized knowledge of APIs to tie the eWay component into the Sun Java Composite Application Platform Suite, also referred to throughout this guide as Java CAPS. The only requirement is an understanding how to connect and exchange data with the external system. eWays created with the eDK conform to the JCA (J2EE Connector Architecture) 1.5 standard.

1.2 What's New in This Version

The following changes apply to eWays created with the eWay Development Kit:

What's New in Version 5.1.2

There are no new features in this release.

What's New in Version 5.1.1

- Allows development of JCA 1.5 compliant custom eWays, which you build, install and run within Java CAPS like any standard Sun SeeBeyond eWay.
- eDK based eWays support Java Collaborations and eInsight Business Processes.
- Ability to develop eWays for both Inbound and Outbound communication.
- Support for using third-party libraries and JNI.
- Support for using user defined classes and data types.
- Support for modifying and extending eWay configuration.
- Ability to regenerate the eWay definition via an **.xml** definition file.

What's New in Version 5.1

This product was not released in version 5.1.

1.3 What's in This Document

This document includes the following chapters:

- **Chapter 1 "Introducing the eWay Development Kit"** provides an overview of the eWay Development Kit.
- **Chapter 2 "Installing the eWay Development Kit"** provides installation instructions, including a list of supported operating systems, system requirements, and JNI protocol considerations.
- **Chapter 3 "eWay Development Process"** describes the process involved in creating an eDK-based eWay.
- **Chapter 4 "Using the eDK Build Tool"** describes the steps required to build an eWay using the eDK Build Tool.
- **Chapter 5 "Implementing Your eWay"** describes how to implement an eDK-based eWay.
- **Chapter 6 "eDK eWay Concepts and Best Practices"** provides suggestions and tips on creating eDK based eWays.
- **Chapter 7 "Sample eDK eWays"**describes how to implement and generate an eDK File eWay.
- **Chapter 8 "Adding and Sending Custom Alert Messages"** describes how to add and send custom Alert messages using the Resource Adapter framework.

- **Appendix A "J2EE Connector Architecture"** contains additional information on the J2EE Connector Architecture and RA Framework.
- **Appendix B "Command Line Code Generation"** contains additional information on Generating eDK code by command line, and manually creating an eDK definition file.
- **Appendix C "JNI Sample eWay"** describes how to create a JNI based eWay.

1.3.1 Scope

This guide describes how to install and use the eWay Development Kit for the purpose of developing new eWays that function within Java CAPS. Additional detailed information, such as the steps required to create eDK based eWay projects in the Sun SeeBeyond Enterprise Designer are not included in this guide.

1.3.2 Intended Audience

This guide is intended for intermediate to experienced developers who have a Java programming background. To use the eDK, you should have basic knowledge of the following technologies:

- J2EE standards and methodology
- JCA standards
- Existing Java CAPS products, such as the Sun SeeBeyond eGate™ Integrator and Sun SeeBeyond eInsight™ Business Process Manager, and the role an eWay plays within that product line
- Some understanding of Web Services and their representation using eInsight Business Processes
- Windows-based operating systems
- Operating systems supported by eDK based eWays

1.3.3 Text Conventions

The following conventions are observed throughout this document.

Table 1 Text Conventions

Text Convention	Used For	Examples
Bold	Names of buttons, files, icons, parameters, variables, methods, menus, and objects	▪ Click **OK**. ▪ On the **File** menu, click **Exit**. ▪ Select the **eGate.sar** file.
`Monospaced`	Command line arguments, code samples; variables are shown in ***bold italic***	`java -jar ***filename***.jar`
Blue bold	Hypertext links within document	See **Text Conventions** on page 10

Table 1 Text Conventions (Continued)

Text Convention	Used For	Examples
Blue underlined	Hypertext links for Web addresses (URLs) or email addresses	http://www.sun.com

1.4 Related Documents

The following documents provide additional information about the product suite:

- *Sun SeeBeyond eGate™ Integrator User's Guide*
- *Sun SeeBeyond eGate™ Integrator Tutorial*
- *Sun SeeBeyond eInsight™ Business Process Manager*
- *Sun SeeBeyond Alert Agent User's Guide*

1.5 Sun Microsystems, Inc. Web Site

The Sun Microsystems web site is your best source for up-to-the-minute product news and technical support information. The site's URL is:

http://www.sun.com

1.6 Documentation Feedback

We appreciate your feedback. Please send any comments or suggestions regarding this document to:

CAPS_docsfeedback@sun.com

Chapter 2

Installing the eWay Development Kit

This chapter describes how to install the eWay Development Kit.

What's in this Chapter

2.1 System Requirements

To use the eWay Development Kit, you need:

- eGate Repository
- TCP/IP network connection

2.2 Installing the eDK

The Java Composite Application Platform Suite Installer, a web-based application, is used to select and upload eWays and add-on files during the installation process. The following section describes how to install the components required for this eWay.

Refer to the readme for the latest information on:

- Supported Operating Systems
- External System Requirements
- Java Composite Application Platform Suite Requirements
- Known Issues

2.2.1 Installing the eDK on an eGate Supported System

Follow the directions for installing Java CAPS in the *Composite Application Platform Suite Installation Guide.*

After you have installed eGate or eInsight, do the following:

1. Create a new eDK root directory folder (for example: C:\eDK).
2. From the Suite Installer, click the Administration tab, and then click the link to install additional products.
3. Select the following product from the eWay category:
   - **eWayDevelopmentKit**

   Select the following in the Documentation category to upload the User's Guide, Help file, and Readme for the eDK:
   - **eWayDevelopmentKitDocs**
4. Once you have selected all of your products, click **Next** in the top-right or bottom-right corner of the **Select Java Composite Application Platform Suite Products to Install** box.
5. From the **Selecting Files to Install** box, locate and select your first product's SAR file. Once you have selected the SAR file, click **Next**. Your next selected product appears. Follow this procedure for each of your selected products. The **Installation Status** window appears and installation begins after selecting the SAR file.
6. Once your product's installation is finished, click the **Downloads** tab and select the eWay Development Kit.
7. Click **Open eWay** from the File Download window and extract the **edk.zip** file to the eDK root directory folder that you created earlier.
8. Unzip the contents of the **edk.zip** file to the same root directory folder.
9. Continue installing the Java Composite Application Platform Suite as instructed in the *Composite Application Platform Suite Installation Guide.*

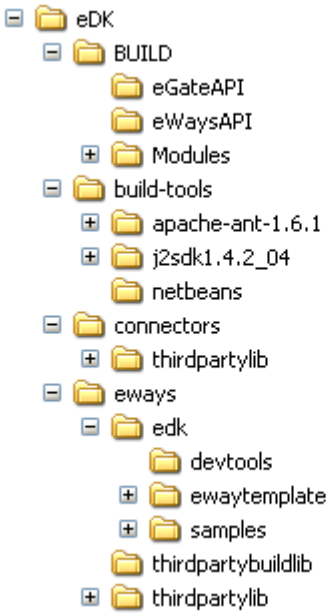
2.3 Directories Created After Installation

Several directories are extracted from the **edk.zip** file, including a subdirectory for development tools and a subdirectory containing code templates.

Main directories created after extraction include:

- **BUILD** – contains all the API **.jar** files required to build an eWay.
- **build-tools** – contains all the build tools (**Apache Ant**, **JDK 1.4™**, and **Netbeans™**) that are necessary to build an eWay.
- **connectors** – contains any third-party **.jar** file required by the eWay.
- **eways** – contains all necessary third-party **.jar** files, and development tools to run the eDK, as well as a folder containing eDK samples.

Figure 1 Directories Created After Installation



2.3.1 Additional Files Created During Installation

The following files are also created during installation of the eWay Development Kit Build Tool.

- **env.bat** – used to set the eDK Build Tool environment.
- **stc.properties** – contains global property settings used in **Apache Ant** build scripts.
- **runedk.bat** - used to launch the eWay Development Build Tool.

2.4 Setting The Implementation Environment and Starting the eDK Build Tool

An implementation environment is required to compile Java source files and to build an eWay **.sar** file.

Steps to set the implementation environment and start the build tool include:

1. Open a Command Prompt window and go to the eDK Root Directory. This is the same directory where you extracted the **edk.zip** file.
2. In the same Command Prompt window, type:

   ```
   set CLASSPATH=
   ```

   This clears your existing class path (required before setting your implementation environment).
3. Run the **runedk.bat** file to start the Build Tool.

Chapter 3

eWay Development Process

This chapter describes the process involved in creating an eDK-based eWay.

What's in this Chapter

3.1 Overview

eDK eWay development is a two-stage process that includes Java source code and component creation, and implementation. To better understand the development process, we commonly describe any eWay shell code generation as being facilitated using the **eDK Build Tool**, and any shell code implementation as being facilitated using the **Implementation Environment**.

3.1.1 eDK Build Tool

The eDK Build Tool is a stand-alone application that you use to generate Java source code for your custom eWay. An eWay is basically a J2EE Resource Adapter that runs within Java CAPS and interacts with the Enterprise Information Systems (EIS). An eWay in Java CAPS comprises the following:

- A J2EE Resource Adapter.
- An OTD that exposes the eWay methods and attributes in the JCD.
- A Web Service Description Language (WSDL) file that defines the eWay's Business Process interface in eInsight. This interface has both operations and attributes.
- A NetBeans module which contains design time artifacts. This module plugs into the Enterprise Designer.
- Connectivity map and Environment configuration, represented by a configuration template.
- Design time and runtime code, which generates the necessary code for both deployment and runtime, in an application server.
- Monitoring code.
- Code to interact with the various modules of the Java CAPS framework during design-time and run-time.

The source code and components you generate with the eDK Build Tool expedites the development process, allowing you to focus on defining the eWay and implementing the EIS interactions.

Some of the files generated by the eDK Build Tool include:

- All Java source files required to create a Java resource adapter.
- A NetBeans module, including all eWay design-time artifacts required to plug into the Enterprise Designer.
- The WSDL required to plug into the Business Process Designer (this code is only generated when eInsight Business Process operations are defined in an eWay definition).
- All Java source files required for eWay code generation activation.
- All Java source files required for eWay run-time components.
- Resource bundles for logging and Alert Codes.
- Other miscellaneous resource files, such as image icons.
- Build scripts required to build the eWay **.sar** file.

For additional information on using the eDK Build Tool, see **Using the eDK Build Tool** on page 18.

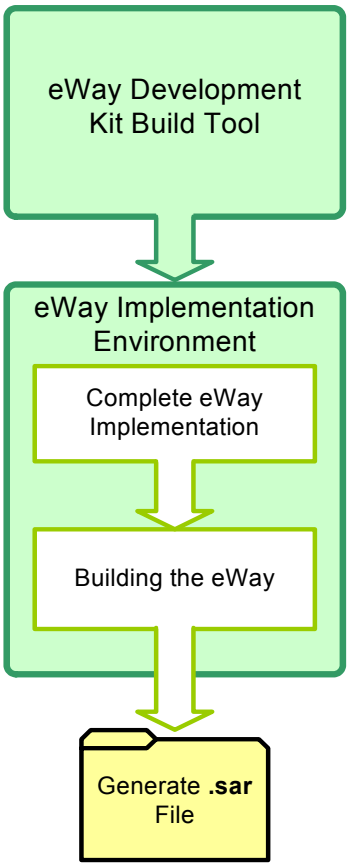
3.1.2 eWay Implementation Environment

An eWay Implementation Environment refers to any development required to implement shell code generated by the eDK Build Tool. For additional information on how to implement an eWay, see **Implementing Your eWay** on page 43.

The final result of eWay Implementation is creation of **.sar** file for use in the Enterprise Designer.

Figure 2 illustrates the relationship between the eDK Build Tool and the eWay Implementation Environment.

Figure 2 eWay Development Kit Component Overview



Use the eWay Development Kit Build Tool to generate eWay shell code.

Use the eWay Implementation Environment to complete the eWay implementation.

Generate a **.sar** file for the new eWay.

Chapter 4

Using the eDK Build Tool

This chapter describes the steps required to build an eWay using the eDK Build Tool.

What's in this Chapter

4.1 Steps Required to Build an eWay

The following steps outline a typical user experience of using the eDK to build an eDK-based eWay for implementation in an Java CAPs Project.

Steps required to build an eWay using the eWay Development Kit include:

1. Start the eWay Development Kit Build Tool.
2. Create and specify details of the new eWay – such as the eWay Name, Description, Version, and so forth.
3. Define the eWay Interfaces to be used in Java Collaborations or the eInsight Business Processes.
4. Define the eWay configuration template.
5. Run the code generator to create the eWay shell code (using either the eWay Development Kit Build Tool or by command line).
6. Implement the generated shell code, and run **Apache Ant scripts** to build the JCA components.
7. Run the **Apache Ant** build scripts to build the eWay **.sar** file.

After building the eWay, upload the newly created **.sar** file to the Repository using the Suite Installer and run the Enterprise Designer Update Center. You can now create, build, and deploy a new Project using the new eDK based eWay. Refer to the steps outlined in **Upload the New eWay to the Repository** on page 63 for more information.

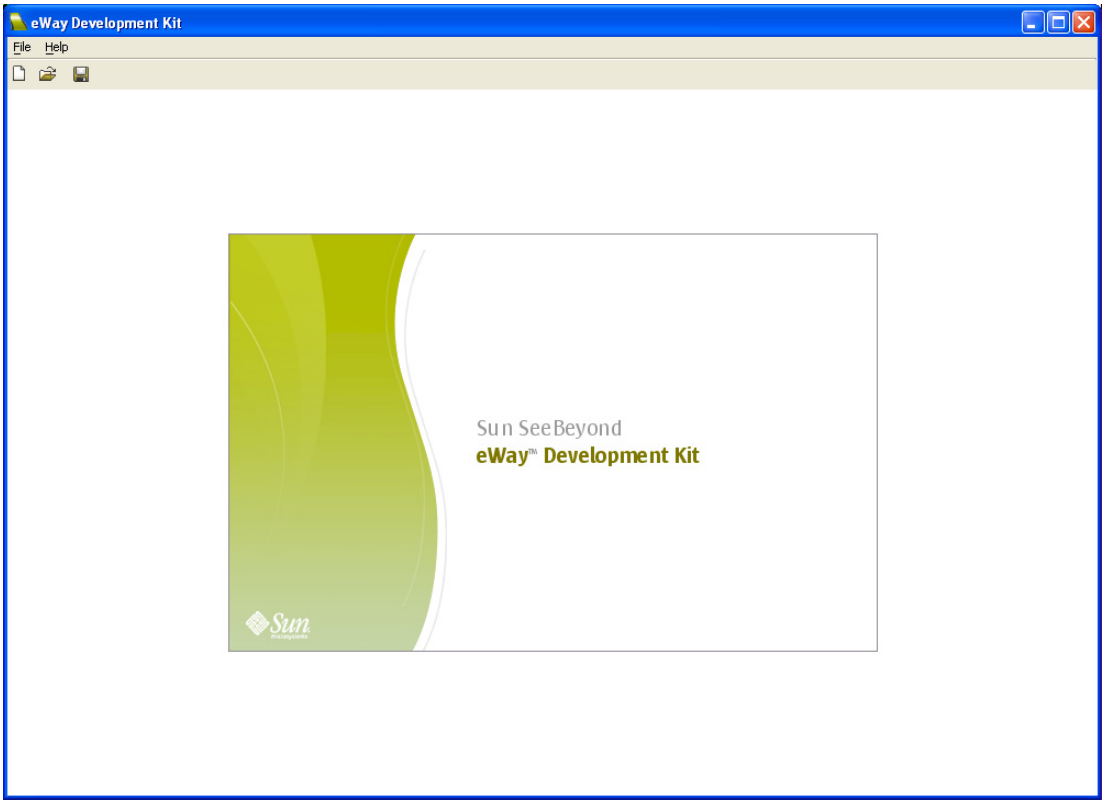
Note: *An Implementation Environment is required to compile Java source files and to build an eWay **.sar** file. For more information, see* **Setting The Implementation Environment and Starting the eDK Build Tool** *on page 14.*

4.2 Start the eWay Development Kit Build Tool

To start the eWay Development Kit build tool:

1. Browse to the eDK root directory.
2. Run the **runedk.bat** file to launch the eDK Build Tool. The eWay Development Kit splash screen appears.

Figure 3 eWay Development Kit Build Tool



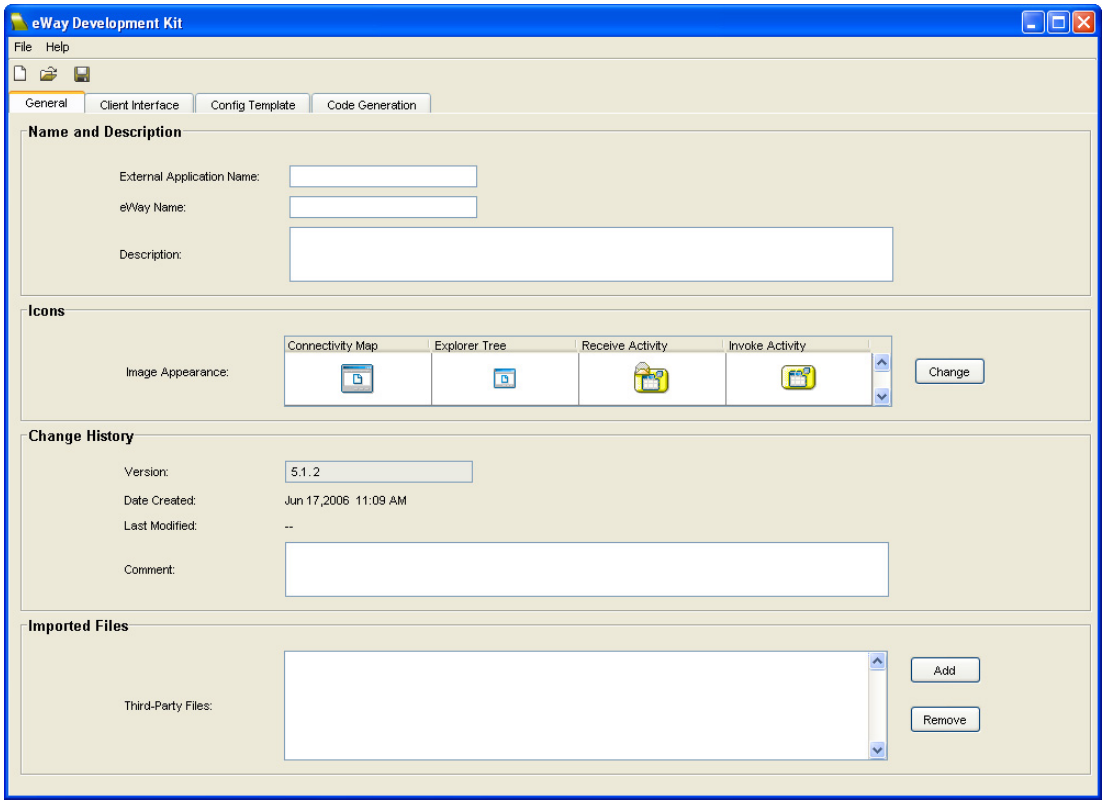
4.3 Create and Specify the New eWay

Perform the following steps to specify a new eWay:

1. From the menu bar, select **File** then select **New eWay**, or click the **New eWay** icon. An empty eWay template appears in the build tool window, see Figure 4.
2. Set up the new eWay definition by entering or setting the following features on the General tab on the eDK Build Tool.
   - External Application Name (required)
   - Name and Description (optional)
   - Icons (optional)
   - Change History (optional)
   - Imported Files (optional)

Figure 4 illustrates the features found on the General tab of the eWay Development Kit Build Tool.

Figure 4 eWay Development Kit Build Tool - General Tab



4.3.1 Name and Description

The Name and Description section contains the following fields:

- **External Application Name** – represents the external application name that is used in the Connectivity Map and Environment Explorer in the Enterprise Designer. This field is required to generate the eWay.
- **eWay Name –** identifies the name of the eWay. The name entered in this field is used to create the folder name in the generated code. For example, if the eWay name is "myeWay", then the folder name for the eWay will be "myewayadapter". An entry for the eWay Name field must be in lower case, and is optional. Note that if you do not provide an eWay Name, the build tool uses the folder name as the external application name.
- **Description –** complete the Description field to enter notes about the new eWay

Naming Restrictions:

The following standard Java Identifier naming conventions apply to the eWay Name and External Application Name:

- **Alphabetic letters** – names should only contain alphabetic characters, such as "AA" or "aa", or the underscore.
- **Digit** – names can only contain numbers between zero and nine (0 - 9). Also, the first position of the name should not contain a digit.

Note: *Using an underscore in the External Application Name can cause the eWay component to not appear on Enterprise Monitor Logging Control.*

4.3.2 Icons

Image Appearance – alters the appearance of eWay icons on the Connectivity Map, Explorer Tree, or the Receive or Invoke web service activities. It is required to have a valid icon file location.

To change an eWay icon:

1. Click **Change**, and browse to the location of a suitable file. File types can be any of the following:
   - **gif**
   - **png**
   - **jpg**
2. Click **Open**. The new icon now appears in the Image Appearance box.

The maximum size for icons used are:

Table 2 Maximum Icon Size Accepted

Image Appearance Icons	Size in Pixels	Appears On
Connectivity Map	30 x 27	Enterprise Designer Connectivity Map

Table 2 Maximum Icon Size Accepted

Image Appearance Icons	Size in Pixels	Appears On
Explorer Tree	20 x 20	Project Explorer, the Connectivity Map, and the External Application
Receive Activity	32 x 32	eInsight Canvas
Invoke Activity	32 x 32	eInsight Canvas

4.3.3 Change History

Change History includes the following:

- **Version** – not a required field. The default version is 5.1.2. The version number appears on the **Administration** tab of the **Java Composite Application Platform Suite Installer**, and in the **Update Center Wizard** of the **Enterprise Designer**.
- **Date Created** – identifies the eWay creation date. This field is read-only, and is automatically created by the build tool.
- **Last Modified** – identifies the date and time of the last saved eWay modification. This is a read-only field that is automatically created by the build.
- **Comments** – is used to enter comments each time you create or update the eWay.

4.3.4 Imported Files

Third-Party Files – is used to import all custom third-party **.jar** files required for the eWay.

To import a file:

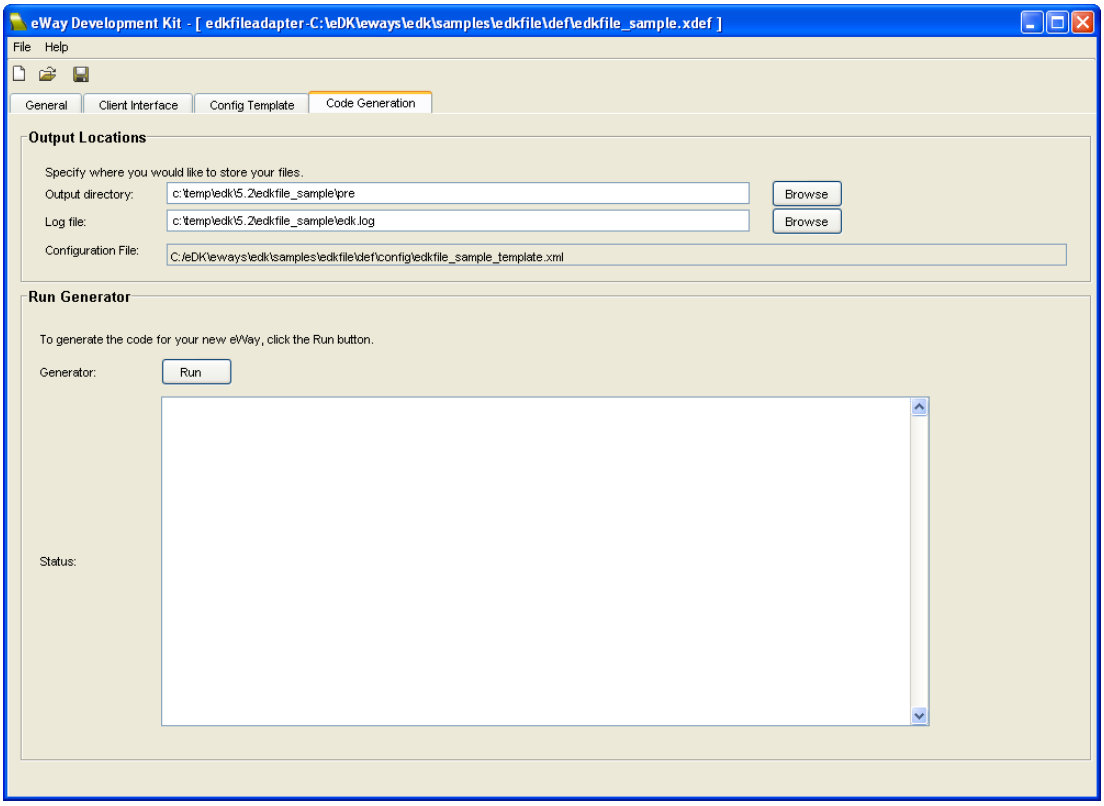
1. Click **Add**, and browse to the location of the **.jar** file.
2. Click **Open**. The name and location of the executable **.jar** file appears in the textbox.

To remove a file, select a third-party file from the Third-Party text box and click **Remove**.

Note: *Code generation copies these .jar files to the implementation environment.*

Figure 5 displays the completed General tab of the **EDKFILE** eWay. Notice that only a few fields are completed since only the External Application Name is required.

Figure 5 eWay Development Kit Build Tool - Completed General Tab



4.4 Enter the Required eWay Client Interfaces

In the eWay Development Kit, the Client Interfaces represent the methods, attributes, operations, user defined classes, and user defined types, to be exposed and implemented in the eWay OTD. The methods defined in the JCD client interface defines what JCD methods are exposed by the eWay.

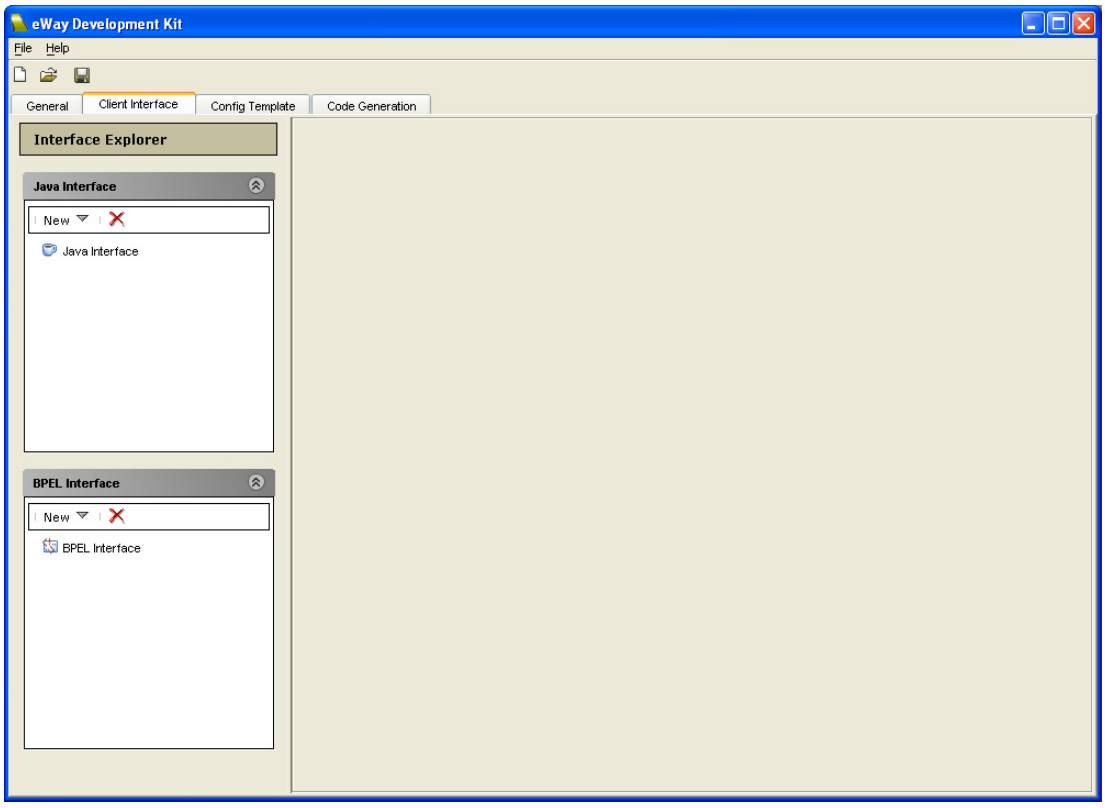
Client interfaces can contain both inbound and outbound operations for Java Collaborations and eInsight Business Processes.

4.4.1 Steps Required to Create eWay Interfaces

Perform the following steps to create eWay Interfaces:

1 Select the Client Interface tab on the eWay Development Kit Build Tool. The Client Interface window appears.

Figure 6 eWay Development Kit Build Tool - Client Interface Tab



2 From the Interface Explorer, click **New** and select one of the following:

- **Java OTD Method** – to create inbound or outbound methods (JCD only).
- **User Defined Class** – to create inbound or outbound class types (data containers in eInsight Business Processes)
- **Java OTD Attribute** – to create inbound or outbound attributes (JCD only).
- **Operation** – to create inbound or outbound operations (eInsight Business Processes only).
- **User Defined Data Type** – to create a user-defined data container (eInsight Business Processes only).

3 Enter the details for the selected Java or eInsight Business Process interface.

Naming Restrictions:

The following standard Java Identifier naming conventions apply to Method, Operation, and Attribute names:

- **Alphabetic letters** – names should only contain alphabetic characters, such as "AA" or "aa", or the underscore.
- **Digit** – the first position of the name should not contain a digit.

4.4.2 Creating Java OTD Methods

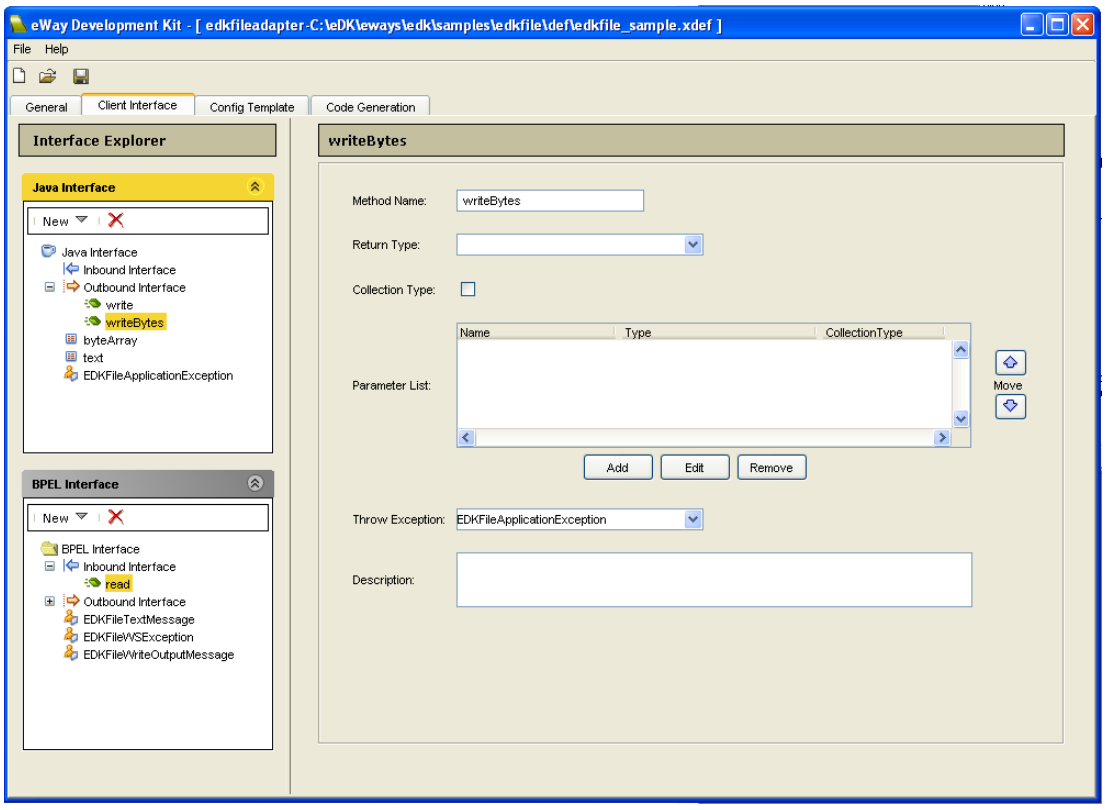
You can use the eDK Build Tool to create inbound and outbound JCD methods. A User Defined Type can contain attributes of primitive types or complex data types, or other User Defined Types. They can be used as the type definition for method parameters, return types, and exception types. Any method created is exposed in Java Collaboration Definitions.

To create a new method:

1. From the Java Interface box, click **New** > **Java OTD Methods** > **Inbound** or **Outbound**.
2. Enter a new method name in the **Method Name** field.
3. Select a return type for the method from the **Return Type** drop-down box. You can also enter a data type (a User Defined class or a JDK library class) if the desired type is not in the desired drop-down list.
4. Select the **Collection Type** checkbox to specify that the type of object returned is a collection.
5. Click **Add** to create a new parameter in the **Parameter List**.
6. Enter or select an exception to throw in the **Throw Exception** combo-box.
7. Enter a description for the method in the **Description** textbox.

Figure 7 displays an example of the **writeBytes()** method in the EDKFILE sample eWay.

Figure 7 writeBytes Method in the EDKFILE Sample eWay



4.4.3 Creating Java OTD Attributes

JCD OTD Attributes are data fields that describe an object's characteristics. For example, an attribute may be the "capacity" of the **elevator** class, or it may describe the state of an object, such as the "isMoving" of the **car** class. Attributes have a name and a type. Data types can be either primitive or a User Defined type (such as a previously created class).

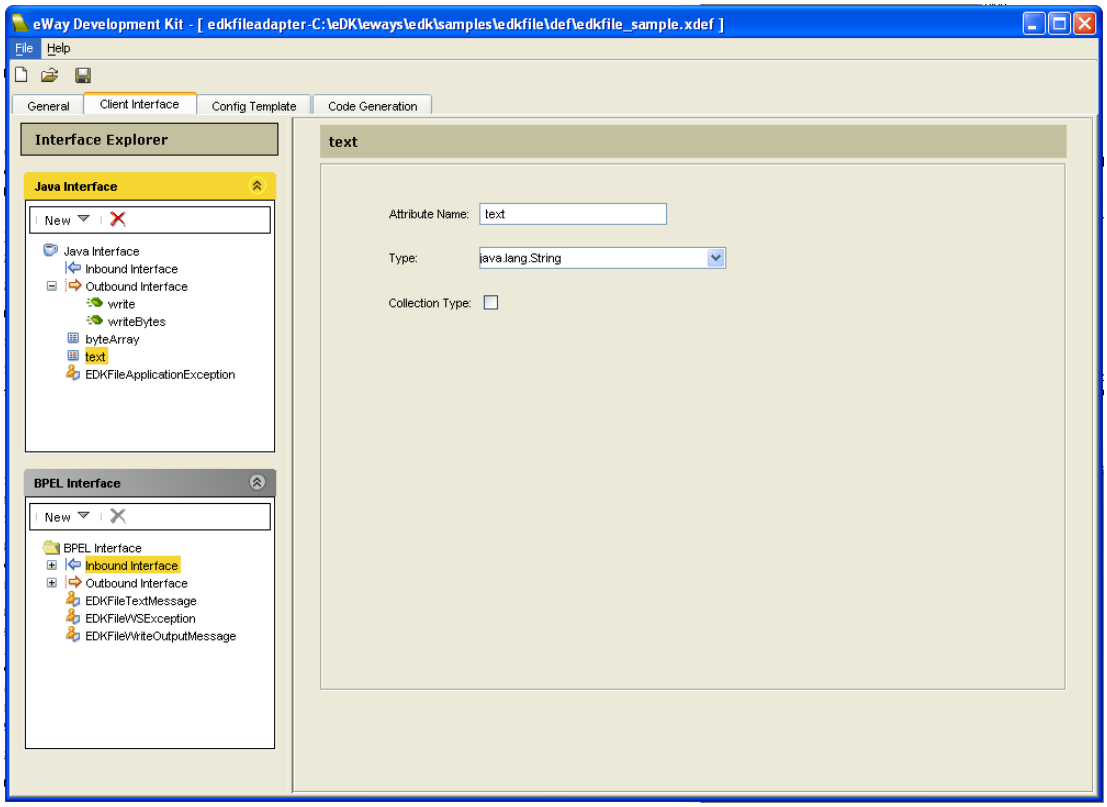
The Client Interface tab in Figure 6 includes an Interface Explorer to add placeholders for new Java based Attributes.

To create a new Java OTD Attribute:

1. From the Java Interface box, click **New** > **Java OTD Attribute**.
2. Enter a new attribute name in the **Attribute Name** field.
3. Enter or select a data type for the attribute from the **Type** drop-down list.
4. If the attribute is a collection, select the **Collection** checkbox.

Figure 8 displays an example of the Java OTD Attribute **text** in the EDKFILE sample eWay.

Figure 8 Java OTD Attribute text in the EDKFILE eWay



4.4.4 Creating User Defined Classes and User Defined Data Types

You create User Defined Classes in the Java Interface box and User Defined Data Types in the BPEL Interface box.

User Defined Class

A User Defined class can represent a Java class, which in turn can contain any number of attributes or methods. You can use these classes as type definitions for attributes, return types, or exception types in Java interfaces.

User Define Data Type

A User Defined data container can represent the complex type definition in the in-line XSD schema in the WSDL. BPEL Interface (eInsight Business Process) User Defined Data Types are data containers only; they are Bean classes that only contain setter or getter methods for attributes. The internal implementation may contain additional methods or attributes, but what is seen by the eWay user is only what is described in the WSDL. For additional information on WSDL files and eInsight Business Processes, see **eInsight Business Process (BPEL) Operations in Java CAPS** on page 32.

To create a new User Defined Class or a User Defined Data Type:

1. From the Java Interface box, click **New** > **User Defined Class**.

   or

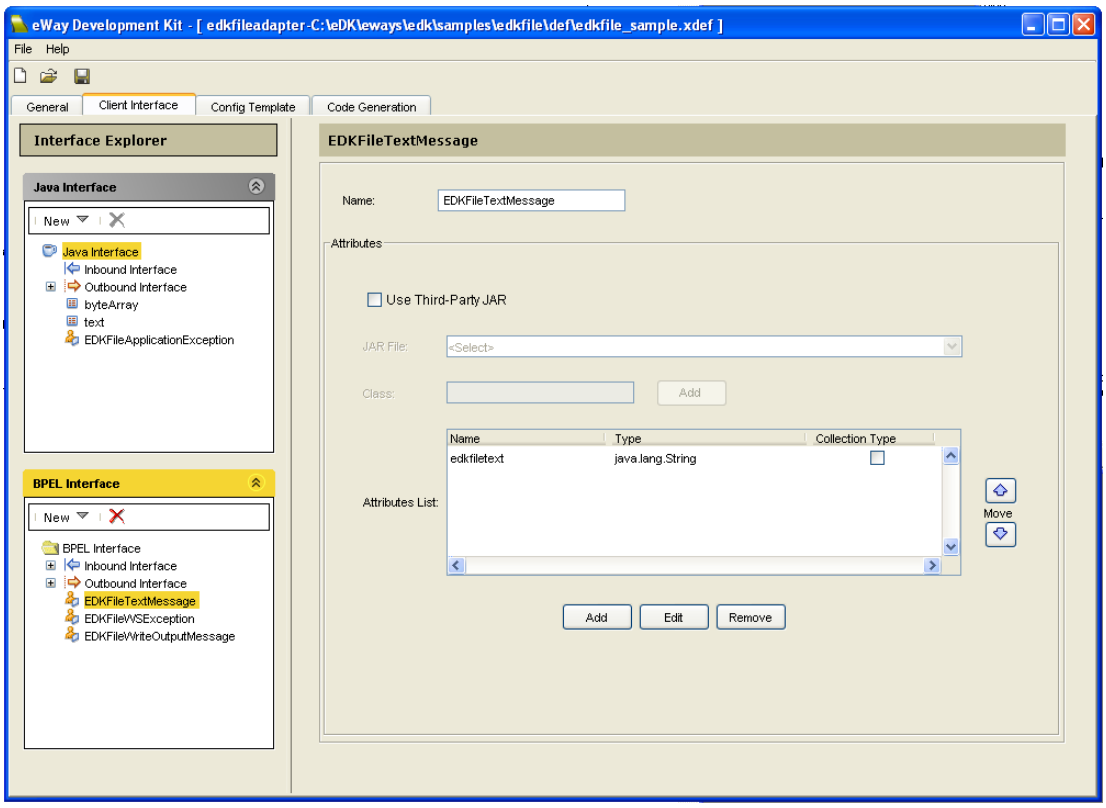
   From the BPEL Interface, click **New** > **User Defined Data Type**.
2. Enter a new name in the **Name** field.

To use a third-party **.jar** file:

3. Select the Use **Third-Party JAR** file checkbox.
4. Select an available **.jar** file from the drop-down list if you previously added one on the **General** tab.
5. Click **Add**. The **Third-Party JAR** file window opens containing class files organized by package.
6. Locate the required class file and click **Select**. The Class file with available attributes appears in the **Attributes List**. These attributes are not editable.

Figure 9 displays an example of the User Defined Data Type **EDKFileTextMessage** in the **EDKFILE** sample eWay.

Figure 9 User Defined Data Type EDKFileTextMessage



Defining User Defined Classes

Defining an eWay's Java Client Interface involves defining the Java methods and attributes. These methods and attributes appear in the eWay's OTD. You define a User Defined class in a Java Interface by first adding the attributes or methods required by the eWay. Code generation then creates a Java class—for each User Defined Class—that includes getter and setter methods for each attribute, and empty method declarations for each method specified.

To create a User Defined Class named **Person**, that contains an attribute or method:

1. Click **New** > **User Defined Class** in the Java Interface box.
2. Name the new class **Person**.
3. Right-click the User Defined Class named Person, select **New Attribute**, and then enter the following:
   - Name = name
   - Type = String
4. Right-click the User Defined Class named Person, select **New Attribute**, and then enter the following:
   - Name = address
   - Type = String

5 Right-click the User Defined Class named Person, select **New Attribute**, and then enter the following:

- Name = employer
- Type = String

6 Right-click the User Defined Class named Person, select **New Method**, and then enter the following:

- Method Name = validateAddress()
- Return Type = boolean
- Parameter List Type = address

These steps generate Person.java, as seen below.

```
public class Person {
   private String mName;
   private String mAddress;
   private String mEmployer;

   private void setName(String name) {
       this.mName = name;
   }

   public String getName() {
      return this.mName;
   }
private void setAddress(String address) {
       this.mAddress = address;
   }

   public String getAddress() {
      return this.mAddress;
   }

private void setEmployer(String employer) {
       this.mEmployer = employer;
   }

   public String getEmployer () {
      return this.mEmployer;
   }


   public boolean valiateAddress(String address) {
       // <Start_User_Code>

        // <End_User_Code>
  }
}
```

This code creates an all purpose Java class, which can serve as a utility class, or the Java type of OTD attributes defined in the Java client interface. You are not required to define any User Defined class, it is a convenience provided if needed.

User defined classes can also be useful in the case when there is an existing class—other than classes provided in the JDK library—that you wish to use as an OTD attribute type or return type for the methods defined in the Java Interface. You can create a User Defined class that points to an existing class in an existing **.jar** file. This newly created User Defined class becomes available in the drop-down box as a selection of Java type.

Note: *It is not required to have any method defined in the User Defined class. If a method is defined, then it becomes your responsibility to complete any applicable implementation.*

Table 3 illustrates how JCD Interfaces created in the eDK Build Tool appear in the Enterprise Designer.

Table 3 Java Interface Comparison

Client Interfaces Appearing in the eWay Development Kit Build Tool:	How these Client Interfaces Appear in Enterprise Designer:
Methods	Methods in the eWay OTD
User Defined (JCD Java Classes)	Not specific
Attributes (JCD only)	Attributes in the eWay OTD

Defining User Defined Data Types

The information you enter in the BPEL interface is used to create a WSDL file and a Java class with empty method declarations for each operation specified.

You define User Defined Data Types in the BPEL Interface box. They are similar to User Defined Classes, but an important distinction is that User Defined Data Types are, just as the name indicates, data containers only. No method definition is allowed. When eDK code generation is finished, a Java Bean class is generated for each User Defined Data Type. The generated Java class follows the Java Bean paradigm.

To create a User Defined Data Type named **Person**, that contains additional attributes:

1. Click **New** > **User Defined Data Type** in the BPEL Interface box.
2. Name the new class **Person**.
3. Right-click the User Defined Data Type named Person, select **New Attribute**, and then enter the following:
   - Name = name
   - Type = String
4. Right-click the User Defined Data Type named Person, select **New Attribute**, and then enter the following:
   - Name = address
   - Type = String
5. Right-click the User Defined Data Type named Person, select **New Attribute**, and then enter the following:
   - Name = employer
   - Type = String

These steps generate Person.java, as seen below.

```
public class Person {
```

```
private String name;
private String address;
private String employer;
private void setName(String name) {
        this.mName = name;
    }

    public String getName() {
       return this.mName;
    }
private void setAddress(String address) {
        this.mAddress = address;
    }

    public String getAddress() {
       return this.mAddress;
    }

private void setEmployer(String employer) {
        this.mEmployer = employer;
    }

    public String getEmployer () {
       return this.mEmployer;
    }
}
```

If the User Defined Data Type is used as either as an input or output message container for the eInsight Business Process operations, then the generated Bean class also implements the interfaces that are required in Java CAPS.

A corresponding Bean class is generated by the eDK Build Tool for each User Defined Data Type. As with User Defined Classes, you can define a User Defined Data Type that points to an existing Java class in an existing **.jar** file, but it is important that the existing class conforms to Java Bean standards. Runtime exceptions, which are difficult to debug when built and used in Java CAPS, occur when the existing class files are not valid Java Bean class files.

Table 4 illustrates how eInsight Business Process Interfaces created in the eDK Build Tool appear in the Enterprise Designer.

Table 4 eInsight Business Process Interface Comparison

Client Interfaces Appearing in the eWay Development Kit Build Tool:	How these Client Interfaces Appear in Enterprise Designer:
User Defined (eInsight Business Process data containers)	Appear in the Business Rule Designer
Operations (eInsight Business Process only)	Appear in the Project Explorer and the Business Rule Designer.

4.4.5 Creating Operations

The BPEL Interface allows you to define web service operations.

To create a new eInsight Business Process Operation:

1 From the BPEL Interface, click **New** > **Operation** > **Inbound** or **Outbound**.

2 Enter a new Operation name in the **Operation Name** field.

3 Enter or select an output message for the Operation from the **Output Message** drop-down list.

4 If the output message type is a collection, select the **Collection** checkbox.

5 Click **Add** to create a new parameter in the **Parameter List** for the input message.

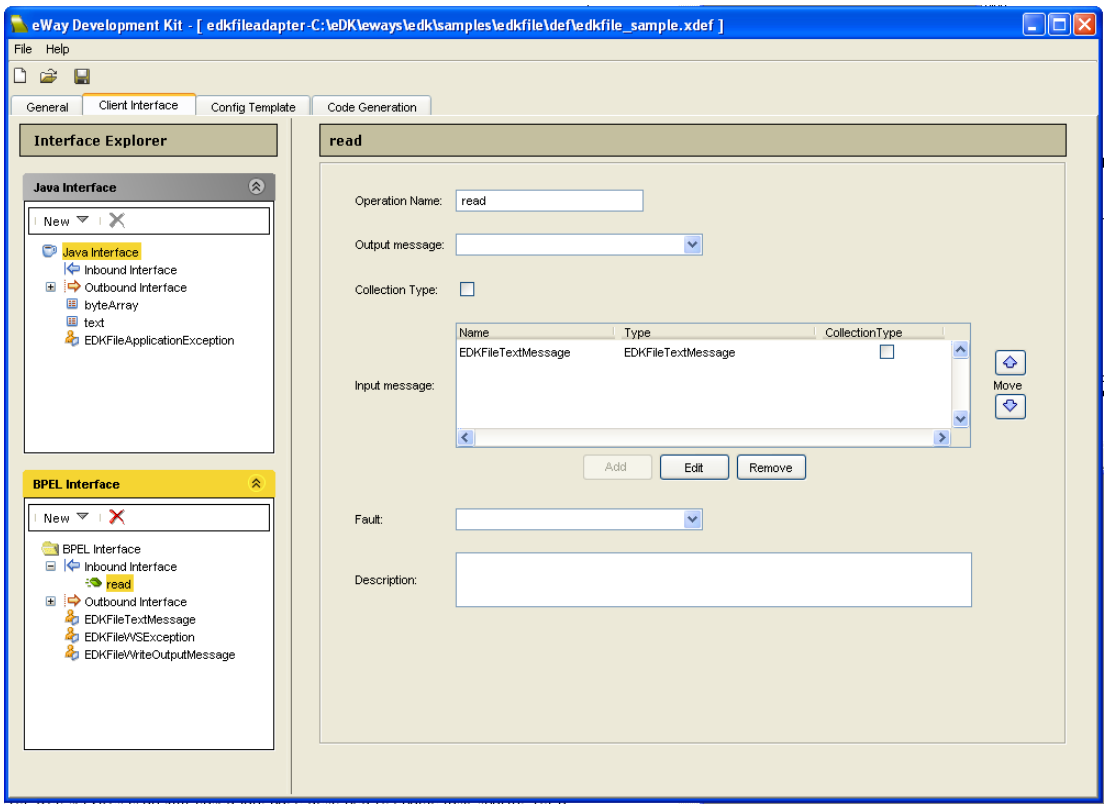
6 Enter or select an exception to throw in the **Throw Exception** combo-box.

7 Enter a description for the operation in the **Description** textbox.

Note: *The message type you enter needs to correspond to the User Defined data type.*

Figure 10 displays an example of the inbound Operation **read** in the EDKFILE sample eWay.

Figure 10 Inbound Operation read in the EDKFILE sample eWay



eInsight Business Process (BPEL) Operations in Java CAPS

BPEL for web services is an xml-based language designed to enable task-sharing for a distributed computing environment for Java developers to publish web services and compose them into reliable and transactional business flows.

BPEL is designed to keep internal business protocols separate from cross-enterprise protocols so that internal processes can be changed without affecting the exchange of

data from enterprise to enterprise. This means that any programmer using BPEL can formally describe a business process in such a way that any cooperating entity can perform one or more steps in the process the same way.

The standardization of communications protocols and message formats makes it increasingly important to structure the way communications are described. WSDL addresses this need by defining an XML grammar for describing network services as collections of communication endpoints capable of exchanging messages.

According to a W3C note dated March 15, 2001, a WSDL document defines services as collections of network endpoints, or ports. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: messages, which are abstract descriptions of the data being exchanged, and port types which are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitutes a reusable binding. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service.

For additional information on WSDL see:

http://www.w3.org/TR/wsdl#_introduction

By design, eInsight Business Process operations must follow WSDL specifications:

There are four types of webservice operations, or transmission primitives that an endpoint can support:

- **One-way** – where the endpoint receives a message.
- **Request-response** – where endpoint receives a message, and sends a correlated message.
- **Solicit-response** – where the endpoint sends a message, and receives a correlated message.
- **Notification** – where the endpoint sends a message.

Note that only one-way and request-response webservice operations are supported in Java CAPS, so all operations are required to have an input message.

When you define one or more eInsight Business Process operations in the build tool, a WSDL definition is created with the following elements:

- **port** – which specifies an address for a binding, thus defining a single communication endpoint.
- **portType** – which is a set of abstract operations. Each operation refers to an input message and output messages.
- **message** – which represents an abstract definition of the data being transmitted. A message consists of logical parts, each of which is associated with a definition within some type system.
- **types** – which provides data type definitions used to describe the messages exchanged.

Note that the WSDL generated does not contain binding and service sections. They are resolved at runtime by the Java CAPS eInsight Business Process engine.

Additional Development Notes:

eInsight Business Process operations are web service operations that should only be stateless with, at most, one input, output and fault message. The Java classes corresponding to messages (input, output, and fault messages) are either generated by the eDK Build Tool or by using a third-party class. These class files must follow the Java Bean conventions.

- eInsight Business Process operations can be either inbound or outbound (receive and invoke activities). Inbound operations and outbound operations are included in separate portypes in the WSDL file, with corresponding Java classes generated for each port type.
- Parameters passed into the operation must contain all the field attributes to be displayed on the eInsight Business Process attribute mapper. This means that any eInsight Business Process "attribute" must be defined in the User Defined Data Type which is input to, or output from the operation.

4.5 Define the eWay Configuration Template

The eWay Development Kit Build Tool's **Config Template** tab is used to create a configuration template for an eWay. A configuration template is a hierarchical-based model—represented as an **.xml** file—that contains a super-set of configuration parameters defined within sections and subsections.

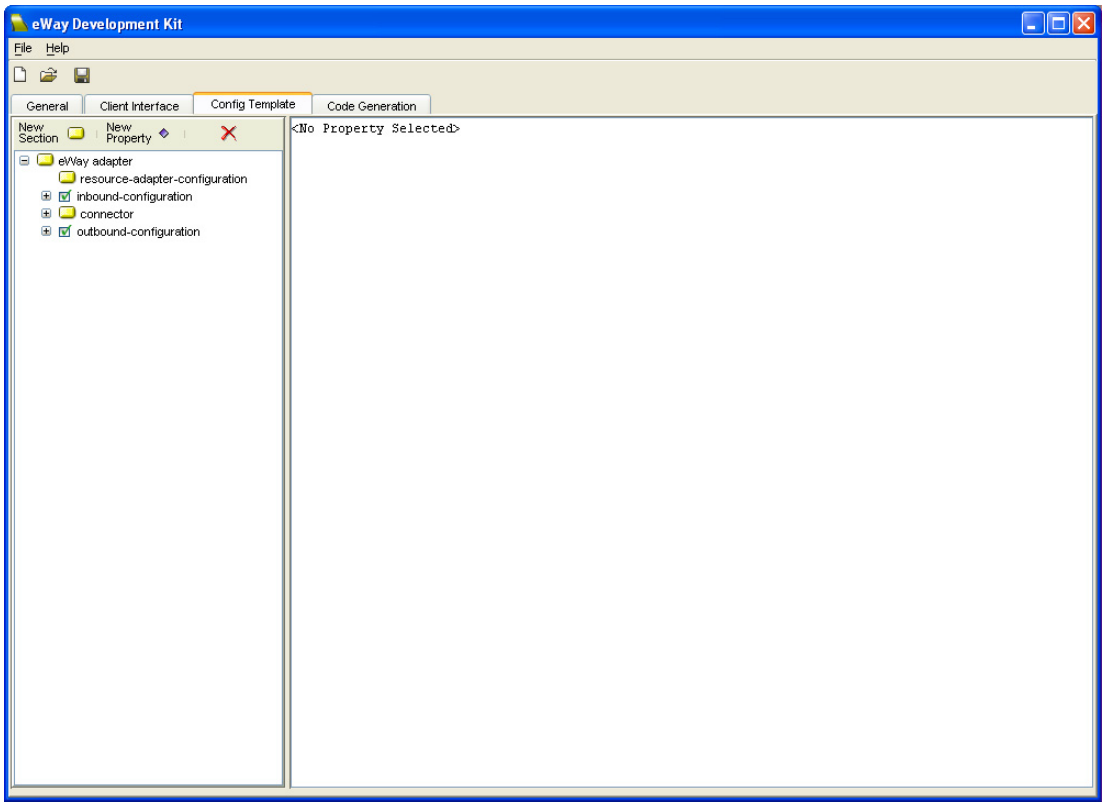
You can set Configuration parameters that contain a number of eWay specific properties. The properties you expose in the Configuration Template are later edited by the eWay user in either the Enterprise Designer **Connectivity Map**, or the **External System** eWay Environment properties.

Note: *Additional information on specifying configuration properties are found in* **Specifying Configuration Properties** *on page 55*

To add new sections and properties to the Configuration Template:

1. Select the **Config Template** tab on the eWay Development Kit Build Tool. The Config Template window appears.

Figure 11 eWay Development Kit Build Tool - Config Template Tab



2 Expand the **inbound-configuration** and **outbound-configuration** sections in configuration template tree.

3 Click on the **New Section** icon to add new sections. Click the **New Property** icon to add new properties. Alternately, you can also right-click a section and create a new section or property with the pop-up menu. An example of creating a new property by right-clicking a section is seen in Figure 12.

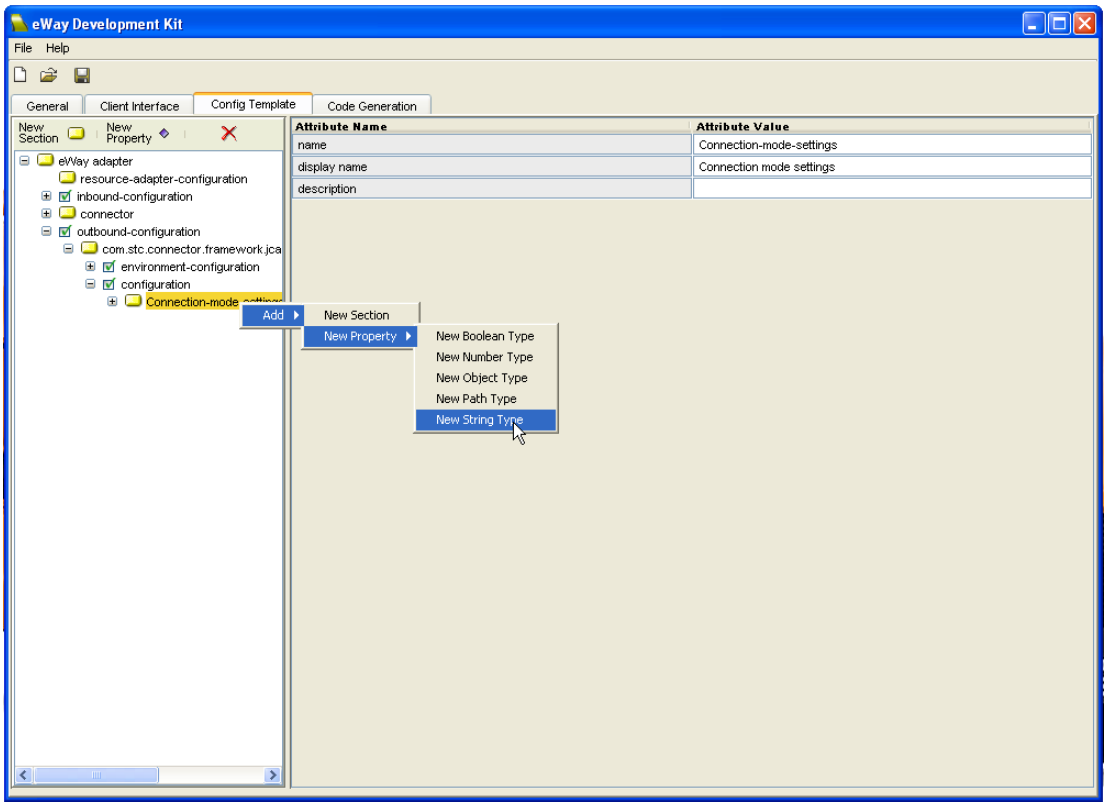
Note: *Enter only those sections that the eWay supports. For example, if the eWay only supports inbound, then do not enter sections for outbound. You can disable sections that are not supported, (see* **Deleting Sections and Properties** *on page 39, or* **Disabling and Enabling Sections** *on page 39 for more information).*

4.5.1 Naming Restrictions:

The following standard Java Identifier naming conventions apply to the Section and Property names:

- **Alphabetic letters** – names should only contain alphabetic characters, such as "AA" or "aa", or the underscore.
- **Digit** – the first position of the name should not contain a digit.

Figure 12 Config Template Tab - Adding a New Property



New Sections contain the following default Attributes:

Table 5 Default Section Attributes

Section	Value
name	The name is used internally to identify a configuration section. Attribute names should follow standard Java naming conventions.
display name	The name that appears on the Connectivity Map and the Environment Explorer.
description	The description of the section that appears in Enterprise Designer.

New Parameters contain the following default Attributes:

Table 6 Default Parameter Attributes

Section	Value
name	The name is used internally to identify a configuration property. Attribute names should follow standard Java naming conventions.

Table 6 Default Parameter Attributes

Section	Value
display name	The name that appears in the Connectivity Map or the External Properties window in Enterprise Explorer.
description	The description of the section that appears in Enterprise Designer.
default	The default value for the parameter. Use the **is choice*** field to add a list of default values for this property.
is choice*	Determines if a choice is possible for the default value. To enter a new choice attribute: 1 Select True from the **is choice*** attribute value. The Attributes window appears. 2 Enter a choice value and click **OK** to close the window, or click the **Tab** button to enter another choice value. To review the updated choice values, click the **default** attribute value field. A drop-down list appears with your new default values.
is choice editable	Determines if the choices entered for the **is choice*** attribute values are editable at design time.
type	Refers to the data type of the parameter (see below for more details).

New Parameters can be of the following type:

Table 7 New Parameter Types

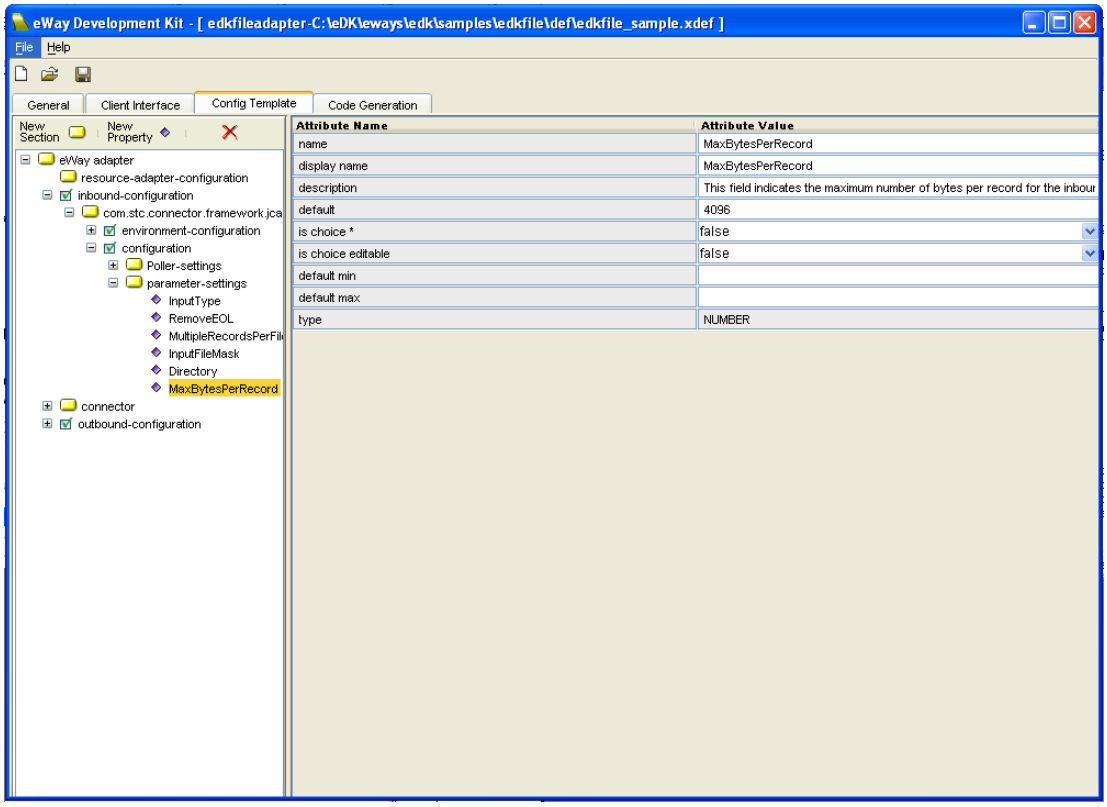
Section	Additional Attribute Values
Boolean (BOOLEAN)	
Number (NUMBER)	Includes the additional default attributes: ▪ **default min** – the default minimum value. ▪ **default max** – the default maximum value.
Object (OBJECT)	
Path (PATH)	
String (STRING)	Includes the additional default attributes: **is encrypted** – determines if the string is encrypted. The encrypted flag is used for password fields. It is recommended not use default values for these fields.

4 Highlight and change the Attribute-Value for each new section, then enter **Return** to save your changes.

You can delete certain inbound or outbound sections by clicking the Delete icon. The Delete icon is disabled for default sections required in the eDK eWay.

Figure 13 illustrates the creation of the parameter-settings section that contains a property called "MaxBytesPerRecord" in the EDKFILE sample eWay.

Figure 13 outputParameter set in inbound-configuration



4.5.2 Connectivity Map Configuration Sections and Properties

Connectivity Map configuration properties added to the **Config Template** tab appear in the eDK eWay's Connectivity Map **Properties** window in Enterprise Designer.

Inbound and Outbound Configuration Parameter Settings

Configuration parameter settings can include both inbound and outbound.

Inbound Configuration properties—and subsequent sub-sections—to a Connectivity Map link associated with an inbound eWay, are located under the configuration template's inbound configurations subsection at:

```
root > inbound-configuration >
com.stc.connector.framework.jca.system.STCActivationSpec >
configuration
```

Outbound Configuration properties—and subsequent sub-sections—to a connectivity map link associated with an outbound eWay, are located under the configuration template's outbound configurations subsection at:

```
root > outbound-configuration >
com.stc.connector.framework.jca.system.
STCManagedConnectionFactory > configuration
```

4.5.3 External System Sections and Properties

External System properties added to the **Config Template** tab appear in the eDK eWay's External System **Properties** window in the Environment Explorer tab of Enterprise Designer.

Inbound and Outbound External System Parameter Settings

External System parameter settings can include both inbound and outbound.

Inbound Configuration properties—and subsequent sub-sections—to an External System, are located under the configuration template's inbound environment-configurations subsection at:

```
root > inbound-configuration > com.stc.connector.framework.jca.
system.STCActivationSpec > environment-configuration
```

Inbound Configuration properties—and subsequent sub-sections—to an External System, are located under the configuration template's outbound environment-configurations subsection at:

```
root > outbound-configuration >
com.stc.connector.framework.jca.system.
STCManagedConnectionFactory > environment-configuration
```

4.5.4 Deleting Sections and Properties

Delete unwanted sections from the Config Template tab by either right-clicking the target section and selecting **Delete** from the pop-up menu, or by clicking the Delete icon located in the tool bar.

Note: *Only user-define sections and properties can be deleted.*

4.5.5 Disabling and Enabling Sections

Depending on the eWay requirements, you may also choose to disable certain sections. Disabling a section on the Config Template tab hides the details of that section during code generation. Disabled sections do not appear in the eDK eWay's External System and Connectivity Map Properties window in the Enterprise Designer.

A small green checkbox icon appears beside sections that can be disabled on the configuration template tree. Sections that are disabled appear with an unchecked icon and also contain an asterisk after the section name.

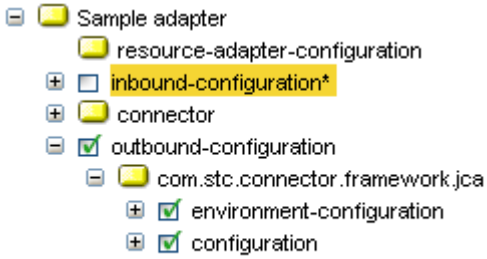
Any section that is disabled can also be enabled. Enabling a disabled section restores the full functionality to that section, including any additional sections or properties that were previously added.

To disable a section on the configuration template tree:

1. Right-click on a section, as for example the "inbound-configuration" section. A pop-up menu appears next to the section.

2 Click **Disable**, and click **Yes** on the Confirm Deletion window that appears. The disabled section appears collapsed and displays an unselected checkbox icon next to it.

Figure 14 Disabled inbound-configuration Section

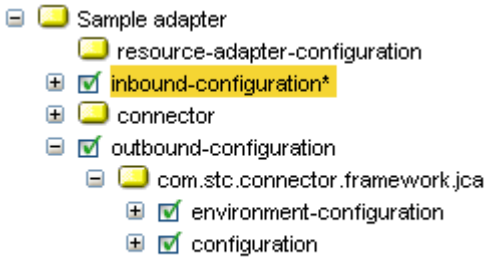


To enable a section on the configuration template tree:

1 Right-click on a previously disabled section. A pop-up menu appears next to the section.

2 Click **Enable**. A selected checkbox icon appears next to the enabled section.

Figure 15 Enabled inbound-configuration Section



4.6 Run the Code Generator

The Code Generation tab on the eWay Development Kit is used to generate the eWay code at a specified location.

To generate eWay code:

1 Select the **Code Generation** tab on the eWay Development Kit.

2 Browse to an output directory. This is the location where eWay code will be generated, (see **eWay Components Created After Generation** on page 51).

3 Enter or browse to an output directory, and then select a file in the folder to select the output location for the log file.

4 Click **Save eWay** to save the eWay definition.

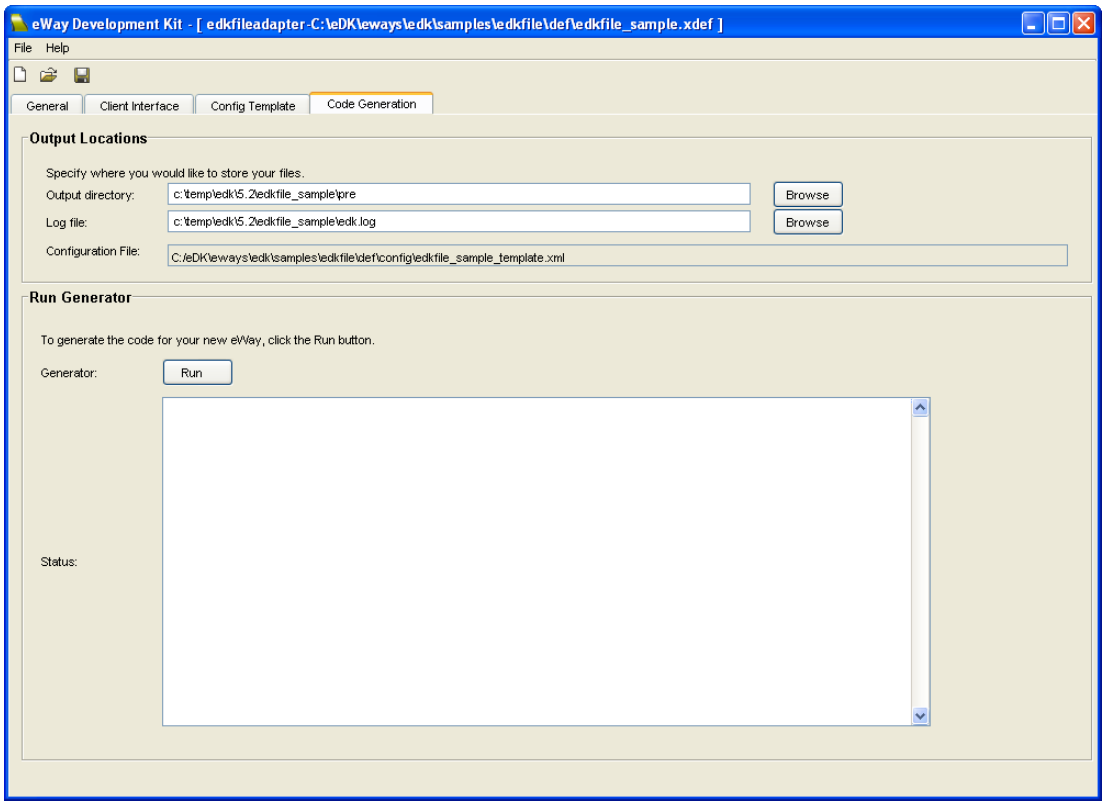
5 Click **Run** from the Run Generator frame.

Note: *You must enter an external application name to generate eWay code. You will also be prompted to save the eWay definition files (.xdef and template .xml) before running the program.*

Note: *You can also choose to generate code by command line. For more information, see* **Generating eDK Code by Command Line** *on page 75.*

Figure 16 displays an example of the completed Code Generation tab in the EDKFILE sample eWay.

Figure 16 Generator Output Tab of the eWay Development Kit Build Tool



During code generation, the eDK Build Tool creates two new folders containing eWay shell code in the specified output directory:

- **connectors folder** – contains the J2EE connector code.
- **eways folder** – contains the eWay artifacts required to plug into Java CAPS.

Both the connectors and eways folders are explained in greater detail in **Implementing Your eWay** on page 43.

4.7 Saving Your Work

Save your eWay definition by clicking **Save eWay** or **Save eWay As** in the file menu, or by clicking the **Save eWay** icon.

Saving creates the following files:

- **<adapter_name>.xdef** – represents the eWay definition file that stores metadata information about the eWay. For more information on the fields generated in this file, see **eDK Definition File** on page 76.
- **<adapter_name>_template.xml** – represents the configuration template which is a hierarchical-based model that contains configuration parameters for the eWay. For more information on the parameters described in the configuration template, see **Connectivity Map Configuration Sections and Properties** on page 38 and **External System Sections and Properties** on page 39.

4.8 Opening Previously Saved Work

To open previously saved work, click **Open eWay** in the file menu, or click the **Open eWay** icon, then browse and select a previously saved **<adapter_name>.xdef** file.

Note: *Make sure the* **.xml** *file is available in the same location as referenced in the* **.xdef** *file.*

Note: *Make sure the third-party JAR files are in the same locations as referenced in the* **.xdef** *file.*

4.9 Choosing a Working Directory

As an optional step, you can choose to either work from the output folder, or a new working directory by doing the following:

1. Copy the **<newEway>adapter** folder, located under the **connectors** folder in the output directory (as specified in the definition file) to:

   ```
   <%STC_ROOT%>\connectors\
   ```

Note: *<%STC_ROOT%> refers to the Environment variable for the root directory. See* **Setting The Implementation Environment and Starting the eDK Build Tool** *on page 14* *for details on setting your classpath.*

2. Copy the **<newEway>adapter** folder, located under the **eways** folder in the output directory (as specified in the definition file) to:

   ```
   <%STC_ROOT%>\eways\
   ```

Chapter 5

Implementing Your eWay

This section describes how to complete implementation of eDK based eWays after eDK code generation.

The goal of the eWay Development Kit is to lighten the workload and expedite the eWay creation process. It does this by reducing the amount of coding required, and by injecting as much complete implementation as possible into difficult areas.

Any additional work required to complete a generated eWay is limited to the implementations needed for connectivity to the underlying EIS system, and to the Java methods or eInsight Business Process operations defined in the eDK Build Tool. Such interactions are specific to a particular EIS system, and cannot be completed by the build tool. You can incorporate additional features into the generated eWay code, such as Transaction support, but these features are not required.

Information on any additional implementation requirements are noted in the following sections.

What's in this Chapter

5.1 Steps to Implement the eWay and Build the .sar File

The following steps describe how to modify and implement the eWay shell code generated by the eDK Build Tool. Additional details on what class files may require implementation are covered in subsequent sections.

1. Browse to the **<newEway>adapter**, located under the **connectors** folder in the output directory to:

   ```
   <%STC_ROOT%>\connectors\
   ```

2. Modify the necessary class files for your specific eWay implementations. For more information, see **connectors Folder** on page 44 and **eways Folder** on page 50.
3. Browse to the **<%STC_ROOT%>\connectors\<newEway>adapter** folder, and run the following:

```
ant clean install -f connector-build.xml
```

This should build the **<newEway>.rar** file, and all the other required **.jar** files at the following locations:

```
<%STC_ROOT%>\BUILD\Modules\connectors\lib\<newEway>.rar
This is a JCA 1.5 compliant .rar file.
<%STC_ROOT%>\BUILD\Modules\connectors\lib\<newEway>_jca10.rar
This is a JCA 1.0 compliant .rar file.
```

4 Build the **.sar** file by running the Apache Ant. To do this:

A Browse to eWay working directory that you defined in **Choosing a Working Directory** on page 42.

B Run the following:

```
ant clean install -f eway-build.xml
```

This creates the new **<newEway>adapter.sar** file in the following location:

```
<%STC_ROOT%>\BUILD\images\products\<newEway>adapter.sar
```

5.2 eWay Folders Created After Generation

During code generation, the eDK Build Tool creates two new folders in the specified output directory that contains eWay code:

- **connectors folder** – located at `<OUTPUT_FOLDER>\connectors`
- **eways folder** – located at `<OUTPUT_FOLDER>\eways`

All implementatoin required to complete the eWay is in the **connectors** folder. You can also add additional features to the **eways** folder, but this is not required.

Note: *The build tool creates the "src_jca15" folder only if inbound operations are defined.*

5.2.1 connectors Folder

The connectors folder contains the J2EE connector code. The build tool creates a subfolder under connectors that is named after the eWay name you enter in the General tab.

If you do not enter an eWay name, then the convention is to convert the external application name all to lowercase letters with the word "adapter" appended. For example, the build tool generates the folder name "myewayadapter" if the External Application Name is "MyeWay".

The following folders are created under the eWay name subfolder:

- **connector-build.xml** – this is the Apache Ant build script to build the eWay connector components.

- **src** – this folder contains all the implementations required to build outbound eWay connector components.
- **src_jca15** – this folder contains all the implementations required to build inbound eWay connector components.
- **thirdpartylib** - this folder contains all the third-party **.jar** files that you specify in eDK Build Tool.

As mentioned earlier, the built tool generates as much implementation as possible, so you only need to focus on application specific details. The build tool also marks the places in the generated source code where implementation is needed, by using the following tags:

```
// <Start_User_Code>

and

// <End_User_Code>
```

You are required to put application specific implementations between these tags. We do not recommend changing implementations elsewhere in the generated source files without in depth knowledge of the eWay architecture. Inadvertent mistakes could have substantial consequences.

Listed below are the classes that require further implementation:

Implementation Required in the "src" Folder

The following class files are found in the src folder:

<external_application_name>EwayConnection

This class represents the physical connection to the underlying EIS system and implements interfaces required to handle connection management in a resource adapter. Connection management includes logic to establish, match and close connections from the external system. This class also allows you to specify any Resource and Transaction management implementation. The build tool does not automatically generate Resource and Transaction management implementations.

Note: *Although this class is responsible for physical connection management, the actual connection logic implementation is delegated methods in the <external_application_name>ClientApplicationImpl class.*

<external_application_name>ClientApplicationImpl

This class represents the OTD exposed in the Java Collaboration Definition (JCD). It contains the actual implementations required to establish and close connections to the external system, along with implementations required for all the outbound Java Interface methods specified in eDK Build Tool. It is important to understand how to interact with the underlying EIS, and make good use of internal APIs provided by the external application. Accessor methods—Bean-like getter and setter methods—are generated for each attribute defined in the Java Interface in eDK Build Tool.

We recommend making good use of any of these exposed attributes in the JCD. You can also add additional member variables and methods in this class to facilitate method implementations if and when they are appropriate.

You are still required to implement the connection handling related methods in this class, even if no Java Interface methods are defined in eDK Build Tool.

Note: *The connect() and disconnect() methods are located in the <external_application_name>EwayConnection class in 5.0.5 eDK eWays, they are now moved to this class. You need to make sure that the connection handling logic is implemented in this class if the eDK eWay is recreated with the current version of the build tool. You can use the same code as long as it still applies.*

<external_application_name>WebClientApplication

This class represents the Java implementation for all the outbound webservice—eInsight Business Process—operations specified in the BPEL Interface in the eDK Build Tool.

If no operations are specified in eDK Build Tool, then no changes are required or recommended in this file.

As with Java Interface methods, we recommend understanding how to interact with underlying EIS. Bear in mind that webservice operations are considered sessionless operations, which should have exactly one input parameter that is enforced by the build tool. We do not recommend making modifications to this file.

The type of the input parameter connected to the operation corresponds to a Java Bean class created in the same package, with the following exceptions:

- The method declaration clearly shows if you specify a User Defined Data Type as the input parameter type to the operation. However, choosing one of the built-in types in the drop-down list for the input message to the operation creates a Java Bean class in the same package to contain the built-in type. In this case, the name of the Java Bean class is *<operation_name>Input*, and the type definition of the input parameter to the operation in this class matches the name of the Java Bean class.

  For example: If the input message type is **java.lang.String** for operation **write**, then a Java Bean class of name **writeInput** is created in the same package, and the type definition of the input parameter in the method declaration of **write**() is **writeInput()**.
- The Same behavior is assumed when the input parameter is a collection. The same convention is also used for the output message. We do not recommend changing any method declarations without first understanding the intricacies of how an eWay works in the Java CAPS Business Process editor.

  Method declarations and the generated Java Bean classes must match the generated WSDL definition to allow successful xpath creation when the eWay participates in Java CAPS eInsight Business Process Collaborations.

Note: *Although the words "operation" and "method" are used interchangeably in other contexts, our convention is to use "method" to denote methods exposed in Java*

Collaboration Editor, and "operation" to denote eInsight Business Process operations in this build tool.

Java Bean classes for User Defined Classes or Data Types:

The build tool generates a cooresponding Java Bean class—with accessor methods that follow the Java Bean paradigm for each attribute defined—for each User Defined Class or User Defined Data Type.

With some exceptions, the build tool generates Java Bean classes—for the User Defined Classes defined in Java Interface— in the same package as the **<external_application_name>ClientApplicationImpl** class.

A User Defined Data Type is mostly used for eInsight Business Process operations. It serves as the data container, implements the required interfaces for data persistence and recovery, and allows for proper xpath expression lookups at eWay runtime.

The Java Bean classes generated for User Defined Classes are more flexible, and allow for additional user methods. You must take care to implement any defined user methods.

It is also important to note that eInsight Business Process (webservice operations) are by design sessionless operations, so each operation itself assumes:

- The establishment of connection, if applicable
- Some type of interaction with the underlying EIS
- Closing of the connection

No duplicate implementation is required in this class since the actual connection establishment and closing logic is implemented in the **<external_application_name>ClientApplicationImpl** class, and is "reused" at eWay runtime. You only need to need to implement the business interaction logic.

Note: *Java classes generated for User Defined Data Types do not need any additional implementation. In fact we recommend that you do not modify the generated implementation without in depth knowledge of it. The same applies for the Java classes generated for User Defined Classes, except when methods are defined.*

<external_application_name>AlertCodes

This class implements the required AlertCodeMap interface, and defines all the eDK eWay Alert Codes. Although all required eWay status codes are already generated, you can define additional Alert Codes in this class for application specific Alerts. The important thing to know is that for each additional Alert Code defined in this class, the value of the static final member variable is actually a key in the resource bundle file for all custom Alert messages.

Please refer to**Adding and Sending Custom Alert Messages** on page 64 for more details on how to add custom Alert messages.

Other classes

Other classes generated in the **src** folder do not require further implementation. We do not recommended modifying any existing implementation without having in-depth knowledge of these class files.

One class worth mentioning is the **<external_application_name>Configuration** class, which contains accessor methods for all outbound user configuration properties.

Implementation Required in the "src_jca15" Folder

The **src_jca15** folder is only created if you define an inbound operation in the eDK Build Tool. The following class files are found in the **src_jca15** folder:

<external_application_name>EwayEndPoint

This class represents an eWay endpoint created by the Resource Adapter upon endpoint activation. The main methods that require further implementation are **activation()** and **deactivation()**.

The **activation()** method is the main entry point to the eDK eWay inbound processor. You should start child work threads to handle inbound messages in this method.

The **deactivation()** method is the main method to deactivate an eWay endpoint and cleanup resources. This class also allows you to specify any Resource implementation, although the eDK does not automatically generate Resource management implementations.

All inbound operations are by default webservice operations, and therefore require corresponding operation definitions in the eWay WSDL. The eWay WSDL is automatically generated by build tool, and it serves as the mapping for the defined inbound operations to work properly in Java CAPS. So, unlike the outbound operations, there are no Java classes generated that contain the "inbound" method declarations. Instead, for each inbound operation defined in either Java Interface or BPEL Interface, there is a corresponding **onContent()** method defined in the **<external_application_name>Listener** interface with the proper input parameter. Please read on for more details on how to implement inbound operations.

<external_application_name>InboundWork

This class represents a work thread that triggers the Message Driven Bean. It is always generated when inbound operations are defined. The main method that requires further implementation is the **run()** method. With this method, you can construct an inbound message container, populate it with proper values depending on the application, and trigger the Message Driven Bean that eventually routes the inbound communication to the processing Collaborations.

Note: *Step-by-step in-line comments are included in the generated Java source file.*

<external_application_name>InboundTimerWork

This class represents a task thread and is used by the Java Timer to schedule its work. The main method that requires further implementation is the **run()** method. With this method, you can create and schedule additional work threads to do the actual work for inbound communication. This class is always generated when inbound operations are defined. You can use it to schedule tasks, such as polling for a particular file based on a predefined time interval.

<external_application_name>Listener

This class represents the interface defined for all inbound operations defined in the Java and BPEL Interface in the eDK Build Tool. This interface is implemented by eDK eWay Message Driven Bean. As mentioned earlier, unlike outbound operations, there are no actual Java method declarations defined in this interface that maps to the inbound operations defined in eDK Build Tool. Instead, the inbound operations serve as the "receive" activity in the eInsight Business Process Designer or as the trigger webservice operation in the Java Collaboration editor and therefore are defined in the generated eWay WSDL file respectively. What this interface defines is a potentially a list of **onContent()** methods which takes the input parameters that corresponds to the proper input message definition in the WSDL. We recommend implementing the appropriate logic to interact with the underlying EIS for the inbound communication, and populate the input message container accordingly in the work threads before triggering the Message Driven Bean. Step-by-step in-line comments details what implementations are needed to achieve this.

Other classes

Other classes generated in the **src_jca15** folder do not require any further implementation. We do not recommend modifying the existing implementation without having in-depth knowledge of these class files.

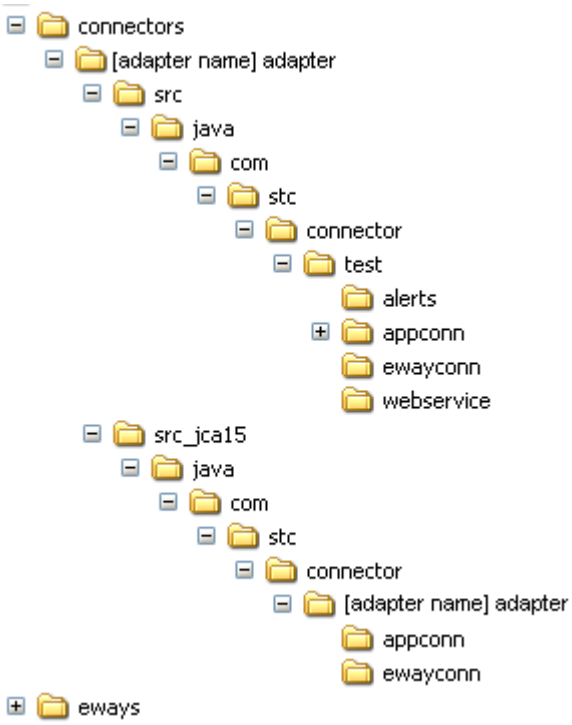
One class worth mentioning is the **<external_application_name>InboundConfiguration** class, which contains accessor methods for all inbound user configuration properties.

Note: *Although it is absolutely valid to have a Request-Response type of inbound operation (an inbound operation defined with both input and output messages), the Connectivity Map in the Java CAPS 5.1.2. Enterprise Designer currently has limited support for them.*

Even though the eDK Build Tool allows you to define Request-Response types of an inbound operation, you may experience problems with this feature. We recommend steering away from these types of inbound operations until future releases of Java CAPS.

There is a work-around to achieve the request response type of communication for inbound connectivities (only available in a Java CAPS Collaboration Definition), by using a User Defined Classes. You can define a User Defined Class as the input message to the Java Interface inbound operation, and take advantage of the methods defined in the User Defined Classes to allow response type of communication to the external system. Please refer to the included TCPIPServer sample as an example.

Figure 17 Files Found in the connectors Folder

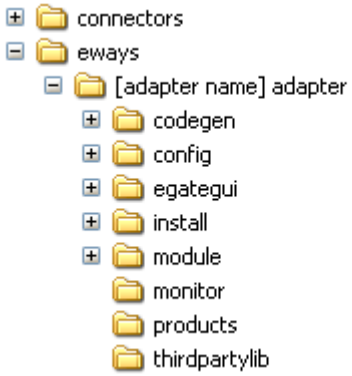


5.2.2 eways Folder

The **eways** folder contains the design-time artifacts required to plug into the Enterprise Designer. It also contains Java CAPS code generation components, including codelets, run-time components, installation descriptors, and other resources.

- **codegen** – contains all the codelets and runtime Enterprise Java Beans.
- **egategui** – contains all the GUI plug-in code for various Enterprise Designer editors.
- **config** – contains the configuration template that merges default eWay configurations and user configurations.
- **install** – contains the WSDL file, **descriptor.xml** for eway installation, and also logging and alerting resource bundles.
- **module** – contains the **manifest.mf** and **layer.xml** files for the eWay NetBeans module.
- **monitor** – contains eWay monitoring metadata properties and build script properties to enable eWay monitoring plug-ins in the Enterprise Manager.
- **products** – contains product and dependency **.xml** files required for the eDK eWay product list **.sar** file.
- **Thirdpartylib** – contains all user specified third-party **.jar** files.

Figure 18 Files Found in the eways Folder



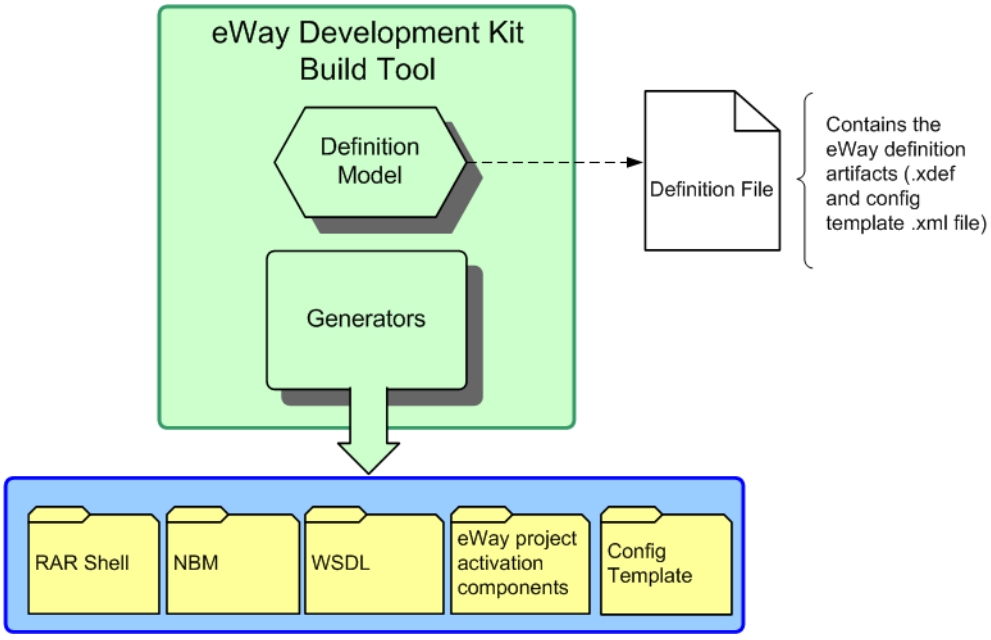
5.2.3 eWay Components Created After Generation

The code generated by the eDK includes the following key components:

- J2EE Connector Architecture resource adapter
- GUI code for plugging into the Enterprise Designer
- Code Generation components
- Runtime EJB components

All of these components are packaged in the eWay **.sar** file that can be uploaded into a Repository. The following diagram illustrates the shell code components created after code generation.

Figure 19 Shell Code Created After Generation



J2EE Connector Architecture Resource Adapter

The J2EE specification includes a Connector Architecture specification defining the component which is used to interact directly with external applications (also referred to as Enterprise Information Systems - EISs). This component is normally referred to as a J2EE connector or Resource Adapter (RA).

Sun developed a framework for developing these resource adapters. The classes generated in the connectors folder by the eDK are based on the Sun SeeBeyond RA framework.

Figure 19 illustrates how Resource Adapter Archive (**RAR Shell**) Code fit into the eDK component process flow.

NetBeans Module to Plug into the Enterprise Designer

The GUI plug-in includes NetBeans code that runs in the Project Explorer, the Java Collaboration Wizard, the Connectivity Map editor, and the Deployment and Environment editors. You do not need to modify any GUI code generated by the eDK .

Figure 19 illustrates how **NBM**, **WSDL**, and **Config Template** fit into the eDK component process flow.

Code Generation components called during deployment

All Java CAPS products provide components to allow Project code generation in eDesigner. These components are referred to as Codelets, and are Java classes which are executed during activation (when a user clicks on Activate in the Deployment editor). Codelets are responsible for generating artifacts that get packaged during project deployment. In Java CAPS, Project deployments come in the form of **.ear** files. eWay codelets primarily package its J2EE resource adapter (**.rar)** file, a Message-driven Bean (MDB) implementation (if inbound) and the associated deployment descriptors it generated based on project properties.

Figure 19 illustrates how **Codelets** fit into the eDK component process flow.

5.3 Suggested Conventions for Writing JNI Code

The Java Native Interface (JNI) is a native programming interface that allows Java code running inside a Java Virtual Machine (JVM) to invoke platform specific code that runs outside the JVM.

Using JNI code requires several steps at the various stages of eWay development, usage, and runtime.

Steps required during the JNI development phase include:

1. Write the JNI code using the native code and compile it into an OS specific native format, such as **.dll** for Windows, or **.so** for Solaris.
2. Create a thin Java wrapper to invoke the JNI code, and then build and package it in a separate **.jar** file.
3. Package both the native library and the **.jar** file created earlier, into a **.zip** file.

4 Modify the **eway-build.xml** to ensure that the **.zip** file is added to the **.sar** file.

5 Modify the **install.xml**, located under:

```
eways\[adapter name]\install
```

Add the following code:

```
<taskdef name="UserDownloadable"
classname="com.stc.installer.UserDownloadableInstallTask" />

<UserDownloadable downloadableName="<DownloadableModuleName>"
file="${basedir}/<TheZipFile>.zip" repDir="InstallManager/50Base/
<eWayExternalName>/" repURL="${stc.rep.url}"
distURLBase="${stc.module.distURLBase}" />
```

This adds the **.zip** file as a downloadable object in Java Composite Application Platform Suite Installer.

6 Download the **.zip** file to a well-known location. You must configure the Integration Server to be aware of the JNI code.

7 Update the path to the JNI code. The path is operating system dependent and is easiest done in the command window right before deploying your eWay Project.

For Windows, update your **PATH** variable to include the path to the JNI code. For Solaris, update your **LD_LIBRARY_PATH** variable to include the path to the JNI code. Refer to your specific OS documentation for setting up other libraries.

Note: *An example of a JNI based eWay appears in* **Appendix C**.

Chapter 6

eDK eWay Concepts and Best Practices

This chapter describes some of the concepts used to successfully create eDK-based eWays.

What's in this Chapter

6.1 Establishing Connections to the EIS

An Enterprise Information System (EIS) is an application or system that provides a set of services to an enterprise system for accessing, manipulating, and managing information.

The eDK supports establishing connections to the EIS both automatically and dynamically. The code that is generated by the eDK includes an Enterprise Java Bean (EJB) which calls the **getConnection()** method on the eWay resource adapter for obtaining a connection from the integration server managed connection pool. The **getConnection()** method implementation is delegated to the implementation of the **EwayConnection** interface.

6.1.1 Automatic Connection Establishment Mode

Automatic mode means a connection to the external system is established when the eWay is initialized.

6.1.2 Dynamic Connection

In a dynamic connection (manual mode), the connection is not established when the **initialize()** method is called. Instead, the eWay end-user must explicitly call a method from the Java Collaboration Definition.

Overriding Configurations in JCD

It is possible to override default configuration values at design-time. This is achieved by calling the appropriate setter methods in the configuration Bean class. You can get an instance of the config Bean class via the appropriate getter method in the OTD.

```
<eWay>ClientApplicationImpl.java

    /**
     * Returns the testConfiguration object.
     *
     * @return the testConfiguration instance.
     */
    public testConfiguration getEwayConfiguration() {
        return appConn.getEwayConfiguration();
    }
}
```

6.2 Specifying Configuration Properties

eDK based eWay configuration properties are created on the Config Template tab of the eWay Development Kit Build Tool. The type of eWay you develop determines the type of inbound or outbound configuration properties added to the eWay.

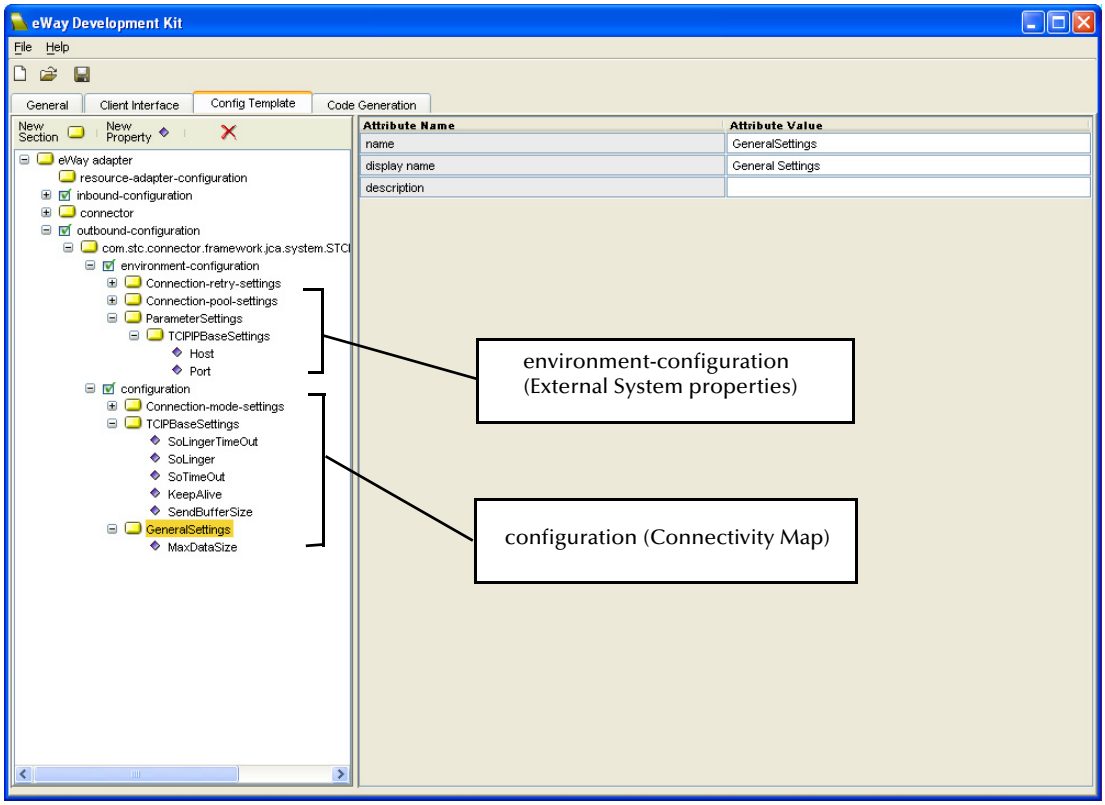
eWay configuration properties created on the Config Template tab are exposed to the Java CAPS user in the Enterprise Designer via:

- **Connectivity Map link**
- **External System Properties**

The Config Template is a superset template that contains a number of designated sections under which configurations for the Connectivity Map link and configurations for the External System properties are specified.

Figure 20 illustrates how configuration settings on an eWay (TCPIPClientAdapter in this example), contain the base settings for both the configuration (Connectivity Map) and environment-configuration (External System properties).

Figure 20 Config Template

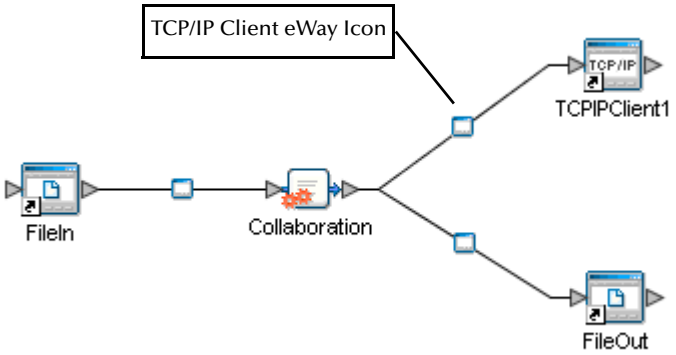


6.2.1 Connectivity Map eWay Properties

To access the Connectivity Map properties:

1. Open the Enterprise Designer – Connectivity Map Editor for the eDK based eWay that you created.
2. Double-click the eDK based eWay icon to access the Properties window.

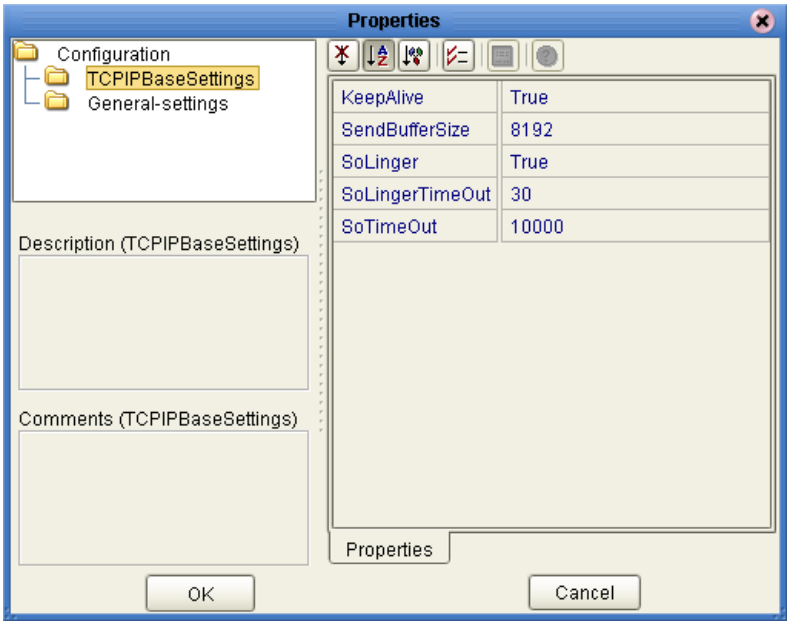
Figure 21 Connectivity Map



3. The eWay Properties window appears with the sections and parameters defined under the configuration section of the Config Template in the eDK Build Tool. Note

that the configuration properties should only provide properties that are independent of the external system's physical location.

Figure 22 Connectivity Map Properties – TCPIPClientAdapter

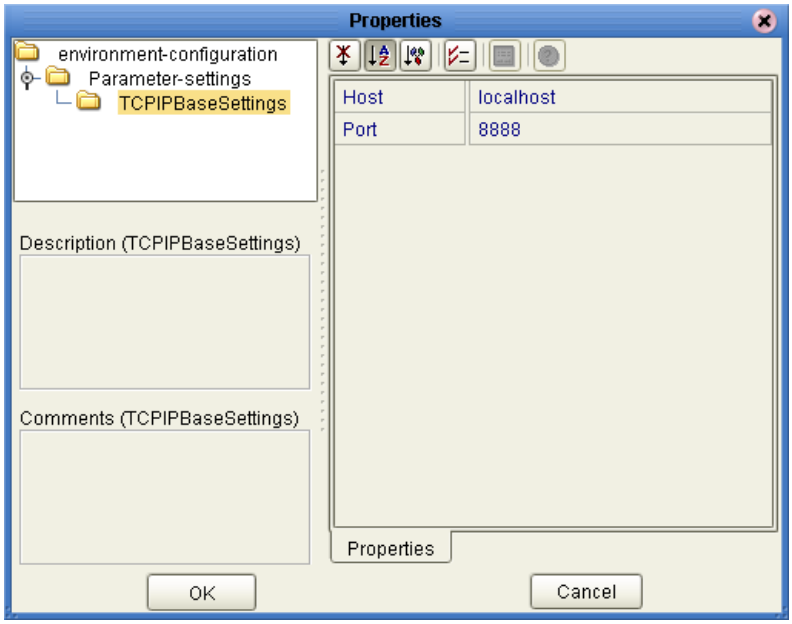


6.2.2 External System Properties

To access the external system properties:

1. Open the Enterprise Designer – Environment Explorer for the eDK based eWay that you created.
2. Right-click the eDK based eWay (TCPIPClientAdapter in this example), and select **Properties**. The Properties window opens to the environment-configuration properties. Note that physical connection properties are normally provided here.

Figure 23 Environment Explorer Properties – TCPIPClientAdapter



6.3 Wrapping Third-Party .jar Files

The Java client interface allows specification of attributes and methods in third-party classes. While it is possible to refer directly to third-party class files, we recommend creating wrapper classes that the Java Collaboration Editor client interface can reference. Encapsulating third-party class files in wrapper classes eliminates any direct interaction, ensuring future flexibility when newer versions of the third-party class files introduce incompatible method signature changes.

6.4 Source Control

The eDK does not have build-in source control support. We suggest using a third-party source control system to ensure adequate source control.

When working with a source control system, try using a separate directory as the target for the stub code from the generator (i.e. the output directory on the generation tab of the eDK Build Tool), and another to implement the methods and operations defined for the OTD. These can also be thought of as the “pre” and “post” implementation directories.

Check eWay code into the source control system after generating for the first time. You can then check the code out to directories where the stub methods and operations are to be implemented. After this, follow the usual source control operations and procedures from the implementation directories.

If modifications to the OTD are required after implementing the code:

1. Delete all of the files in the current output directory and then regenerate the stubs.
2. Merge the new stub code with the existing files and code in the implementation directories.
3. Check your merged code back into the source control system. You can now use the implementation directories to build the eWay **.sar** file.

6.5 Generating Javadocs

You can generate Javadocs for both the "connectors" folder and the "eways" folder after completing eWay implementation.

To generate Javadocs for the connectors folder:

1. From the command window, browse to the eWay adapter under the connectors folder.

```
<%STC_ROOT%>\connectors\<Adapter Name>adapter
```

2. Run the following to generate Javadocs for the connectors folder:

```
ant -f connector-build.xml build-javadoc
```

3. Browse to the following to access the generated Javadoc:

```
<%STC_ROOT%>\BUILD\Modules\connectors\<Adapter
Name>adapter\javadoc
```

To generate Javadocs for the eways folder:

1. From the command window, browse to the eWay adapter under the eways folder:
2. Run the following to generate Javadocs for the eways folder:

```
ant -f eway-build.xml build-javadoc
```

3. Browse to the following to access the generated Javadoc:

```
<%STC_ROOT%>\BUILD\Modules\eways\<Adapter Name>adapter\javadoc
```

Chapter 7

Sample eDK eWays

This chapter describes the sample eWays included in the eWay Development Kit.

What's in this Chapter

7.1 About the eDK Samples

Four eDK samples are included in the **eWayDevelopmentKit.sar** file. These samples are designed to provide a general understanding of how to build an eWay using the eWay Development Kit Build Tool.

With the exception of the **JNISample** found in **Appendix C**, we do not describe how to re-create the eWay samples included with the eDK with step-by-step instructions, simply because there is very little value for you to recreate these samples using our implementation.

Instead, we provide files in a "pre" and "post" implementation state. Review these "pre" and "post" implementation files with a file comparison tool. You can implement the eWay in whatever manner you choose. The choice of implementation we provide in the "post" folder is only an example, and should not be taken as the de facto procedure when creating eDK based eWays. For practice, try creating your own implementation of these samples.

Each eWay sample contained in the sample eWay folders includes the following:

- An **.xdef** file is the eWay Definition that is created in the build tool. You should load this file first in the build tool.
- A **pre_implementation_<Sample Name>.zip** file that includes all the files created after code generation, but before implementation. You can recreate this file by running the code generator using the **.xdef** file for each sample.

- A **post_implementation_<Sample Name>.zip** file that includes all the files and components created after implementation. This is our implementation of the eWay.

7.1.1 Sample Projects Packaged in the eDK

eDK eWay samples packaged in the eDK include:

- File eWay inbound/outbound example, located under:

  ```
  <%STC_ROOT%>\eways\edk\samples\edkfile
  ```

- JNI outbound client example, located under:

  ```
  <%STC_ROOT%>\eways\edk\samples\edkjni
  ```

- TCP/IP outbound client example, located under:

  ```
  <%STC_ROOT%>\eways\edk\samples\tcpipclient
  ```

- TCP/IP inbound server example, located under:

  ```
  <%STC_ROOT%>\eways\edk\samples\tcpipserver
  ```

7.1.2 About the EDKFILE Sample

The **EDKFILE** sample eWay provides a simple inbound/outbound scenario that is similar to the *Sun SeeBeyond eWay™ File Adapter*, which is used to exchange data between an external file system and the Java CAPS.

Additional Files Used to Implement the EDKFILE Sample

Our implementation of the **EDKFILE** sample includes a number of additional files. If you choose to recreate our implementation of the sample **EDKFILE**, then you must include the following files:

- PrintfFormat.java
- counterManager.java

In the following directory:

```
<%STC_ROOT%>\connectors\edkfileadapter\edkfileadapter\src\java\com\st
c\connector\edkfileadapter\appconn\appimpl
```

For inbound implementation, include the following files:

- EDKFILEReaderWork.java
- JZOOWildCardFilter.java

In the following directory:

```
<%STC_ROOT%>\connectors\edkfileadapter\edkfileadapter\src_jca15\java\
com\stc\connector\edkfileadapter\ewayconn
```

7.1.3 About the JNISample

The **JNISample** sample eWay provides a simple outbound scenario that copies a file from one directory to another.

Additional infomation on creating a JNI based eWay appears in **Appendix C**.

7.1.4 About the TCPIPClient Sample

The **TCPIPClient** sample eWay provides an example of how an outbound eWay enables the eGate™ Integrator to communicate with client applications using TCP/IP, and provides real-time data transfer for systems that support TCP/IP.

7.1.5 About the TCPIPServer Sample

The **TCPIPServer** sample eWay provides an example of how to implement inbound eWays. The server listens on a specified port for client connections and can interact with the client, based on the design of the Collaboration. The eWay end user can use the TCP/IP server eWay to implement a TCP/IP communications protocol.

7.2 Import the Samples into the Build Tool

As mentioned earlier, you can import the eDK samples into the build tool.

1. Start the eWay Development Kit Build Tool.
2. Click the Open eWay icon, or choose **File** > **Open eWay** from the file menu.
3. Browse to and select one of the eDK eWay samples under:

```
${env.STC_ROOT}\eways\edk\samples\<eDK Sample>\def\<eDK
Sample>_sample.xdef
```

4. Click **Open**. The sample appears in the eWay Development Kit Build Tool.

7.3 Build the .sar File

Two steps are required to build the **.sar** file for the implemented eWay. The following steps describe how to build a **.sar** file for the **EDKFILE** sample eWay.

Note: *Remember to set your environment, by running the env.bat file, before building your .sar file.*

1. Browse to the **<Working_Directory>\connectors\edkfileadapter** folder, and run the following:

```
ant clean install -f connector-build.xml
```

   This should build the **edkfile.rar** file and all the other required **.jar** files at the following location:

```
<%STC_ROOT%>\BUILD\Modules\connectors\lib
```

2. Browse to the **<Working_Directory>\eways\edkfileadapter** folder and run the following:

```
ant clean install -f eway-build.xml
```

This creates the new **eDK_EDKFILEeWay.sar** file and **Product_List.sar** file in the following location:

```
<%STC_ROOT%>\BUILD\Images\Products
```

7.4 Upload the New eWay to the Repository

The following steps describe how to upload the new **eDK_EDKFILEeWay.sar** file that we created in the previous section, to the Java CAPS Repository.

1. Click the **Administration** tab in the Java Composite Application Platform Suite Installer.
2. Click the **Click to install additional products link**, and then click **Browse** to update the Product List.
3. Locate the **Product_List.sar** file that you created for the **eDK_EDKFILEeWay**, and then click **Submit**.
4. Expand the list of available eWays from the **eWays and Addons** folder, select the **eDK_EDKFILEeWay**, and then click **Next**.
5. Browse to and select the **eDK_EDKFILEeWay.sar** file, then click **Next** to install the new eWay.

7.5 Run the Enterprise Designer Update Center

For detailed information on running the Enterprise Designer Update Center, see the *Composite Application Platform Suite Installation Guide*.

7.6 Create, Build, and Deploy the Sample Projects

The creation and deployment of new eWay Projects created using an eDK based eWay is beyond the scope of this user's guide. Detailed information, including examples of how to create and deploy sample Projects are found in the *Sun SeeBeyond eGate™ Integrator Tutorial*.

Chapter 8

Adding and Sending Custom Alert Messages

This chapter describes how to add and send custom Alerts using the Resource Adapter framework.

What's in this Chapter

8.1 eDK Alerts

In Java CAPS, an Alert is triggered when a specified condition occurs in a Project component. The condition might be some type of problem that must be corrected. For example, an Alert might indicate that a Integration Server is no longer running.

The Enterprise Manager system administration tools handle a number of tasks, including using Alert and log files to troubleshoot problems. The *Sun SeeBeyond eGate Integrator System Administration Guide* includes a chapter that describes how to monitor Alerts using the Enterprise Manager and the command-line client.

Note: *The Sun SeeBeyond Alert Agent User's Guide describes how the Alert Agent can monitor both predefined Alerts and custom Alerts. The "Collaboration Definitions (Java)" chapter in the Sun SeeBeyond eGate™ Integrator User's Guide describes how to create custom Alerts at design time.*

eDK Alerts are managed through the **<new eWay>AlertCodes.java** file, which is located under:

```
<Output-Location>\connectors\<new eWay>\src\java\com\stc\
connector\<new eWay>\alerts
```

Implementation of the <new eWay>AlertCodes class is required to display your eWay specific Alerts. The common Alert codes provided by default in the eDK generated code are:

```
<new eWay>_EWAY_STARTED
<new eWay>_EWAY_RUNNING
<new eWay>_EWAY_STOPPING
<new eWay>_EWAY_STOPPED
<new eWay>_EWAY_SUSPENDING
<new eWay>_EWAY_SUSPENDED
```

You can add custom Alert codes by modifying the **<new eWay>AlertCodes.java** file.

8.2 Adding eWay Specific Alert Message Codes

This section describes how to add custom Alerts using the Resource Adapter framework.

8.2.1 Java Code Changes

Use the following method on the **<external_application_name>EwayConnection.java** class to send Alerts:

```
public void sendAlert (String alertCode,
                       String [] alertMsgArgs,
                       String detailMsg,
                       int notificationSeverity)
```

8.2.2 What to Pass for alertMsgCode and alertMsgCodeArgs

You need to create a properties file using the localization resource bundle naming convention.

As an example:

```
FILE_en_US.properties (for US English),
FILE_fr_FR.properties (French)
FILE_fr_CA.properties (Canadian French)
```

You must place this file in the directory:

```
<output-location>/eways/<new eWay>/install/alertcodes/
```

The file is packaged in the eWay's **.emr** file which you deploy in the Sun SeeBeyond Enterprise Manager.

Note that the file naming convention specifies a string corresponding to your eway name. The same prefix string (e.g. FILE) must match the prefix of the Alert message code variables contained in the Properties file. The contents of the Properties file will be your Alert Code message variable and the Alert message. For more information, see **Installing Alert Code Properties Files** on page 66.

As an example:

```
FILE-REN000001="Unable to rename file {0}"
FILE-WRT000001="Unable to write output file {0}."
```

The placeholders in the above message are: (convention {0}...{1}). These are specified in the alertMsgCodeArgs argument to sendAlert. So, a sample send Alert call using this would look like:

```
String [] args =  { "myfilename" };
String alertMsg = "Unable to rename file " + ... + "Skipping ..." ;
sendAlert(EDKFILEAlertCodes.FILE-ASRENAMEFAILED, args, alertMsg,
com.stc.eventmanagement.NotificationEvent.SEVERITY_TYPE_CRITICAL);
```

To isolate where you update the Alert Codes (first argument above), you can create a class containing the constants corresponding to them.

As an example:

```
public class EDKFILEAlertCodes {

    /**
     * File rename alert
     * <code>Unable to rename file {0}</code>
     * Params:
     *      {0} file name
     */
    public static final String FILE_ASRENAMEFAILED = "FILE-
ASRENAMEFAILED000001";

}
```

The sendAlert call appears as follows:

```
sendAlert(EDKFILEAlertCodes.ASRENAMEFAILED args, alertMsg,
com.stc.eventmanagement.NotificationEvent.SEVERITY_TYPE_CRITICAL);
```

8.2.3 Installing Alert Code Properties Files

Custom Alert codes are installed automatically when the eWay is uploaded to the Repository. You must edit the following file to add custom Alert codes:

```
<output_folder>\eways\<eway_adapter_name>\install\alertcodes\<exte
rnal_application_name>.properties
```

8.3 Sending eWay Specific Alerts

This section describes how to send custom Alerts using the Resource Adapter framework.

For Alerts that are associated with your managed connection, make your EwayConnection class implement the following:

```
com.stc.connector.management.STCManagedSlave
```

This interface includes the **setMonitor()** method, which is called from the associated STCManagedConnection. This provides access to the MBean object reference, which for sending Alerts. Use the following method in the **EwayConnection** class to send Alerts.

```
public void sendAlert(String alertMsgCode,
                      String[] alertMsgCodeArgs,
                      String alertMsg,
                      int severity);
```

Valid values for severity are:

```
com.stc.eventmanagement.NotificationEvent.SEVERITY_TYPE_CRITICAL
com.stc.eventmanagement.NotificationEvent.SEVERITY_TYPE_INFO
com.stc.eventmanagement.NotificationEvent.SEVERITY_TYPE_MAJOR
com.stc.eventmanagement.NotificationEvent.SEVERITY_TYPE_MINOR
com.stc.eventmanagement.NotificationEvent.SEVERITY_TYPE_WARNING
```

The EventManagement API is called by the above method. See theEventManagement module for details on this API.

For Alerts associated with your ActivationSpec, do the same as above for your EwayEndpoint class (implement STCManagedSlave).

Appendix A

J2EE Connector Architecture

This Appendix provides information on the Connector Architecture, Resource Adapter, Sequence diagrams, and Connection Interfaces.

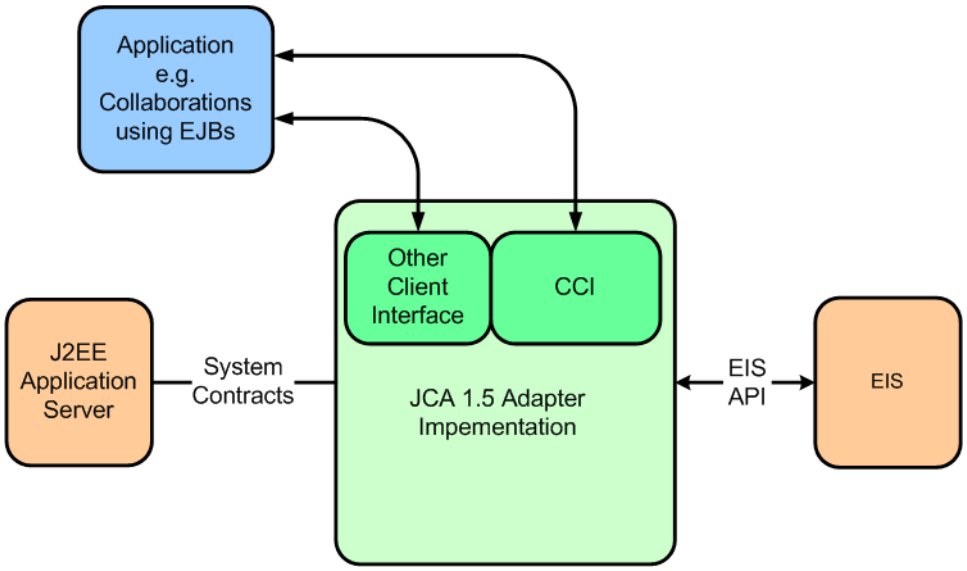
What's in this Chapter

A.1 J2EE Connector Architecture Overview

The J2EE Connector Architecture specifies how to develop resource adapters that are used to interact with Enterprise Information Systems (EIS). It describes the interfaces between the J2EE Application Server and the resource adapter which provide for transaction management, connection management, security management, work management, and life cycle management. These Application server/resource adapter interfaces are also referred to as System Contracts, see Figure 24.

The Connector architecture also describes the client interface to the resource adapters. Figure 24 illustrates allowed client interfaces. The resource adapter client (normally an EJB) is shown interacting with the resource adapter either through the Client Connection Interface (CCI) or the Application Connection (AppConn) interface.

Figure 24 J2EE Connector Architecture Overview



A.2 About the Resource Adapter Framework

Sun developed a Resource Adapter (RA) framework for developing resource adapters based on the J2EE Connector Architecture. The framework is a set of interfaces and abstract classes which simplify the development of resource adapters. The framework also facilitates the development of the resource adapter client interface which is based on the Sun SeeBeyond AppConn interface used by the Java CAPS Collaboration framework.

The eDK generates J2EE Connector code based on the Sun SeeBeyond Resource Adapter (RA) framework. The framework was designed so that development is focused on the definition and implementation of the client interface to be exposed by the eWay.

A.3 Sun RA Framework Class Diagram

The Resource Adapter (RA) framework is best specified in a Unified Modeling Language (UML) class and sequence diagram.

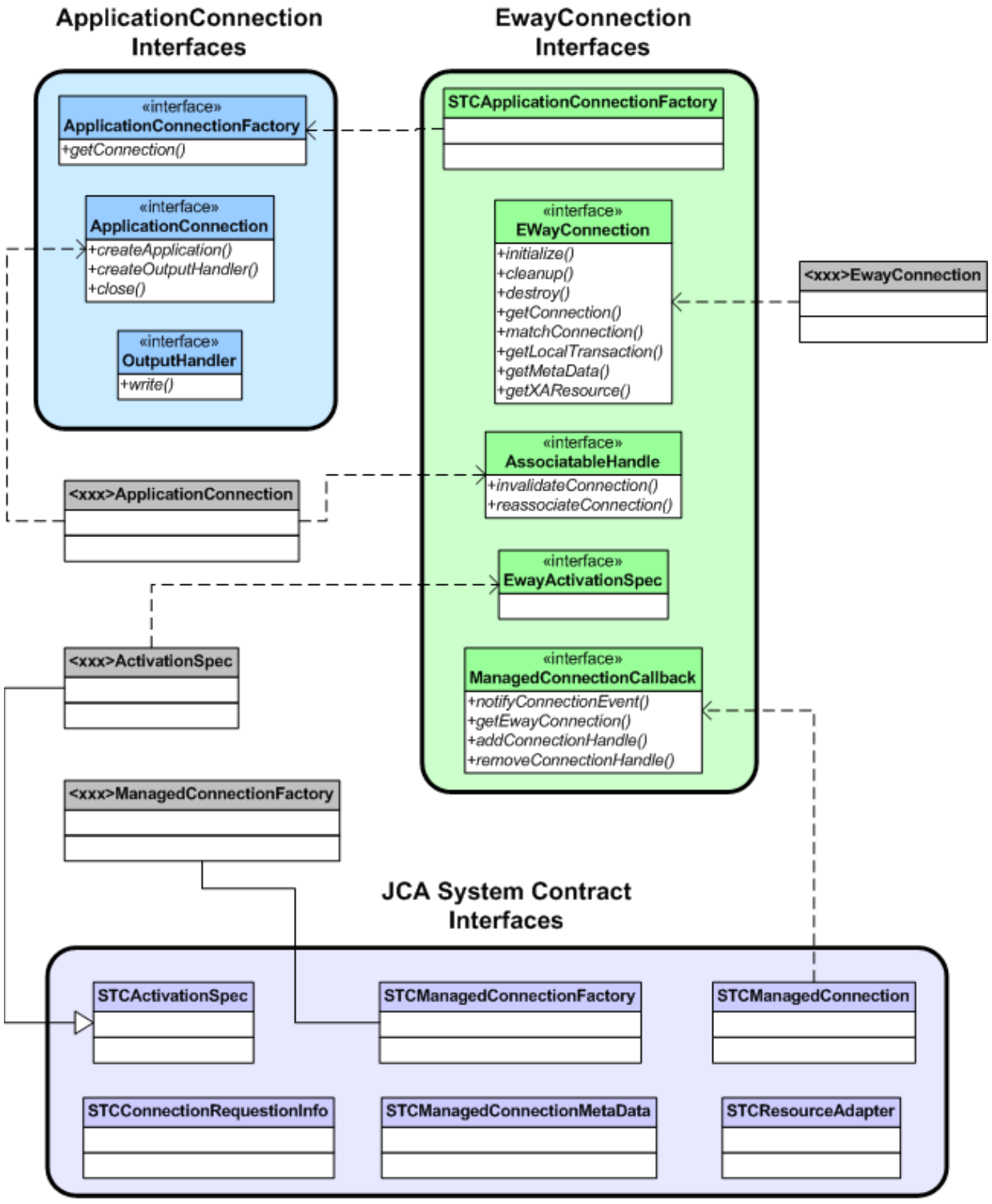
The following interfaces represent the core eDK tool.

- <eWay Name>ApplicationConnection Interfaces
- <eWay Name>EwayConnection Interfaces
- <eWay Name>JCA System Contract Interfaces

The following class files contain methods to be implemented in the eWay.

- <eWay Name>ApplicationConnection
- <eWay Name>ApplicationSpec
- <eWay Name>EwayConnection
- <eWay Name>ManagedConnectionFactory
- <eWay Name>Application

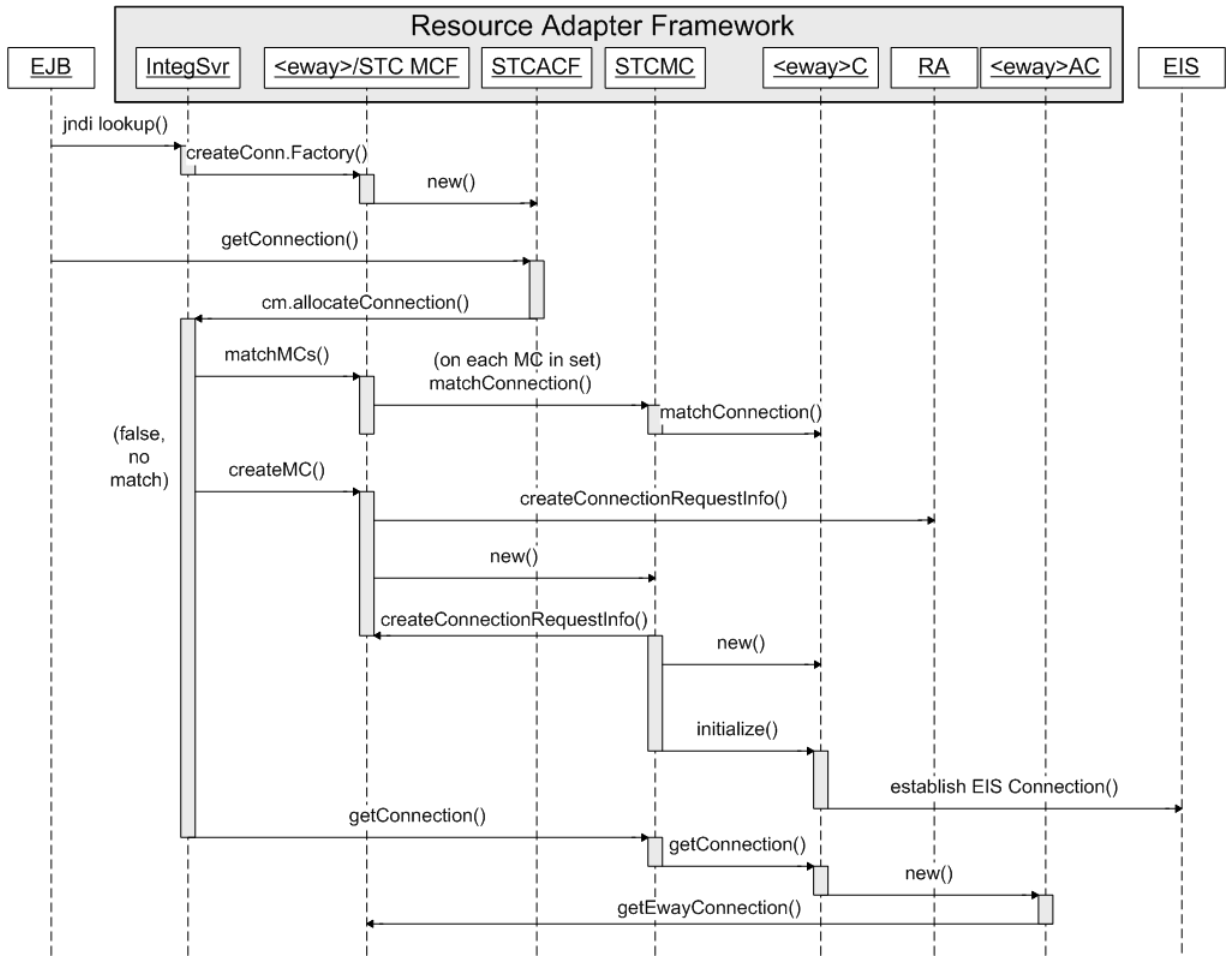
Figure 25 RA Framework Class Diagram



A.4 RA Framework Sequence Diagram

The following diagram describes the client interaction with a resource adapter using the RA framework.

Figure 26 RA Framework Sequence



The RA framework sequence occurs as follows:

1. The EJB client performs a JNDI lookup to obtain a client connection factory (**ApplicationConnectionFactory**).
2. The Application Server (Integration server) uses the RA framework managed connection factory to create an **ApplicationConnectionFactory** which is passed back to the EJB client's JNDI lookup.
3. The EJB client obtains an ApplicationConnection by calling the **getConnection()** method on the **ApplicationConnectionFactory**.
4. The Integration server and RA framework classes interact by first trying to obtain a matching connection from the connection pool. If a matching connection is not found, a new connection is requested by the Integration server from the resource

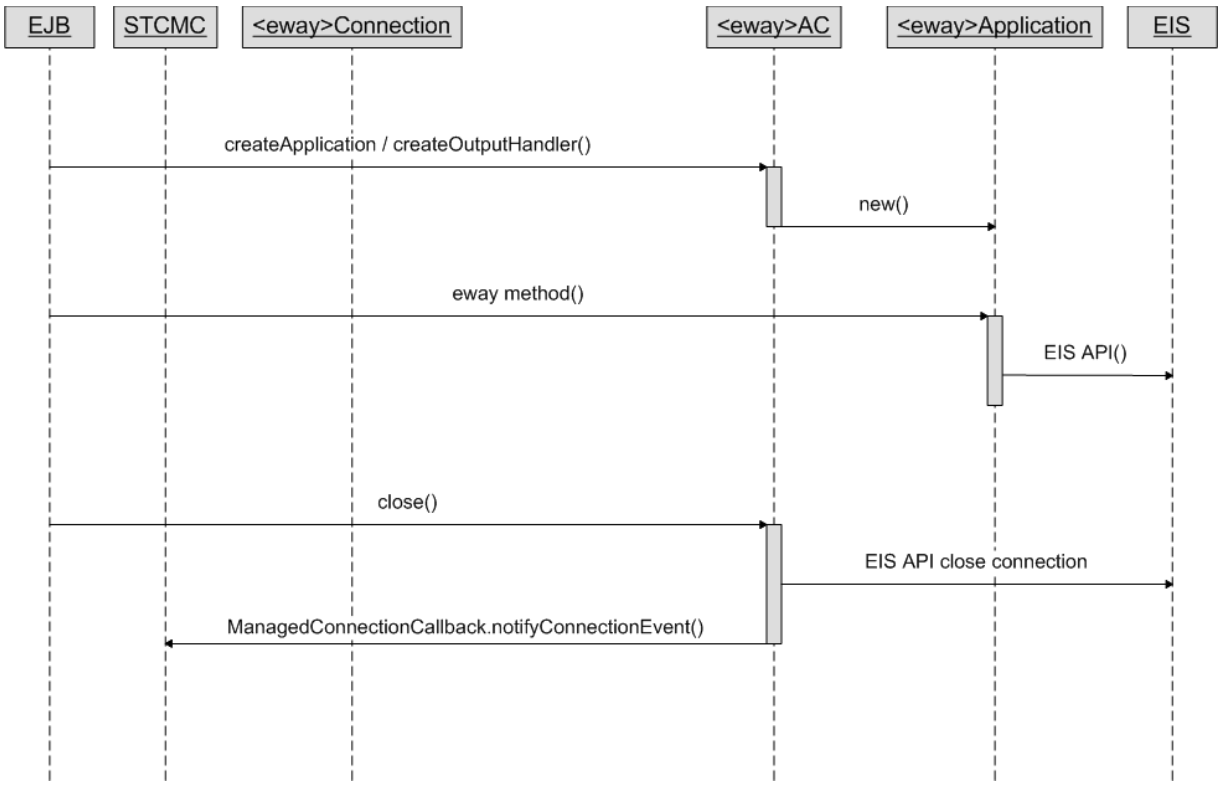
adapter. Note that a **ConnectionRequestInfo** object containing the connection configuration properties is used when checking for a match.

5 The **EwayConnection** interface implementation is used when establishing a physical connection to the EIS. The initialize method is normally where this is performed. It is also used in connection matching via the **matchConnection()** method. The **getConnection()** method is used to return the established connection.

A.5 Client Application Sequence Diagram

The following diagram shows the interaction after the EJB client has obtained a resource adapter connection.

Figure 27 JCA Framework Sequence



The JCA sequence occurs as follows:

1 The **createApplication()** method is called—once an ApplicationConnection is obtained—to obtain the eWay-specific Application object which exposes the methods interacting with the EIS.

2 The EJB client calls the various methods in the eWay's Application object

3 The EJB's resource adapter connection is closed via the close method in the ApplicationConnection after completing execution of the method calls.

4 The Connector architecture provides a notification mechanism between the Application server and the resource adapter for connection events. These are hidden in the RA framework.

Appendix B

Command Line Code Generation

This Appendix covers topics such as generating code by Command Line, and the eDK Definition File.

What's in this Chapter

B.1 Generating eDK Code by Command Line

In addition to using the eWay Development Kit Build Tool to generate code, you can also create eWays via the command line.

To Generate Code by Command Line:

1. Run the **env.bat** file located at the root level of the extracted eDK folder to set up the Implementation Environment.
2. Create the eDK definition **<adapter_name>.xdef** file using the eDK Build Tool. See **Steps Required to Build an eWay** on page 18. Alternately, you can also create a definition file manually. See **eDK Definition File** on page 76.

   An eDK definition file contains all required eWay metadata, including eWay name, external application name, third-party **.jar** files, operations to be exposed in Java and eInsight Business Processes.
3. Create a configuration template **<adapter_name_template.xml** file using the eDK Build Tool.
4. Run the following command from the **<%STC_ROOT%>\eways\edk\devtools** folder.

   ```
   ant run -Dxmlfilename=<xdef_file_with_full_path>
   ```

   This generates the "connectors" and "eways" folder under the output directory (as specified in the definition file).

Choosing Your Working Directory

As an optional step, you can choose to either work from the generated output folder, or choose a new working directory.

For more information, see **Choosing a Working Directory** on page 42

B.2 eDK Definition File

The eDK definition file is an **.xdef** file that stores metadata information about an eWay. You can choose to manually create an eDK definition file, or use the eDK Build Tool.

All information about the eDK definition itself, such as eDK user, version number, revision number, date created, date modified, are stored as attributes in the MetaData element. Note that the eWay user does not need to populate the date created/modified fields; this information is automatically generated in the proper date format by the eDK Build Tool.

The Properties element contains the following basic information needed to build an eWay:

- **External_application_name (required):** The name of the external application.
- **Eway_adapter_name:** The name of the eWay. This is used for the directory name under connectors and eways folder.
- **Output_directory (required):** The output folder for the generated eWay shell code.
- **Log_File_Name (optional):** When specified, all eWay generation loggings will be saved to this file
- **Resource_locations (required):** This element allows you to specify resource file locations, namely configuration template location, and optional icon file locations.

Resource file locations include:

- **Config_template_location:** The file location of the configuration template.
- **Icon_file_location:** This element allows you to specify icon files to be used for the eWay. Other eWay icons not specified in this element, but created with the eWay Development Kit Build Tool include:
  - **External_app_icon:** The file location of the External System icon to be displayed on eDesigner Environment and Project Explorer.
  - **Connectivity_map_icon:** The file location of the External System icon to be displayed on eDesigner Connectivity Map.
  - **BPEL_invoke_icon:** The file location of the Business Activity icon to be displayed on eInsight Business Process canvas. This icon is used for all outbound business activities.
  - **BPEL_receive_icon:** The file location of the Business Invoke Activity icon to be displayed on eInsight Business Process canvas. This icon will be used for all inbound business activities.

The **Java_client_interface** describes all inbound and/or outbound operations to be exposed in Java Collaboration Editor. The **Java_client_interface** allows three elements to be defined.

The **Operation** element defines the operation to be exposed in Java Collaboration Editor. For each operation, it is required to define the operation name and mode. The value of the mode attribute can be either "inbound" or "outbound". You can also define

the description of the operation using the "description" attribute. Operation elements allow you to specify input, output, and exception, with following sub-elements:

- **Input element:** has the required name and type attributes. In case of Java Collaboration operations, any number of inputs can be added to the operation.
- **Output element:** has the required type attribute.
- **Exception element:** has the required type attribute.

The **Attributes** element allows you to define all attributes for a Java Collaboration OTD. This element can hold as many attributes as required to define a JCD OTD. Each Attribute element requires a name and type (**.xml**) attribute.

The **Types** element contains all User Defined data types and/or User Defined class definitions. This element can hold any number of User Defined types.

- **User_defined_data_container elements:** allows users to specify only data elements. It has the required "Element" (**.xml**) sub-element, which in turn has the required "name" and "type" attributes. This definition allows the eWay Development Kit Build Tool to generate java Bean classes for each "Element" defined. The "type" attribute can also refer to the name of another "User_defined_data_container" element.
- **User_defined_class elements:** allows users to specify a Java class definition. It allows the "Element" sub-element, "Attribute" sub-element and "Method" subelement. "Method" element behaves very much like the "Operation" element. The reason for User_defined_class elements is to allow you to specify the "inbound OTD" type for JCD inbound operations. It is *only* intended to be used for an "input" type definition of a JCD inbound operation.

The **BPEL_client_interface** describes all inbound and/or outbound operations to be exposed in Business Process Editor. It contains very similar elements as in **JCD_client_interface**. A notable difference is that only one "input" element is allowed for any BPEL operation. Also, no "User_defined_class" element is allowed in the "Types" element.

Appendix C

JNI Sample eWay

This Appendix discusses the steps required to create the JNI Sample eWay packaged with the eDK.

What's in this Chapter

C.1 Overview

The **JNISample** sample outbound eWay provides a simple outbound scenario that shows how the eDK makes use of JNI. The sample eWay provides a method in JCD as well as an Operation in the eInsight Business Process that copies a file from one directory to another.

The following steps outline the procedures required to create the **JNISample** eWay using the eDK.

Steps required to build an the using the eWay Development Kit include:

1. Start the eWay Development Kit Build Tool.
2. Create and specify details of the new eWay – such as the eWay Name, Description, Version, and so forth.
3. Define the eWay Interfaces to be used in Java Collaborations or the eInsight Business Processes.

4 Define the eWay configuration template.

5 Run the Code Generator to create the eWay shell code (using either the eWay Development Kit Build Tool or by command line).

6 Implement the generated shell code and run **Apache Ant scripts** to build the JCA components.

7 Run the **Apache Ant** build scripts to build the eWay **.sar** file.

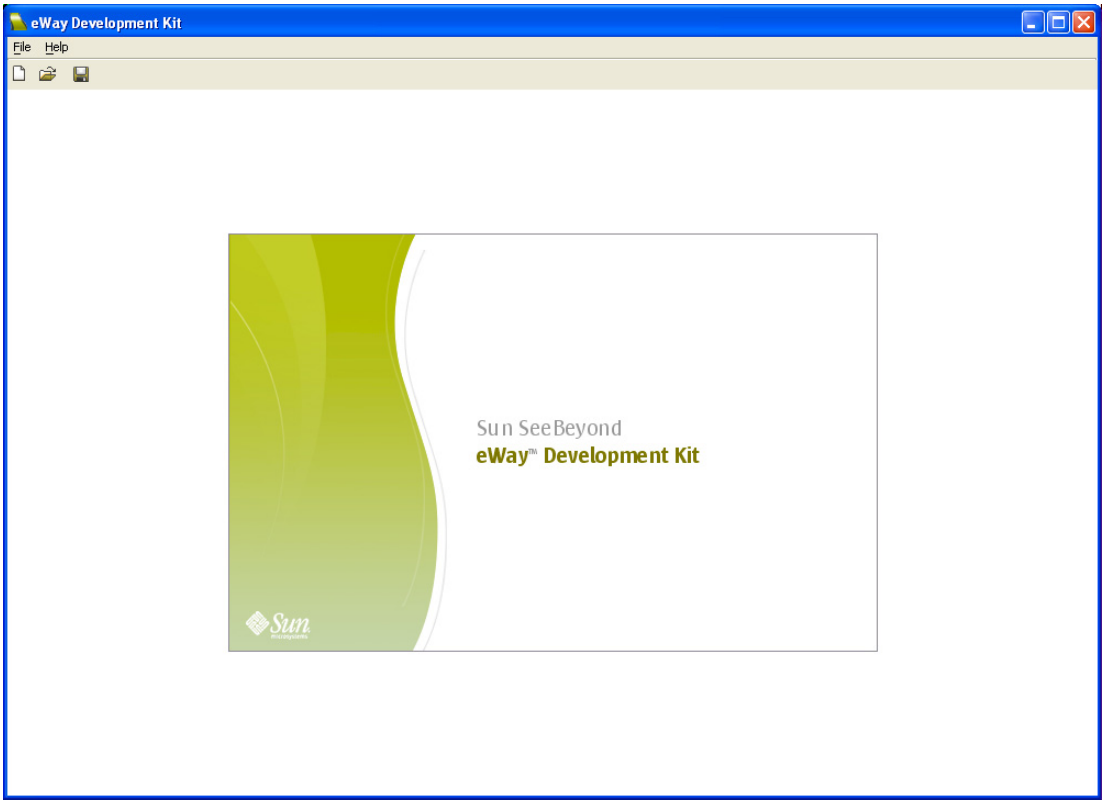
C.2 Start the eWay Development Kit Build Tool

To start the eWay Development Kit build tool:

1 Browse to the folder that contains the extracted eDK files.

2 Run the **runedk.bat** file to launch the eDK Build Tool. The eWay Development Kit splash screen appears.

Figure 28 eWay Development Kit Build Tool



C.3 Create and Specify the New eWay

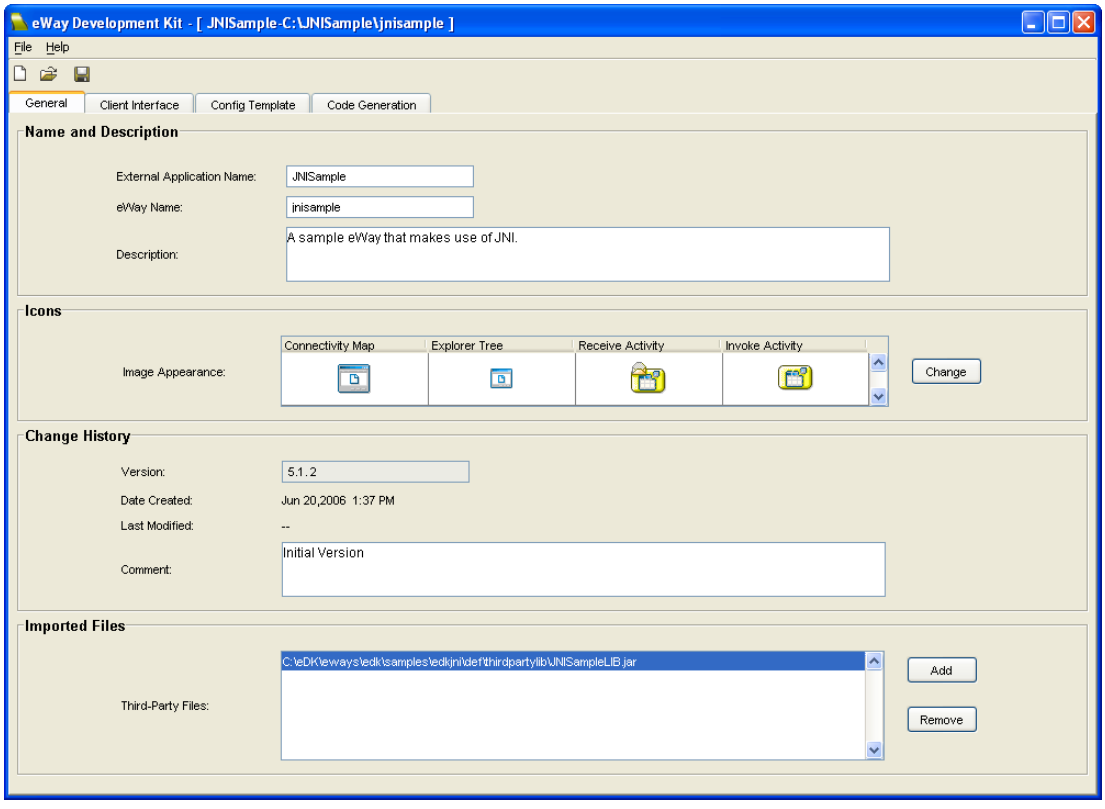
Perform the following steps to create a new eWay:

1. From the menu bar, select **File** then select **New eWay**, or click the **New eWay** icon. An empty eWay template appears in the build tool window.
2. Enter the following information into the General tab of the eDK Build Tool:
   - **External Application Name:** JNISample
   - **eWay Name:** jnisample
   - **Description:** A sample eWay that makes use of JNI.
   - **Comment:** Initial Version
   - **Third-Party Files:** JNISampleLIB.jar

You can find the **JNISampleLIB.jar** file, and its corresponding shared library **libtfu.dll**, in the following location:

```
<%STC_ROOT%>\eways\edk\samples\edkjni\def\thirdpartylib
```

Figure 29 JNISample General Tab



C.4 Enter the Sample JNI eWay Client Interfaces

In the eWay Development Kit, the Client Interfaces represent the methods, user defined operations, and attributes exposed to the eWay user. The Client Interface also represents the eWay's OTD.

C.4.1 Java Interfaces

For the sample JNI eWay, there are two methods in the JCE outbound interface:

- copy
- lastErr

There is no inbound interface for this sample.

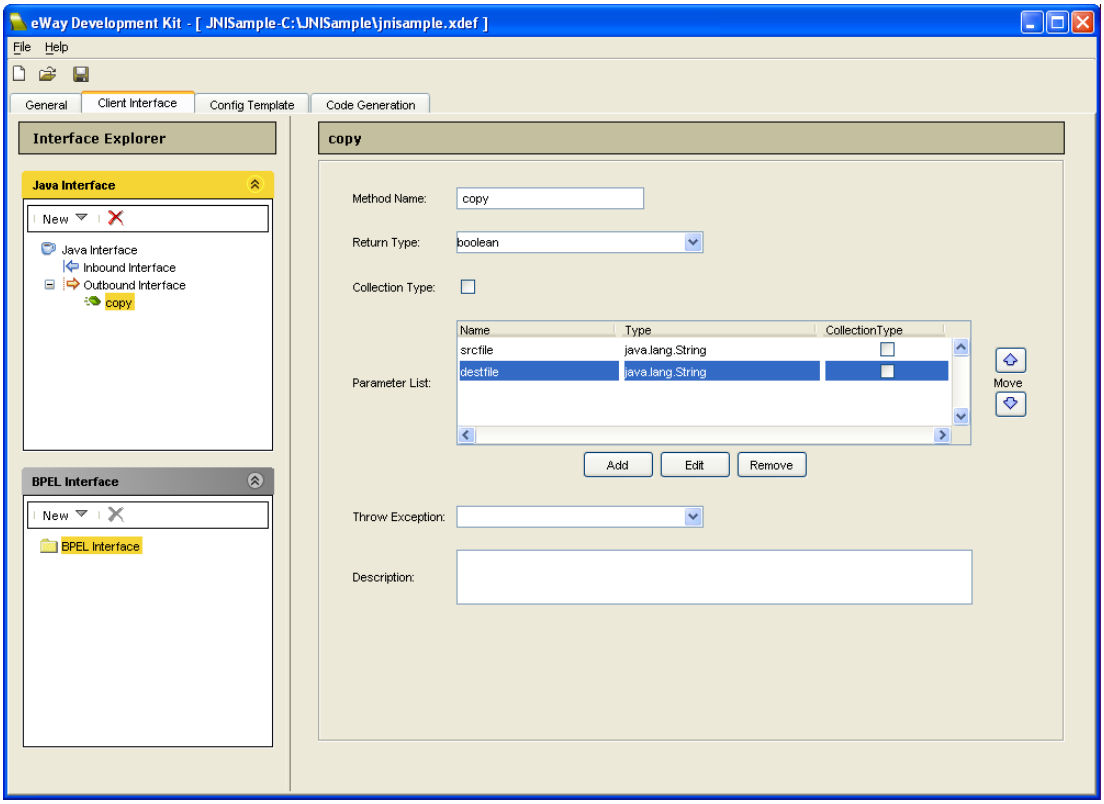
About the copy Method

The **copy()** method has a return type (of boolean) and two input parameters (both with type java.lang.String). The purpose of this method is to copy a text file from the source directly to the destination directory.

To create the copy method:

1. Select the Client Interface tab on the eWay Development Kit Build Tool. The Client Interface window appears.
2. From the Java Interface box, click **New** > **Java OTD Methods** > **Outbound**.
3. Enter the following:
   - **Method Name:** copy
   - **Return Type:** boolean
   - **Parameter List:** The copy method has two parameters.
     - **Name**: srcfile **Type:** java.lang.String
     - **Name**: destfile **Type:** java.lang.String

Figure 30 Java Interface copy Method



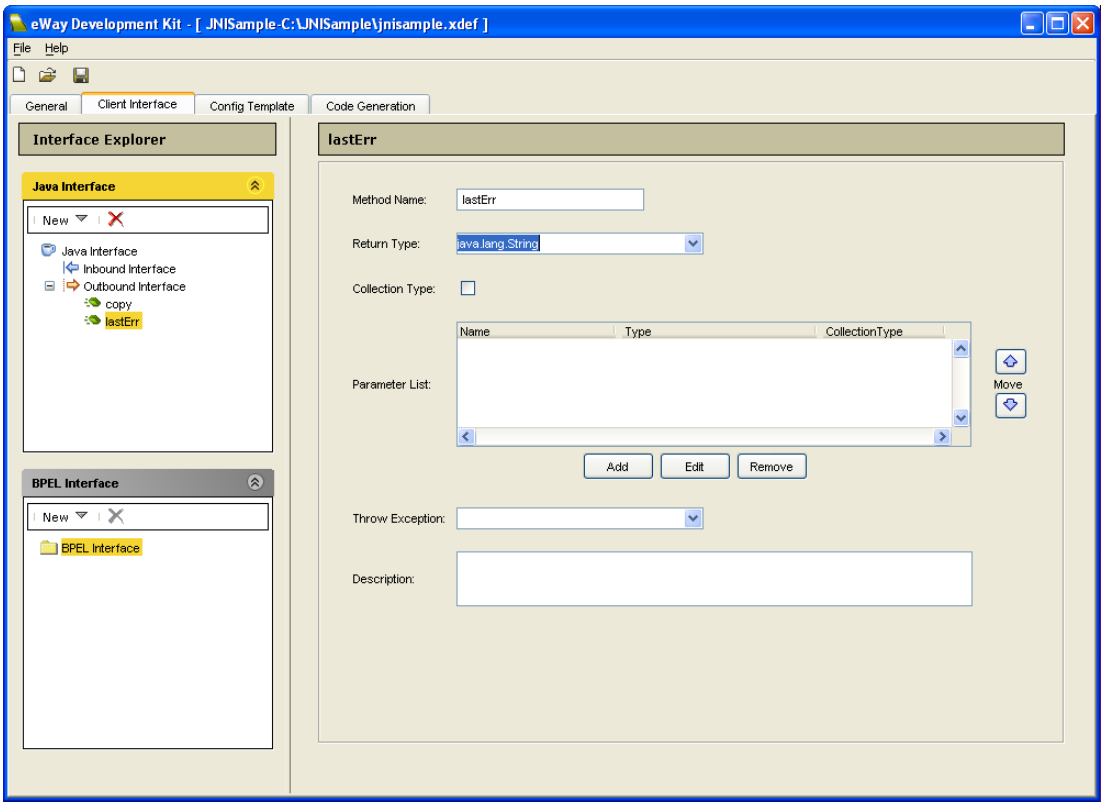
About the lastErr Method

The **lastErr()** method has a return type of **java.lang.String** and does not take any input parameters. The purpose of this method is to return the last error message, if any.

To create the lastErr method:

1. From the Java Interface box, click **New** > **Java OTD Methods** > **Outbound**.
2. Enter the following:
   - **Method Name:** lastErr
   - **Return Type:** java.lang.String

Figure 31 Java Interface lastErr Method



C.4.2 BPEL Interfaces

The outbound BPEL interface only has the **copy()** method. The **lastErr()** method added in the Java Interface box does not make sense in this case since eInsight Business Processes are stateless.

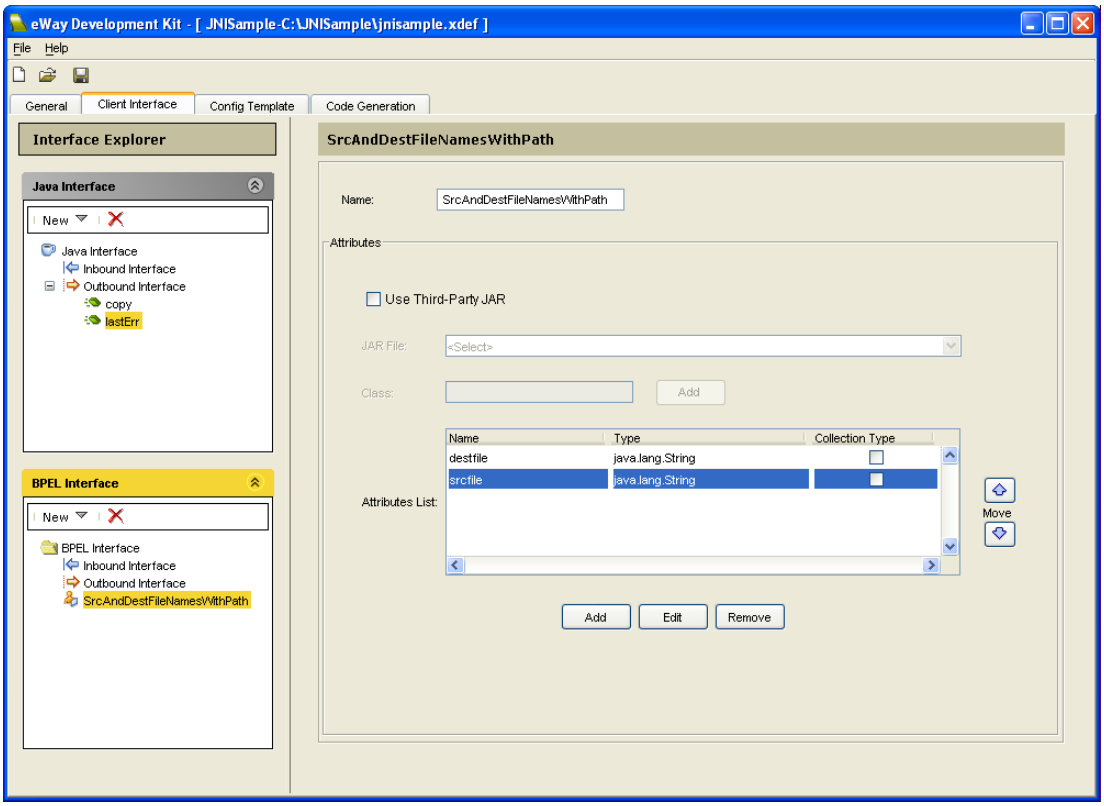
The **copy()** method has an output message of type **boolean** and an input message of User Defined Data Type **SrcAndDestFileNamesWithPath**. Therefore, we need to define the **SrcAndDestFileNamesWithPath** type first before specifying the input parameters.

To create the User Defined Data Type SrcAndDestFileNamesWithPath:

1. From the BPEL Interface box, click **New** > **User Defined Data Type**.
2. Enter the following:
   - **Name:** SrcAndDestFileNamesWithPath
   - **Attribute List:** SrcAndDestFileNamesWithPath has two attributes.
     - **Name:** destfile **Type:** java.lang.String
     - **Name:** srcfile **Type:** java.lang.String

Once you create the User Defined Data Type, go back and add the input parameters for the **copy()** method.

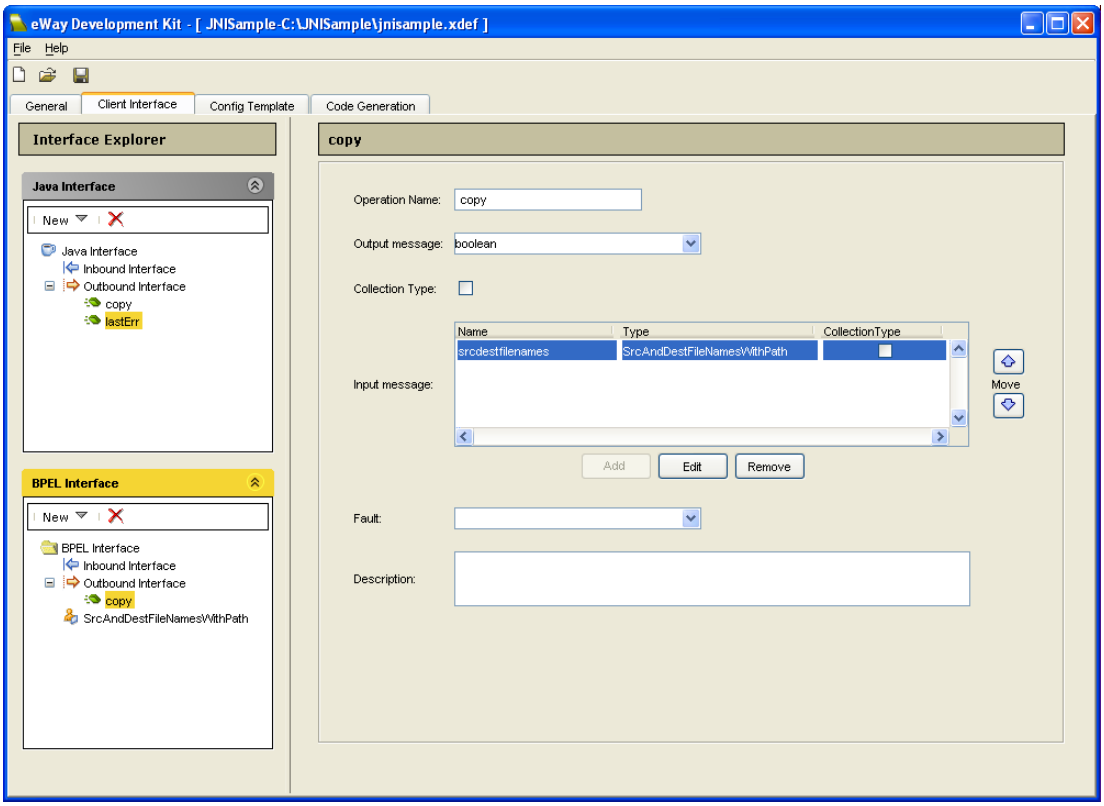
Figure 32 BPEL Interface SrcAndDestFileNamesWithPath



To create the copy() method:

1. From the BPEL Interface box, click **New** > **Operation** > **Outbound**.
2. Enter the following:
   - **Method Name:** copy
   - **Return Type:** boolean
   - **Parameter List:** The **copy()** method has one attribute.
     - **Name:** srcdestfilenames **Type:** SrcAndDestFileNamesWithPath

Figure 33 BPEL Interface copy Method



C.5 Define the eWay Configuration Template

Since the third-party JNI library is basically a collection of class methods, there is no need to specify any specific configuration parameters.

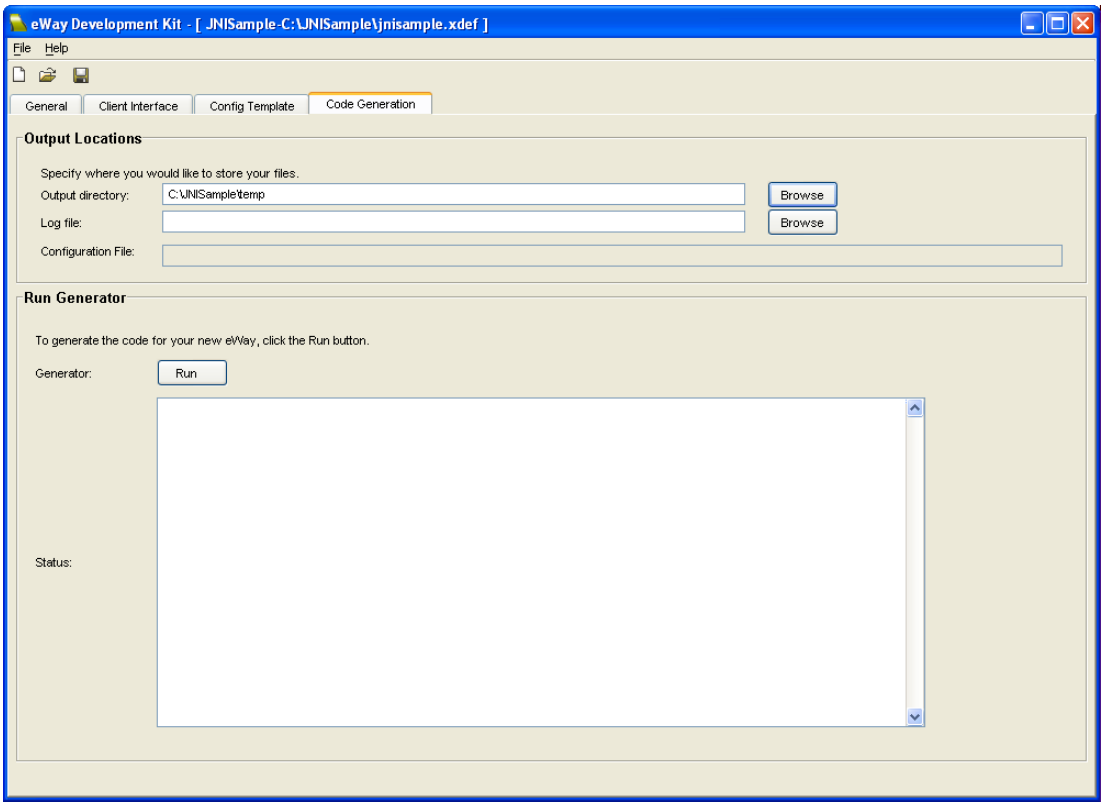
C.6 Run the Code Generator

The Code Generation tab on the eWay Development Kit is used to generate the "connectors" and "eways" folders at a specified location.

To generate code:

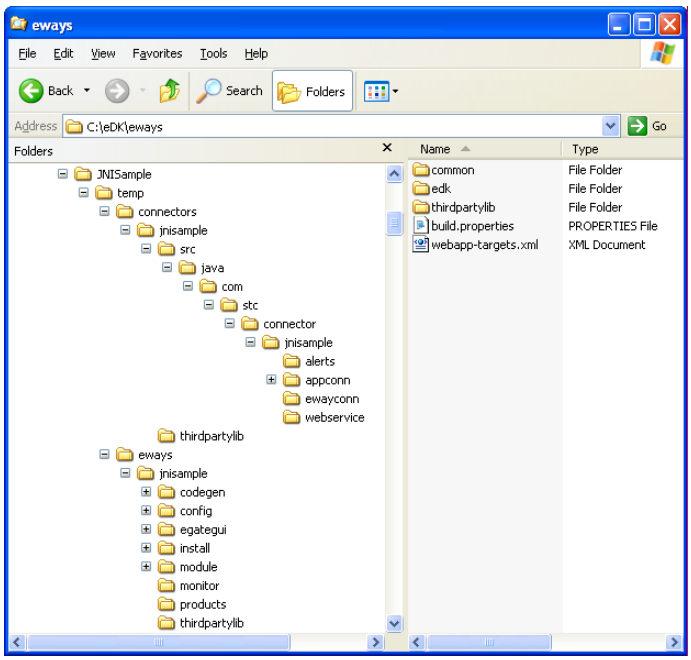
1. Select the Code Generation tab on the eWay Development Kit.
2. Browse to an output directory. This is the location where the "connectors" and "eways" folders are created. For example, c:\JNISample\temp.
3. Click **Run** from the Run Generator frame.

Figure 34 JNISample Code Generation Tab



The following file structure appears after code generation.

Figure 35 JNISample Directory Structure



C.7 Implement the Sample JNI eWay Functionality

You must modify the following Java files in the generated code to implement the required functionality.

- JNISampleClientApplciationImpl.java
- JNISampleEwayConnection.java
- JNISampleWebClientApplication.java

The following code snippets describe what implementation is required in each of these class files.

C.7.1 JNISampleClientApplciationImpl

Change the return code from "false" to "true":

```
/** No implementation is necessary in the case when interaction with
the EIS
*  is connection-less, e.g. Email, HTTP etc.
*  Un-comment the following return clause to always
*  return true when connection validation is not applicable.
*/
    return true;
```

Enter the following code:

```
public java.lang.String lastErr() {
        // <Start_User_Code>
        return TextFileUtils.lastErr();
        // <End_User_Code>
    }
```

Enter the following code:

```
public boolean copy(java.lang.String srcfile, java.lang.String
destfile) {
        // <Start_User_Code>
        return TextFileUtils.copy(srcfile, destfile);
        // <End_User_Code>
    }
```

C.7.2 JNISampleEwayConnection

Enter the following code:

```
public ManagedConnectionMetaData getMetaData() throws
ResourceException {
/** Currently, the integration server does NOT use this metadata.
* however, this could be used by other application servers
* to gather information about the underlying EIS instance.
          * Please modify the following clause if necessary
          */
        // <Start_User_Code>
        return new STCManagedConnectionMetaData(
            "JNISample", "UNKNOWN", 1, "UNKNOWN");
        // <End_User_Code>
    }
```

C.7.3 JNISampleWebClientApplication

Enter the following code:

```
public CopyOutput copy(SrcAndDestFileNamesWithPath srcdestfilenames)
{
        // <Start_User_Code>
        CopyOutput co = new CopyOutput();
co.setCopyOutput(TextFileUtils.copy(srcdestfilenames.getSrcfile(),
srcdestfilenames.getDestfile()));
        return co;
        // <End_User_Code>
    }
```

C.8 Build the .sar File

Two steps are required to build the **.sar** file for the implemented eWay. The following steps describe how to build a **.sar** file for the **JNISample** eWay.

Note: *Remember to set your environment, by running the env.bat file, before building your .sar file.*

1. Browse to the **<Working_Directory>\connectors\jnisample** folder, and run the following:

   ```
   ant clean install -f connector-build.xml
   ```

   This should build the **jnisample.rar** file and all the other required **.jar** files at the following location:

   ```
   <%STC_ROOT%>\BUILD\Modules\connectors\lib
   ```

2. Browse to the **<Working_Directory>\eways\jnisample** folder and run the following:

   ```
   ant clean install -f eway-build.xml
   ```

   This creates the new **eDK_JNISampleeWay.sar** file and **Product_List.sar** file in the following location:

   ```
   <%STC_ROOT%>\BUILD\Images\Products
   ```

C.9 Upload the New eWay to the Repository

The following steps describe how to upload the new **eDK_EDKFILEeWay.sar** file that we created in the previous section, to the Java CAPS Repository.

1. Click the **Administration** tab in the Java Composite Application Platform Suite Installer.
2. Click the **Click to install additional products link**, and then click **Browse** to update the Product List.

3 Locate the **Product_List.sar** file that you created for the **eDK_JNISampleeWay**, and then click **Submit**.

4 Expand the list of available eWays from the **eWays and Addons** folder, select the **eDK_JNISampleeWay**, and then click **Next**.

5 Browse to and select the **eDK_JNISampleeWay.sar** file, then click **Next** to install the new eWay.

C.10 Run the Enterprise Designer Update Center

For detailed information on running the Enterprise Designer Update Center, see the *Composite Application Platform Suite Installation Guide*.

C.11 Create, Build, and Deploy the Sample Projects

The creation and deployment of new eWay Projects created using an eDK based eWay is beyond the scope of this user's guide. Detailed information, including examples of how to create and deploy sample Projects are found in the *Sun SeeBeyond eGate™ Integrator Tutorial*.

Index

Symbols

A

B

C

D

E

S

T

U

V

W