SUN SEEBEYOND

# eWAY™ SNA ADAPTER USER'S GUIDE

**Release 5.1.1**

Sun microsystems

Version 20060614130929

# Contents

Chapter 3



Chapter 4

Chapter 1

# Introducing the SNA eWay

Welcome to the *Sun SeeBeyond eWay™ SNA Adapter User's Guide*. This document includes information about installing, configuring, and using the Sun Java Composite Application Platform Suite SNA eWay™ Adapter, referred to as the SNA eway throughout this guide.

This chapter provides an overview of System Network Architecture (SNA) data communications and supported logical unit types.

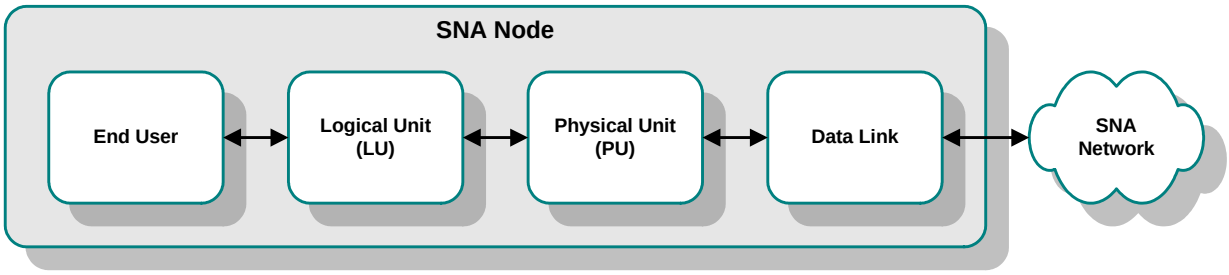


## 1.1 About SNA

SNA is a data communications architecture developed by IBM to specify common conventions for communication between various IBM hardware and software products. It is specifically designed to address issues of reliability and flexibility of sharing data between components and their peripherals. Many vendors other than IBM also support SNA, allowing their products to interact with SNA networks.

An addressable unit on an SNA network is called a node, and is made up of four functional components forming a hierarchy as shown in Figure 1.

**Figure 1** SNA Node Architecture

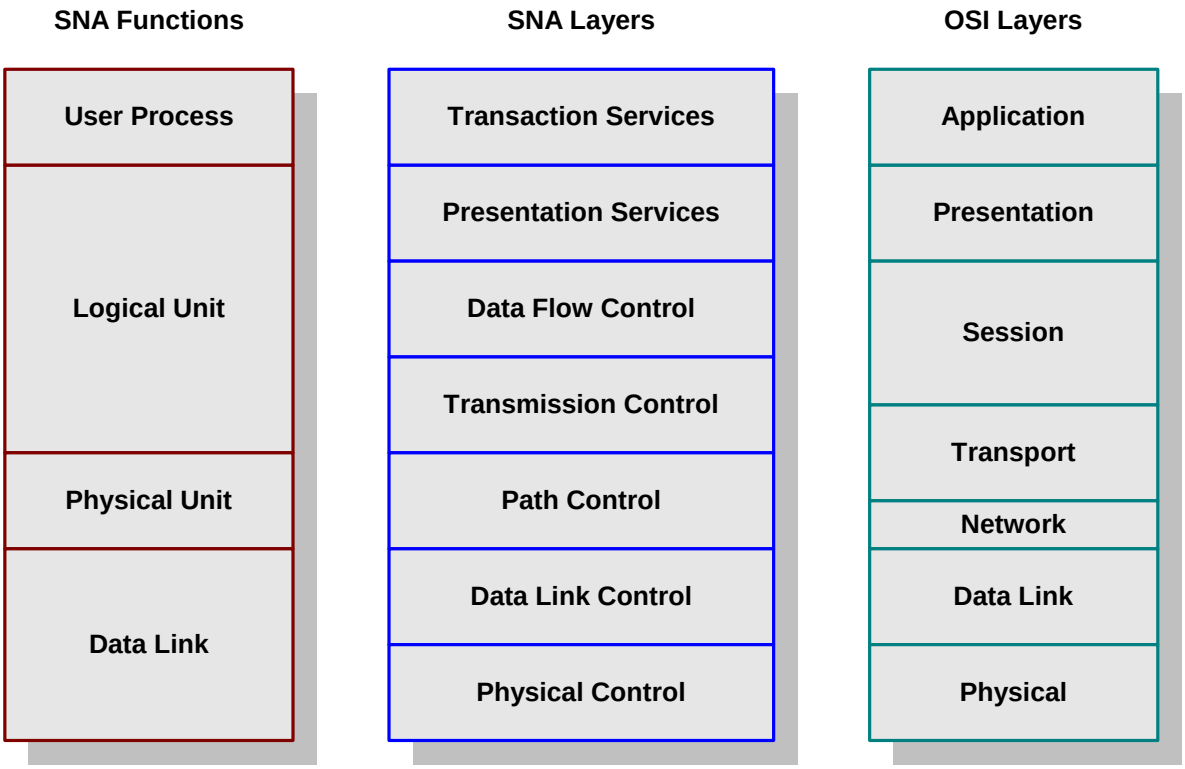



To establish a communications session, SNA uses Logical Units (LUs) as entry points into the network. There are several types of LUs, currently type 0 through type 6.2. Most of the LU types are specific to IBM operating environments, but type 6 is intended for use in a distributed data processing environment.

Generally, an LU can communicate only with another LU of the same type, but specific exceptions to this rule exist with type 6.2. LU6.2 is the least-restrictive of the various LU types, and also supports multiple concurrent sessions. As a result, it is the LU most widely supported by other system vendors.

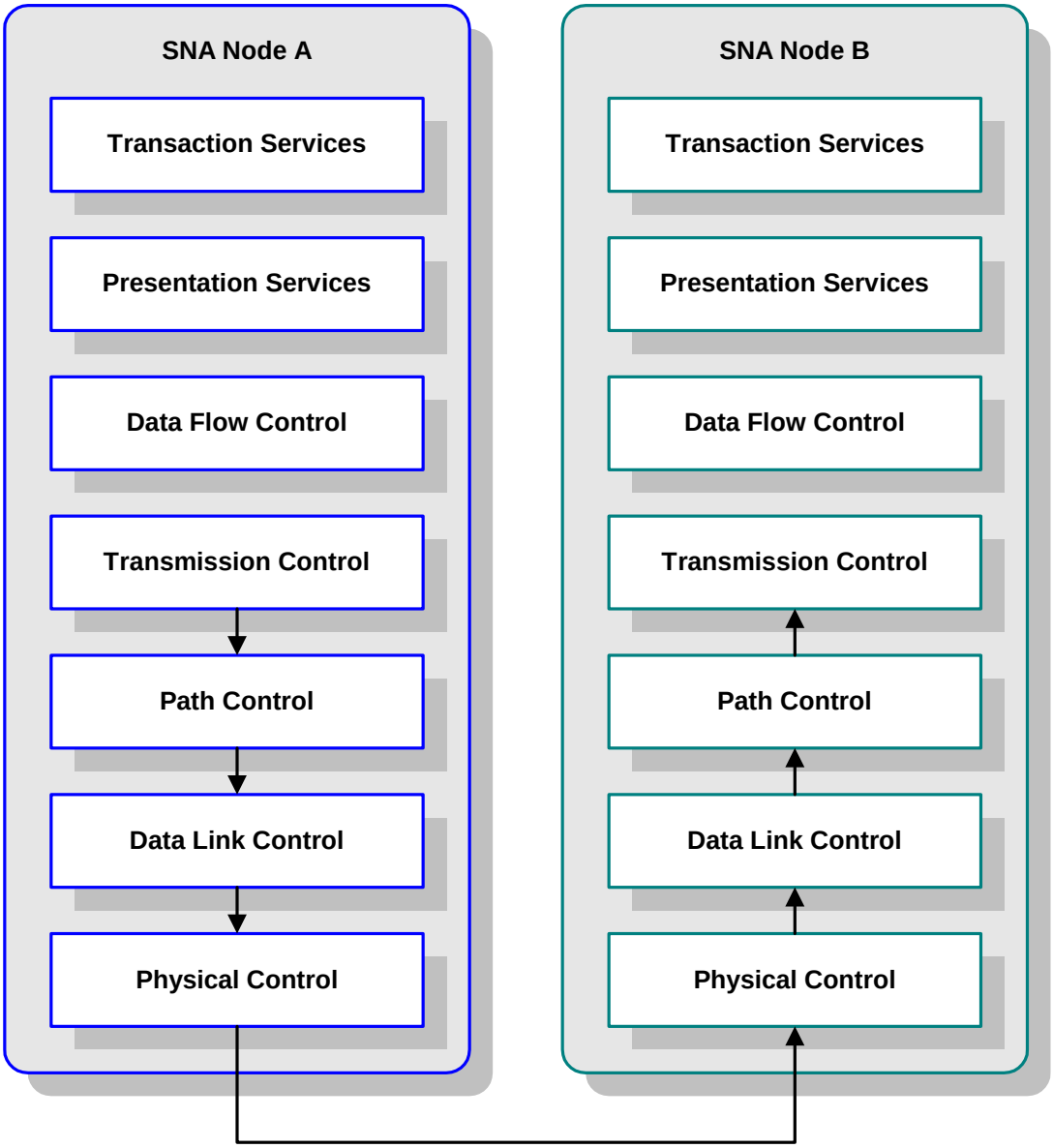
Like the OSI model, SNA functions are divided into seven hierarchical layers, but the layers are not identical. Their relationships to each other, and to the SNA node functionality, are shown in Figure 2. The Transport Network handles the lower three layers, while the Network Accessible Units (NAU) implement the upper four layers by using the services of the Transport Network to establish communication between nodes.

**Figure 2** SNA Functional Layers




SNA defines formats and protocols between these layers that allow equivalent layers in different nodes to communicate with each other. Also, each layer provides services to the layer above, and requests services from the layer below. As an example, the communication path between two Transmission Control layers would appear as shown in Figure 3.

**Figure 3** Equivalent-Layer Communications Path



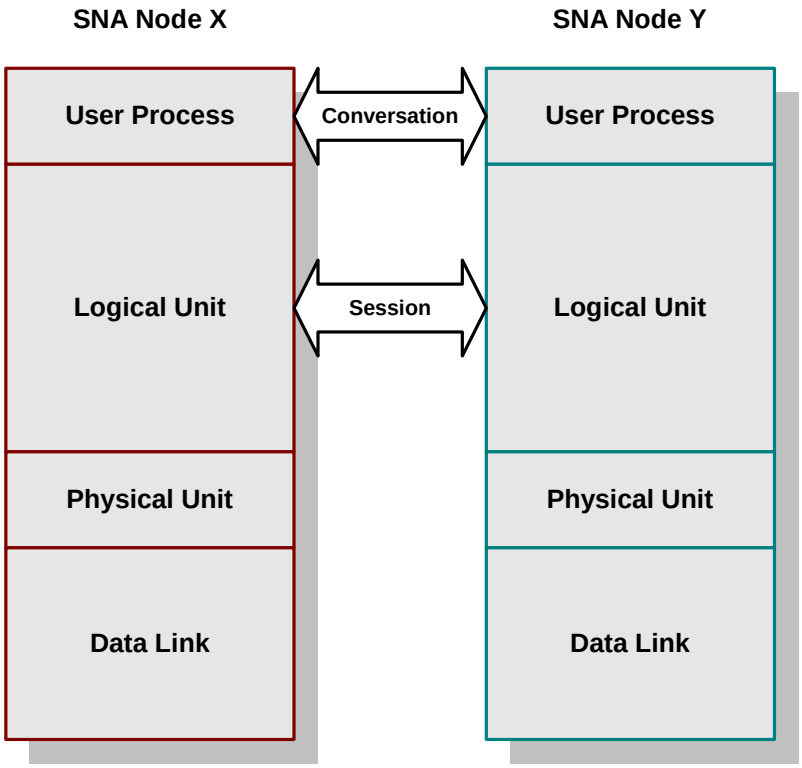

SNA uses a standard method for the exchange of data within a network. This standard method defines how to establish a route between components, how to send and receive data reliably, how to recover from errors, and how to prevent flow problems.

Originally designed for networks in which a mainframe computer controls the communications relationships, SNA has since evolved to incorporate protocols and implementations to allow two user processes to communicate with each other directly. These two different networking models, or roles, are referred to as hierarchical and peer-oriented, respectively. The peer-oriented model is designed to allow distributed control of the communications process independent of the mainframe.

The peer-to-peer connection between two user processes is known as a *conversation*, while the peer-to-peer connection between two LUs is known as a *session*. A session is

generally a long-term connection between two LUs, while a conversation is generally of shorter duration.

**Figure 4** Sessions and Conversations




What is shown in Figure 2 and Figure 4 as a *User Process* is also known as a *Transaction Program* (TP). Also, the interface between a User Process and an LU is known as *Presentation Services*.

## 1.1.1 Supported Logical Unit Types

### SNA LU6.2

LU 6.2, also known as APPC (Advanced Program-to-Program Communication), is used for Transaction Programs communicating with each other in a distributed data processing environment. In a CPIC (Common Programming Interface for Communications) implementation, CPIC provides the API that contains the commands, known as verbs, that are used by LU 6.2 to establish communication sessions.

Two types of Presentation Service interfaces are possible with LU6.2: mapped conversations and unmapped, or basic, conversations. Table 1 summarizes the set of LU6.2 commands for basic conversations. Equivalent commands for mapped conversations have the prefix <MC_> added to the command name. Note that “control operator verbs” are not listed.

**Table 1** LU6.2 Commands

| Name | Description |
|---|---|
| ALLOCATE | Allocates a conversation with another program. |
| CONFIRM | Sends a confirmation request to the remote process and waits for a reply. |
| CONFIRMED | Sends a confirmation reply to the remote process. |
| DEALLOCATE | De-allocates a conversation. |
| FLUSH | Forces the transmission of the local SEND buffer to the other LU. |
| GET_ATTRIBUTES | Obtains information about a conversation. |
| PREPARE_TO_RECEIVE | Changes the conversation state from SEND to RECEIVE. |
| RECEIVE_AND_WAIT | Waits for information (either data or confirmation request) to be received from the partner process. |
| RECEIVE_IMMEDIATE | Receives any information that is available in the local LU's buffer, but does not wait for information to arrive. |
| REQUEST_TO_SEND | Notifies the partner process that the local process wants to send data. When a "send" indication is received from the partner process, the conversation state changes. |
| SEND_DATA | Sends one data record to the partner process. |
| SEND_ERROR | Informs the partner process that the local process has detected an application error. |

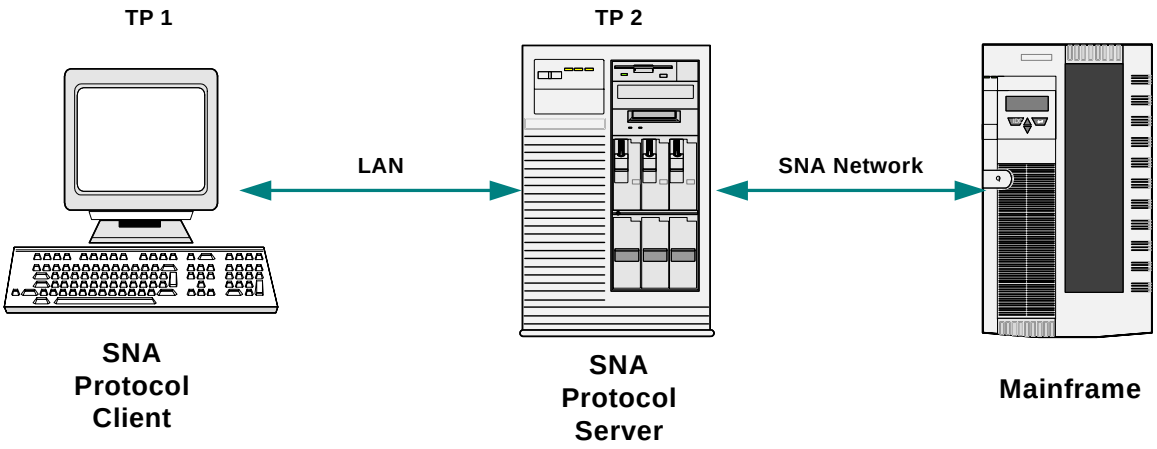
## 1.2 About the SNA eWay

The SNA eWay enables the eGate Integrator system to access an SNA network environment to drive entire transactions, including conversational transactions.

The SNA eWay is an interface that makes calls to an SNA Server. The SNA Server acts as a high-speed gateway between distributed SNA Clients and the SNA network having a mainframe host system (see Figure 5).

In a typical data exchange using the SNA eWay, the eWay invokes the LU6.2 protocol--through the invocation of CPI-C calls--to enable the SNA client to send requests to the SNA server. For outbound eWays, the eWay can be triggered by any incoming message. For inbound eWays, the eWay is triggered by established conversation activity.

**Figure 5** SNA Data Exchange




## 1.3 What's New in This Release

The SNA eWay Adapter version 5.1.1 includes the following changes and new features:

- Version Control: An enhanced version control system allows you to effectively manage changes to the eWay components.
- Manual Connection Management: Establishing a connection can now be performed automatically (configured as a property) or manually (using OTD methods from the Java Collaboration).
- Multiple Drag-and-Drop Component Mapping from the Deployment Editor: The Deployment Editor now allows you to select multiple components from the Editor's component pane, and drop them into your Environment component.
- Support for Runtime LDAP Configuration: eWay configuration properties now support LDAP key values.
- MDB Pool Size Support: Provides greater flow control (throttling) by specifying the maximum and minimum MDB pool size.
- Connectivity Map Generator: Generates and links your Project's Connectivity Map components using a Collaboration or Business Process.

Many of these features are documented further in the *Sun SeeBeyond eGate™ Integrator User's Guide* or the *Sun SeeBeyond eGate™ Integrator System Administration Guide*.

## 1.4 About This Document

This document includes the following chapters:

- **Chapter 1 "Introducing the SNA eWay"**: Provides an overview description of the product as well as high-level information about this document.
- **Chapter 2 "Installing the SNA eWay"**: Describes the system requirements and provides instructions for installing the SNA eWay.
- **Chapter 3 "Configuring the SNA eWay"**: Provides instructions for configuring the eWay to communicate with your legacy systems.
- **Chapter 4 "Implementing the SNA eWay Sample Projects"**: Provides instructions for installing and running the sample Projects.
- **Appendix A "Working with SNA Collaborations"**: Provides guidelines on implementing SNA Java Collaborations.
- **Appendix B "Implementing the SNA Custom Handshake Class"**: Provides instructions on how to deploy a custom handshake class.

### SNA eWay Javadoc

A SNA eWay Javadoc is also provided, that documents the Java methods available with the SNA eWay. The Javadoc is uploaded with the eWay's documentation file (**SNAeWayDocs.sar**) and downloaded from the **Documentation** tab of the Sun Java Composite Application Platform Suite Installer. To access the full Javadoc, extract the Javadoc to an easily accessible folder, and double-click the **index.html** file.

## 1.4.1 Scope

This user's guide provides a description of the SNA eWay Adapter. It includes directions for installing the eWay, configuring the eWay properties, and implementing the eWay's sample Projects. This document is also intended as a reference guide, listing available properties, functions, and considerations. For a reference of available SNA eWay Java methods, see the associated Javadoc.

## 1.4.2 Intended Audience

This guide is intended for experienced computer users who have the responsibility of helping to set up and maintain a fully functioning Java Composite Application Platform Suite system. This person must also understand any operating systems on which the Java Composite Application Platform Suite will be installed (Windows and UNIX), and must be thoroughly familiar with Windows-style GUI operations.

Developers that choose to create projects with the exposed CPIC Java methods provided by this eWay should be expert CPIC programmers that possess extensive knowledge and understanding of CPIC. To use either the exposed CPIC Java methods or the Helper Java methods to create your eWay Collaborations, you should have a working knowledge and understanding of SNA LU6.2.

## 1.4.3 Text Conventions

The following conventions are observed throughout this document.

**Table 2** Text Conventions

| Text Convention | Used For | Examples |
|---|---|---|
| **Bold** | Names of buttons, files, icons, parameters, variables, methods, menus, and objects | ▪ Click **OK**.<br>▪ On the **File** menu, click **Exit**.<br>▪ 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 13 |
| Blue underlined | Hypertext links for Web addresses (URLs) or email addresses | http://www.sun.com |

## 1.5 Related Documents

The following Sun documents provide additional information about the Sun Java Composite Application Platform Suite product:

- *Sun SeeBeyond eGate™ Integrator User's Guide*
- *Sun Java Composite Application Platform Suite Installation Guide*

## 1.6 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

## 1.7 Documentation Feedback

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

CAPS_docsfeedback@sun.com

Chapter 2

# Installing the SNA eWay

This chapter describes the requirements and procedures for installing the SNA eWay.

**What's In This Chapter:**



## 2.1 SNA eWay System Requirements

The SNA eWay Readme contains the latest information on:

- Supported Operating Systems
- System Requirements
- External System Requirements

The SNA eWay Readme is uploaded with the eWay's documentation file (**SNAeWayDocs.sar**) and can be accessed from the **Documentation** tab of the Sun Java Composite Application Platform Suite Installer. Refer to the SNA eWay Readme for the latest requirements before installing the SNA eWay.

## 2.2 Installing the SNA eWay

The Sun 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.

**Note:** *When the Repository is running on a UNIX operating system, the eWays are loaded from the Sun Java Composite Application Platform Suite Installer running on a Windows platform connected to the Repository server using Internet Explorer.*

### 2.2.1 Installing the SNA eWay on an eGate supported system

Follow the directions for installing the Sun Java Composite Application Platform Suite in the *Sun Java Composite Application Platform Suite Installation Guide.* After you have installed Core Products, do the following:

1. From the Sun Java Composite Application Platform Suite Installer's **Select Sun Java Composite Application Platform Suite Products to Install** table (Administration tab), click the **Click to install additional products** link.
2. Expand the **eWay** option.
3. Select the products for your Sun Java Composite Application Platform Suite and include the following:
   - **File eWay** (the File eWay is used by most sample Projects)
   - **SNAeWay**

   To upload the SNA eWay User's Guide, Help file, Javadoc, Readme, and sample Projects, expand the **Documentation** option and select **SNAeWayDocs**.
4. Once you have selected all of your products, click **Next** in the top-right or bottom-right corner of the **Select Sun 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 the last SAR file has been selected.
6. Once your product's installation is finished, continue installing the Sun Java Composite Application Platform Suite as instructed in the *Sun Java Composite Application Platform Suite Installation Guide*.

## Adding the eWay to an Existing Sun Java Composite Application Platform Suite Installation

If you are adding the eWay to an existing Sun Java Composite Application Platform Suite installation, do the following:

1. Complete steps 1 through 4 above.
2. Once your product's installation is complete, open the Enterprise Designer and select **Update Center** from the Tools menu. The **Update Center Wizard** appears.
3. For Step 1 of the wizard, simply click **Next**.
4. For Step 2 of the wizard, click the **Add All** button to move all installable files to the **Include in Install** field, then click **Next**.
5. For Step 3 of the wizard, wait for the modules to download, then click **Next**.
6. The wizard's Step 4 window displays the installed modules. Review the installed modules and click **Finish.**

7 When prompted, restart the IDE (Integrated Development Environment) to complete the installation.

### After Installation

Once you install the eWay, it must then be incorporated into a Project before it can perform its intended functions. See the *eGate Integrator User's Guide* for more information on incorporating the eWay into an eGate Project.

## 2.2.2 Extracting the Sample Projects and Javadocs

The SNA eWay includes sample Projects and Javadocs. The sample Projects are designed to provide you with a basic understanding of how certain database operations are performed using the eWay, while Javadocs provide a list of classes and methods exposed in the eWay.

**Steps to extract the Javadoc include:**

1. Click the **Documentation** tab of the Sun Java Composite Application Platform Suite Installer, then click the Add-ons tab.
2. Click the **SNA eWay Adapter** link. Documentation for the SNA eWay appears in the right pane.
3. Click the icon next to **Javadoc** and extract the ZIP file.
4. Open the index.html file to view the Javadoc.

**Steps to extract the Sample Projects include:**

1. Click the **Documentation** tab of the Sun Java Composite Application Platform Suite Installer, then click the **Add-ons** tab.
2. Click the **SNA eWay Adapter** link. Documentation for the SNA eWay appears in the right pane.
3. Click the icon next to **Sample Projects** and extract the sample project file **SNA_eWay_Sample.zip**.

   Refer to **"Importing a Sample Project" on page 44** for instructions on importing the sample Project into your repository via the Enterprise Designer.

# 2.3 ICAN 5.0 Project Migration Procedures

This section describes how to transfer your current ICAN 5.0.x Projects to the Sun Java Composite Application Platform Suite 5.1.1. To migrate your ICAN 5.0.x Projects to the Sun Java Composite Application Platform Suite 5.1.1, do the following:

**Export the Project**

1. Before you export your Projects, save your current ICAN 5.0.x Projects to your Repository.

2. From the Project Explorer, right-click your Project and select **Export** from the shortcut menu. The Export Manager appears.
3. Select the Project that you want to export in the left pane of the Export Manager and move it to the Selected Projects field by clicking the **Add to Select Items** (arrow) button, or click **All** to include all of your Projects.
4. In the same manner, select the Environment that you want to export in the left pane of the Export Manager and move it to the Selected Environments field by clicking the **Add to Select Items** (arrow) button, or click **All** to include all of your Environments.
5. Browse to select a destination for your Project ZIP file and enter a name for your Project in the **ZIP file** field.
6. Click **Export** to create the Project ZIP file in the selected destination.

### Install Java CAPS 5.1.1

1. Install the **Java CAPS 5.1.1**, including all eWays, libraries, and other components used by your ICAN 5.0 Projects.
2. Start the Java CAPS 5.1.1 Enterprise Designer.

### Import the Project

1. From the Java CAPS 5.1.1 Enterprise Designer's Project Explorer tree, right-click the Repository and select **Import Project** from the shortcut menu. The Import Manager appears.
2. Browse to and select your exported Project file.
3. Click **Import**. A warning message, **"Missing APIs from Target Repository**," may appear at this time. This occurs because various product APIs were installed on the ICAN 5.0 Repository when the Project was created that are not installed on the Java CAPS 5.1.1 Repository. These APIs may or may not apply to your Projects. You can ignore this message if you have already installed all of the components that correspond to your Projects. Click **Continue** to resume the Project import.
4. Close the Import Manager after the Project is successfully imported.

### Deploy the Project

1. A new Deployment Profile must be created for each of your imported Projects. When a Project is exported, the Project's components are automatically *"checked in"* to Version Control to write-protected each component. These protected components appear in the Explorer tree with a red padlock in the bottom-left corner of each icon. Before you can deploy the imported Project, the Project's components must first be *"checked out"* of Version Control from both the Project Explorer and the Environment Explorer. To *"check out"* all of the Project's components, do the following:
   A. From the Project Explorer, right-click the Project and select **Version Control > Check Out** from the shortcut menu. The Version Control - Check Out dialog box appears.
   B. Select **Recurse Project** to specify all components, and click **OK**.

C Select the Environment Explorer tab, and from the Environment Explorer, right-click the Project's Environment and select **Version Control > Check Out** from the shortcut menu.

D Select **Recurse Environment** to specify all components, and click **OK**.

2 If your imported Project includes File eWays, these must be reconfigured in your Environment prior to deploying the Project.

To reconfigure your File eWays, do the following:

A From the Environment Explorer tree, right-click the File External System, and select **Properties** from the shortcut menu. The Properties Editor appears.

B Set the inbound and outbound directory values, and click **OK**. The File External System can now accommodate both inbound and outbound eWays.

3 Deploy your Projects.

**Note:** *Only projects developed on ICAN 5.0.2 and later can be imported and migrated successfully into the Sun Java Composite Application Platform Suite.*

## 2.4 Installing Enterprise Manager eWay Plug-Ins and Bridge Files

The **Sun SeeBeyond Enterprise Manager** is a Web-based interface you use to monitor and manage your Sun Java Composite Application Platform Suite applications. The Enterprise Manager requires an eWay specific "plug-in" for each eWay you install. These plug-ins enable the Enterprise Manager to target specific alert codes for each eWay type, as well as start and stop the inbound eWays.

The *Sun Java Composite Application Platform Suite Installation Guide* describes how to install Enterprise Manager. The *Sun SeeBeyond eGate Integrator System Administration Guide* describes how to monitor servers, Services, logs, and alerts using the Enterprise Manager and the command-line client.

The eWay Enterprise Manager Plug-ins are available from the **List of Components to Download** under the Sun Java Composite Application Platform Suite Installer's **Downloads** tab.

**Figure 6** Java CAPS Installer - SNA Plug-in and Bridge files

| |
|---|
| SNALU62 eWay Enterprise Manager Plug-In |
| SNALU62 eWay - Runtime win32 bridge DLL |
| SNALU62 eWay - Runtime aix32 bridge so |
| SNALU62 eWay - Runtime aix64 bridge so |
| SNALU62 eWay - Runtime sparc32 SNAP-IX bridge so |
| SNALU62 eWay - Runtime sparc64 SNAP-IX bridge so |
| SNALU62 eWay - Runtime sparc32 Brixton bridge so |
| SNALU62 eWay - Runtime sparc64 Brixton bridge so |
| SNALU62 eWay - Runtime JNI |

There are two ways to add eWay Enterprise Manager plug-ins:

- From the **Sun Java Composite Application Platform Suite Installer**
- From the **Sun SeeBeyond Enterprise Manager**

### To add plug-ins from the Sun Java Composite Application Platform Suite Installer

1. Navigate to the**Sun Java Composite Application Platform Suite Installer**'s **Downloads** tab. The SNA eWay plug-ins that are available from your Repository appear.
2. Select the plug-ins you require and save them to a temporary directory.
3. Copy the plug-in files to the Logical Host's Integration Server library folder:

   `<JavaCAPS51>\logicalhost\is\lib\`

   where <*JavaCAPS51*> is the directory where the Sun Java Composite Application Platform Suite is installed.
4. From the **Enterprise Manager**'s Explorer toolbar, click **configuration**.
5. Click the **Web Applications Manager** tab and go to the **Manage Applications** sub-tab.
6. Browse for and select the WAR file for the application plug-in that you downloaded, and click **Deploy**. The plug-ins are installed and deployed.

### To add plug-ins from the Enterprise Manager

1. From the **Enterprise Manager**'s Explorer toolbar, click **configuration**.
2. Click the **Web Applications Manager** tab, go to the **Auto-Install from Repository** sub-tab, and connect to your Repository.
3. Select the application plug-ins you require, and click **Install**. The application plug-ins are installed and deployed.

### To add bridge files from the Sun Java Composite Application Platform Suite Installer

1. Copy the Runtime JNI (`snalu62jni.jar`) file to the Integration Server classpath.
2. Copy the corresponding JNI bridge file for your operating system. Available bridge files are displayed in Figure 6.

For win32 JNI bridge files, save the file to a directory that is declared in the system PATH statement (for example, `<c:>\WINNT\system32\`).

For Unix bridge files, follow these steps:

A Save the appropriate bridge file to your local system.

B Move the bridge file to your UNIX directory.

- The target directory for AIX systems must be declared in LIBPATH.
- The target directory for Solaris must be declared in LD_LIBRARY_PATH.

For further information on configuring the Logical Host's Integration Server, see **"Configuring the Logical Host" on page 71**.

## 2.4.1 Viewing Alert Codes

You can view and delete alerts using the Enterprise Manager. An alert is triggered when a specified condition occurs in a Project component. The purpose of the alert is to warn the administrator or user that a condition has occurred.

### To View the eWay Alert Codes

1. Add the eWay Enterprise Manager plug-in for this eWay.
2. From the **Enterprise Manager**'s Explorer toolbar, click **configuration**.
3. Click the **Web Applications Manager** tab and go to the **Manage Alert Codes** sub-tab. Your installed eWay alert codes display under the **Results** section. If your eWay alert codes are not displayed under **Results**, do the following:

   A From the **Install New Alert Codes** section, browse to and select the eWay alert properties file for the application plug-in that you added. The alert properties files are located in the **alertcodes** folder of your Sun Java Composite Application Platform Suite installation directory.

   B Click **Deploy**. The available alert codes for your application are displayed under **Results**. A listing of the eWay's available alert codes is displayed in Table 3.

**Table 3** Alert Codes for the SNA eWay

| Alert Code\Description | Description Details | User Actions |
|---|---|---|
| SNALU62-CONNECT-FAILED000001=Failed to connect. | Occurs during the initial system connection establishment. | ▪ System is down; start your system.<br>▪ External configuration information may be invalid. You may need to verify the configured parameters.<br><br>Refer to the server log for further information. |
| SNALU62-RECEIVE-FAILED000003=Failed to receive message. | Occurs when a collaboration attempts to receive a message, and no message is available to be received. | ▪ Verify that a message is available to be received.<br>▪ External configuration information may be invalid. You may need to verify the configured parameters.<br>▪ Verify the connection logic. |

| Alert Code\Description | Description Details | User Actions |
|---|---|---|
| SNALU62-SEND-FAILED000002=Failed to send message. | Occurs when a destination is not ready to receive a message. | ▪ There is a collaboration error. Verify the collaboration design is valid.<br>▪ External configuration information may be invalid. You may need to verify the configured parameters. |

For information on Managing and Monitoring alert codes and logs, as well as how to view the alert generated by the project component during runtime, see the *Sun SeeBeyond eGate™ Integrator System Administration Guide*.

**Note:** *An alert code is a warning that an error has occurred. It is not a diagnostic. The user actions noted above are just some possible corrective measures you may take. Refer to the log files for more information. For information on Managing and Monitoring alert codes and logs, see the Sun SeeBeyond eGate Integrator System Administration Guide.*

Chapter 3

# Configuring the SNA eWay

This chapter describes how to set the properties of the SNA eWay.

**What's In This Chapter:**



## 3.1 Creating and Configuring a SNA eWay

All eWays contain a unique set of default configuration parameters. After the eWays are established and a SNA External System is created in the Project's Environment, the eWay parameters are modified for your specific system. The SNA eWay configuration parameters are modified from two locations:

- **Connectivity Map**: These parameters most commonly apply to a specific component eWay, and may vary from other eWays (of the same type) in the Project.
- **Environment Explorer** : These parameters are commonly global, applying to all eWays (of the same type) in the Project. The saved properties are shared by all eWays in the SNA External System window.
- **Collaboration or Business Process**: SNA eWay properties may also be set from your Collaboration or Business Process, in which case the settings will override the corresponding properties in the eWay's Connectivity Map configuration. Any properties that are not overridden retain their configured default settings.

## 3.2 Configuring the eWay Connectivity Map Properties

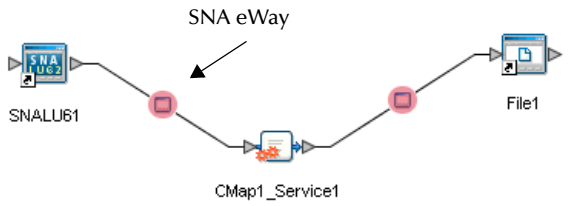
When you connect an External Application to a Collaboration, Enterprise Designer automatically assigns the appropriate eWay to the link. Each eWay is supplied with a template containing default configuration properties that are accessible on the Connectivity Map.

The SNA eWay can be configured for both inbound and outbound modes in a Connectivity Map.

### To Configure the Inbound eWay Properties:

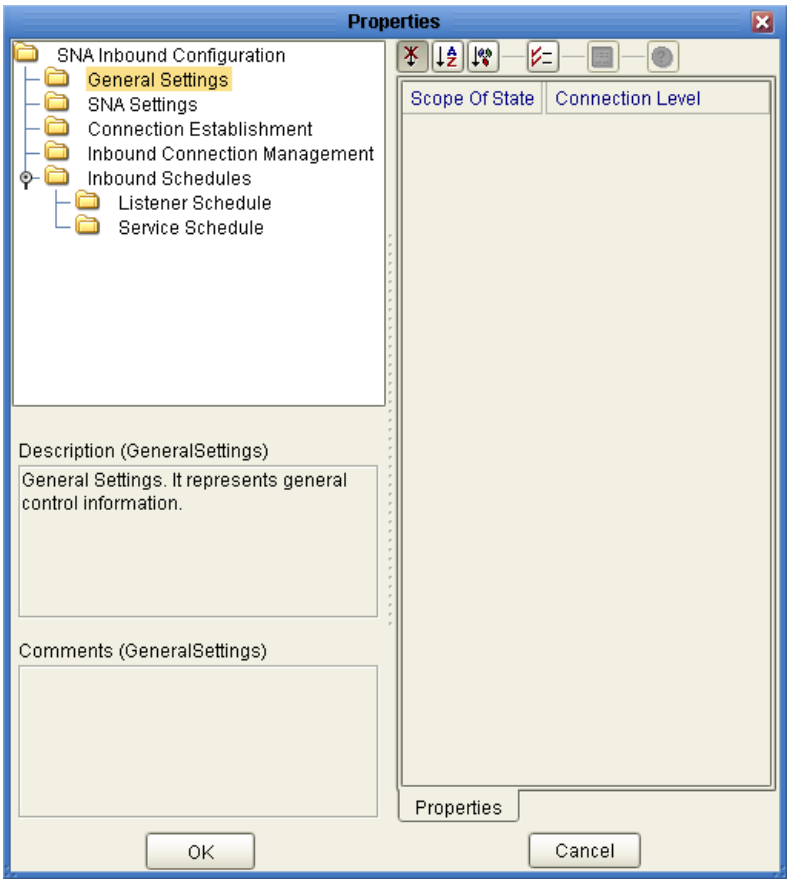
1 On the Enterprise Designer's Connectivity Map, double-click the SNA eWay icon.

**Figure 7** Connectivity Map with Components - Inbound




The eWay Properties window appears, displaying the default properties for the Inbound eWay.

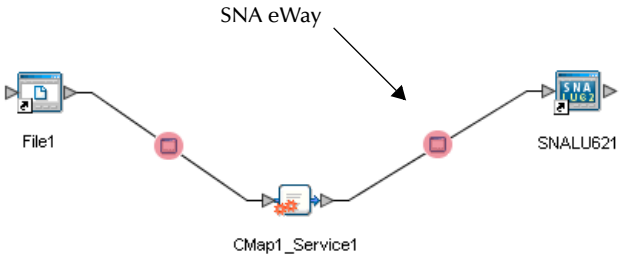
**Figure 8** Inbound eWay Properties




### To Configure the Outbound eWay Properties:

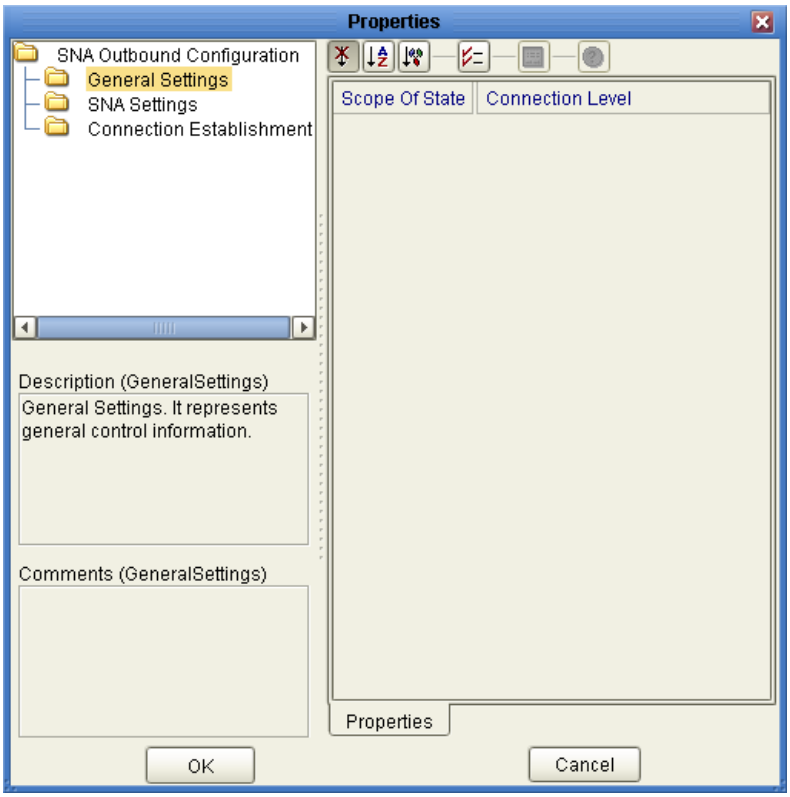
1 On the Enterprise Designer's Connectivity Map, double-click the SNA eWay icon.

**Figure 9** Connectivity Map with Components - Outbound




The eWay Properties window appears, displaying the default properties for the Oubound eWay.

**Figure 10** Outbound eWay Properties




## 3.3 Configuring the eWay Environment Properties

The eWay Environment Configuration properties contain parameters that define how the eWay connects to and interacts with other eGate components within the Environment. When you create a new SNA External System, you may configure the type of External System required.

Available External System properties include:

- SNALU62 Inbound eWay
- SNALU62 Outbound eWay

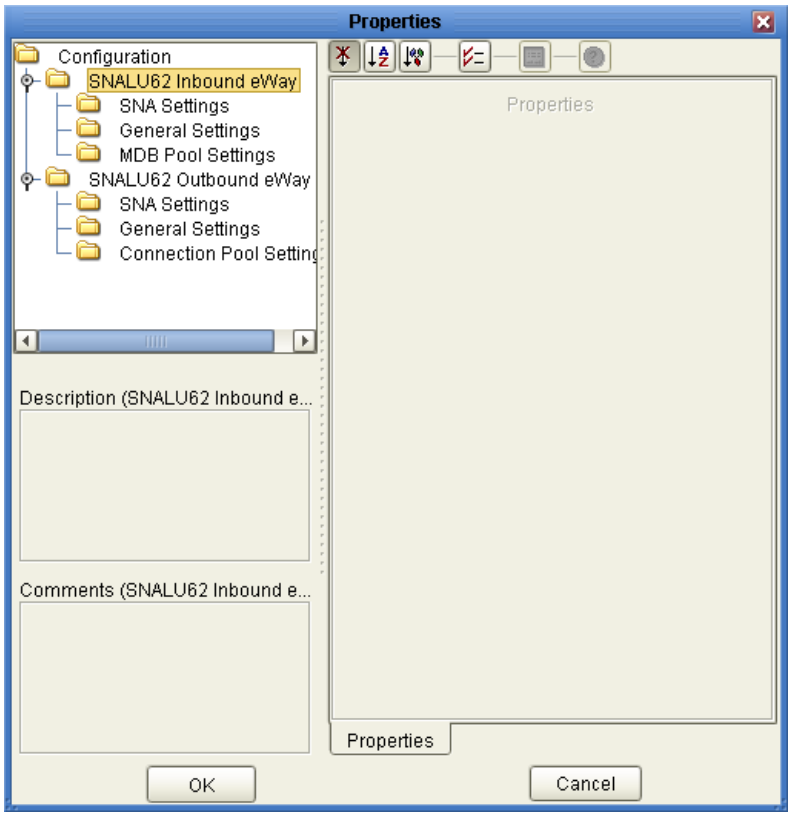
**To Configure the Environment Properties:**

1. In Enterprise Explorer, click the **Environment Explorer** tab.
2. Expand the Environment created for the SNA Project and locate the SNA External System.

**Note:** *For more information on creating an Environment, see the "Sun SeeBeyond eGate Integrator Tutorial."*

3. Right-click the External System created for the SNA Project and select **Properties** from the list box. The **Environment Configuration Properties** window appears.

**Figure 11** SNA eWay Environment Configuration




4. Click on any folder to display the default configuration properties for that section.
5. Click on any property field to make it editable.

   After modifying the configuration properties, click **OK** to save the changes.

## 3.4 eWay Connectivity Map Properties

The eWay Connectivity Map consists of the following properties categories.

**SNA Inbound eWay Configuration Sections Include:**


- **Connectivity Map Inbound eWay General Settings** on page 27
- **Connectivity Map Inbound eWay SNA Settings** on page 28
- **Connectivity Map Inbound eWay Connection Establishment** on page 29
- **Connectivity Map Inbound eWay Inbound Connection Management** on page 29
- **Connectivity Map Inbound eWay Inbound Schedules** on page 30


**SNA Outbound eWay Configuration Sections Include:**


- **Connectivity Map Outbound eWay General Settings** on page 33
- **Connectivity Map Outbound eWay SNA Settings** on page 34
- **Connectivity Map Outbound eWay Connection Establishment** on page 35


### 3.4.1 Connectivity Map Inbound eWay General Settings

The Inbound eWay General Settings are included in Table 4.

**Table 4** Inbound eWay—General Settings

| Name | Description | Required Value |
|---|---|---|
| **Scope of State** | Defines the scope of the State object, which is an OTD sub-node. | The valid options for this parameter are:<br>▪ **Connection Level**: The State has the same life cycle as the connection.<br>▪ **Resource Adapter Level**: The State has the same life cycle as the resource adapter. The life terminates when the resource adapter is recycled.<br>▪ **OTD Level**: The State has the same life cycle as the OTD object. This scope represents the life cycle of the State. The life terminates when the collaboration finishes.<br>The default is **Connection Level**. |

## 3.4.2 Connectivity Map Inbound eWay SNA Settings

The Inbound eWay SNA Settings are included in Table 5.

**Table 5** Inbound eWay—SNA Settings

| Name | Description | Required Value |
|---|---|---|
| **Packet Size** | The number of bytes per packet of data. This number also determines the size of the buffers. | A valid numeric value. The default is **1024**. |
| **Timeout** | Specifies the milliseconds of pause before receiving a response from a server following a sent request. | A valid numeric value. The default is **1000**. |
| **Initialize Conversation** | Specifies how the eWay will establish a SNA conversation. Options are:<br><br>▪ **TRUE**: The eWay will initialize SNA conversations as an invoking TP.<br>▪ **FALSE**: The eWay will accept SNA conversations as an invokable TP. | Select **TRUE** or **FALSE**. The default is **FALSE**. |
| **Deallocation Type** | Specifies the type of deallocation required at the end of a conversation when a shutdown is issued. Please refer to your SNA documentation for more information. | Select one of the following four options:<br><br>▪ **0 - SYNC_LEVEL**.<br>▪ **1 - FLUSH**.<br>▪ **2 - CONFIRM**.<br>▪ **3 - ABEND**.<br><br>The default is **0 - SYNC_LEVEL**. |
| **Synchronization Level** | Specifies the synchronization level parameter (CM_SYNC_LEVEL). Please refer to your SNA manual for more information.<br>**0 - None** (Default)<br>**1 - Confirm** | Select **0 - None** or **1 - Confirm**. The default is **0 - None**. |
| **Custom Handshake Class Name** | Defines your SNA handshake logic (see **Appendix B** to deploy a custom handshake class). | A fully qualified class name such as **com.abc.MyClass**. The class must implement the interface **com.stc.connector.snalu62.api.snaCustomerHandshake**. No value (leaving the property blank) indicates that no SNA conversation handshake logic is defined. Instead, a built-in standard handshake logic is used. |

### 3.4.3 Connectivity Map Inbound eWay Connection Establishment

The Inbound eWay Connection Establishment properties are included in Table 6.

**Table 6** Inbound eWay—Connection Establishment

| Name | Description | Required Value |
|---|---|---|
| **Max Connection Retry** | Specifies the maximum number of retries to establish a connection upon failure to acquire one. | A valid numeric value. The default is **3**. |
| **Retry Connection Interval** | Specifies the milliseconds of pause before each attempt to reaccess the SNA LU62 destination. This setting is used in conjunction with the **Max Connection Retry** setting.<br><br>For example: In the event that the eWay cannot connect to the SNA destination, the eWay will try to reconnect three times in 30 second intervals when the Connection Retries value is 3 and the Connection Retry Interval is 30000. | A valid numeric value. The default is **30000**. |

### 3.4.4 Connectivity Map Inbound eWay Inbound Connection Management

The Inbound eWay Inbound Connection Management properties are included in Table 7.

**Table 7** Inbound eWay—Inbound Connection Management

| Name | Description | Required Value |
|---|---|---|
| **Max Connection Pool Size** | Defines the maximum number of concurrent connections for the particular listener/monitor over the specified SNALU62 destination. **0** (zero) indicates that there is no maximum. | A valid numeric value. The default is **50.** |

**Table 7** Inbound eWay—Inbound Connection Management (Continued)

| Name | Description | Required Value |
|---|---|---|
| **Scope of Connection** | Defines the scope of the accepted connection used by the eWay. Options are:<br><br>▪ **Collaboration Level**: The connection will be closed once the execution of the Collaboration is completed. The connection has the same life cycle as the Collaboration.<br>▪ **Resource Adaptor Level**: The resource adapter will close the connection upon closure request. The connection may remain live across multiple executions of the Collaboration. | Select **Collaboration Level** or **Resource Adapter Level**. The default is **Resource Adapter Level**. |

## 3.4.5 Connectivity Map Inbound eWay Inbound Schedules

### Listener Schedule

Listner Schedule properties specify the schedule that the server must wait for the new client connection establishment request. This schdeule is for the listener/monitor. The Listener Schedule properties are included in Table 8.

**Table 8** Inbound Schedules—Listener Schedule

| Name | Description | Required Value |
|---|---|---|
| **Scheduler** | Specifies the scheduler type for this inbound communication. Options are:<br><br>▪ **Timer Service**: The task is scheduled according to the **Schedule Type**, **Delay**, **Period**, and **At Fixed Rate** values.<br>▪ **Work Manager**: The work is scheduled according to the **Schedule Type**, **Delay**, and **Period** values.<br><br>If your container does not support JCA Work Management (prior to JCA1.5), select **Timer Service**. | Select **Timer Service** or **Work Manager**. The default is **Work Manager**. |
| **Schedule Type** | Defines the type of schedule for inbound communication. **Repeated** indicates a task is scheduled for repeated execution at regular intervals defined by the parameter **Period** (see below). | The configured default is **Repeated**.<br><br>***Note:*** This value cannot be changed. |
| **Delay** | Specifies the delay in milliseconds before a task is executed. For further details, refer to the SNA eWay Javadoc. | A valid numeric value. The default is **0**. |

**Table 8** Inbound Schedules—Listener Schedule (Continued)

| Name | Description | Required Value |
|---|---|---|
| **Period** | Specifies the regular interval in milliseconds between successive task executions. This parameter is used in conjunction with the **Schedule Type** parameter when set to **Repeated**. | A valid numeric value. The default is **100.** |
| **At Fixed Rate** | Used in conjunction with the **Repeated** setting for the **Schedule Type** parameter and the **Timer Service** type of **Scheduler**. Options are:<br><br>▪ **TRUE**: Denotes a fixed rate. Each execution is scheduled relative to the scheduled time of the initial execution. If an execution is delayed for any reason, two or more executions will occur in rapid succession to return to the preset execution schedule. Overall, the frequency of executions will be exactly the reciprocal of the specified period.<br>▪ **FALSE**: Denotes a fixed delay. Each execution is scheduled relative to the actual execution time of the previous execution. If an execution is delayed for any reason, subsequent executions are delayed as well. Overall, the frequency of executions will generally be lower than the reciprocal of the specified period. | Select **TRUE** or **FALSE**. The default is **FALSE**. |

## Service Schedule

The Inbound eWay Service Schedule properties are included in Table 9.

**Table 9** Inbound Schedules—Service Schedule

| Name | Description | Required Value |
|---|---|---|
| **Scheduler** | Specifies the scheduler type for this inbound communication. Options are:<br><br>▪ **Timer Service**: The task is scheduled according to the **Schedule Type**, **Delay**, **Period**, and **At Fixed Rate** values.<br>▪ **Work Manager**: The work is scheduled according to the **Schedule Type**, **Delay**, and **Period** values.<br><br>If your container does not support JCA Work Management (prior to JCA1.5), select **Timer Service**. | Select **Timer Service** or **Work Manager**. The default is **Work Manager**. |
| **Schedule Type** | Defines the type of schedule for inbound communication. Options are:<br>▪ **One Time**: A task is scheduled for a one-time execution.<br>▪ **Repeated**: A task is scheduled for repeated execution at regular intervals defined by the parameter **Period** (see below). | Select **One Time** or **Repeated**. The default is **Repeated**. |
| **Delay** | Specifies the delay in milliseconds before a task is executed. For further details, refer to the SNA eWay Javadoc. | A valid numeric value. The default is **0**. |
| **Period** | Specifies the regular interval in milliseconds between successive task executions. This parameter is used in conjunction with the **Schedule Type** parameter when set to **Repeated**. | A valid numeric value. The default is **100.** |

**Table 9** Inbound Schedules—Service Schedule (Continued)

| Name | Description | Required Value |
|---|---|---|
| **At Fixed Rate** | Used in conjunction with the **Repeated** setting for the **Schedule Type** parameter and the **Timer Service** type of **Scheduler**. Options are:<br><br>▪ **TRUE**: Denotes a fixed rate. Each execution is scheduled relative to the scheduled time of the initial execution. If an execution is delayed for any reason, two or more executions will occur in rapid succession to return to the preset execution schedule. Overall, the frequency of executions will be exactly the reciprocal of the specified period.<br>▪ **FALSE**: Denotes a fixed delay. Each execution is scheduled relative to the actual execution time of the previous execution. If an execution is delayed for any reason, subsequent executions are delayed as well. Overall, the frequency of executions will generally be lower than the reciprocal of the specified period. | Select **TRUE** or **FALSE**. The default is **FALSE**. |

## 3.4.6 Connectivity Map Outbound eWay General Settings

The Outbound eWay General Settings are included in Table 10.

**Table 10** Outbound eWay—General Settings

| Name | Description | Required Value |
|---|---|---|
| **Scope of State** | Defines the scope of the State object, which is an OTD sub-node. | The valid options for this parameter are:<br>▪ **Connection Level**: The State has the same life cycle as the connection.<br>▪ **Resource Adapter Level**: The State has the same life cycle as the resource adapter. The life terminates when the resource adapter is recycled.<br>▪ **OTD Level**: The State has the same life cycle as the OTD object. This scope represents the life cycle of the State. The life terminates when the collaboration finishes.<br>The default is **Connection Level**. |

## 3.4.7 Connectivity Map Outbound eWay SNA Settings

The Outbound eWay SNA Settings are included in Table 11.

**Table 11** Outbound eWay—SNA Settings

| Name | Description | Required Value |
|---|---|---|
| **Packet Size** | The number of bytes per packet of data. This number also determines the size of the buffers. | A valid numeric value. The default is **1024**. |
| **Timeout** | Specifies the milliseconds of pause before receiving a response from a server following a sent request. | A valid numeric value. The default is **1000**. |
| **Initialize Conversation** | Specifies how the eWay will establish a SNA conversation. Options are:<br>▪ **TRUE**: The eWay will initialize SNA conversations as an invoking TP.<br>▪ **FALSE**: The eWay will accept SNA conversations as an invokable TP. | Select **TRUE** or **FALSE**. The default is **TRUE**. |

**Table 11** Outbound eWay—SNA Settings (Continued)

| Name | Description | Required Value |
|---|---|---|
| **Deallocation Type** | Specifies the type of deallocation required at the end of a conversation when a shutdown is issued. Please refer to your SNA documentation for more information. | Select one of the following four options:<br><br>▪ **0 - SYNC_LEVEL**.<br>▪ **1 - FLUSH**.<br>▪ **2 - CONFIRM**.<br>▪ **3 - ABEND**.<br><br>The default is **0 - SYNC_LEVEL**. |
| **Synchronization Level** | Specifies the synchronization level parameter (CM_SYNC_LEVEL). Please refer to your SNA manual for more information.<br>**0 - None** (Default)<br>**1 - Confirm** | Select one of the following two options:<br><br>▪ **0 - None**.<br>▪ **1 - Confirm**.<br><br>The default is **0 - None**. |
| **Custom Handshake Class Name** | Defines your SNA handshake logic (see **Appendix B** to deploy a custom handshake class). | A fully qualified class name such as **com.abc.MyClass**. The class must implement the interface **com.stc.connector.snalu62.api.snaCustomerHandshake**. No value (leaving the property blank) indicates that no SNA conversation handshake logic is defined. Instead, a built-in standard handshake logic is used. |

## 3.4.8 Connectivity Map Outbound eWay Connection Establishment

The Outbound eWay Connection Establishment properties are included in Table 12.

**Table 12** Outbound eWay—Connection Establishment

| Name | Description | Required Value |
|---|---|---|
| **Connection Mode** | Specifies how or when a connection will become available. Options are:<br><br>▪ **AUTOMATIC**: The eWay will establish a SNA conversation automatically.<br>▪ **MANUAL**: The SNA conversation will become available to you only when you manually call the OTD function **startConversation()** from the Collaboration; the conversation will become unavailable when you call the OTD function **endConversation()**.<br><br>***Note:*** The OTD functions **startConversation()** and **endConversation()** are expected for Manual mode only. Automatic mode does not allow you to call them explicitly. | Select **AUTOMATIC** or **MANUAL**. The default is **AUTOMATIC**. |
| **Max Connection Retry** | Specifies the maximum number of retries to establish a connection upon failure to acquire one. | A valid numeric value. The default is **3**. |
| **Retry Connection Interval** | Specifies the milliseconds of pause before each attempt to reaccess the SNA LU62 destination. This setting is used in conjunction with the **Max Connection Retry** setting.<br><br>For example: In the event that the eWay cannot connect to the SNA destination, the eWay will try to reconnect three times in 30 second intervals when the Connection Retries value is 3 and the Connection Retry Interval is 30000. | A valid numeric value. The default is **30000**. |
| **Always Create New Connection** | Specifies whether to ALWAYS attempt to create a new connection for a connection establishment request. Options are:<br><br>▪ **TRUE**: The eWay will always attempt to create a new connection without trying to match connection.<br>▪ **FALSE**: The eWay will attempt to match an existing connection. | Select **TRUE** or **FALSE**. The default is **FALSE**. |

**Table 12** Outbound eWay—Connection Establishment (Continued)

| Name | Description | Required Value |
|---|---|---|
| **Auto Reconnect Upon Matching Failure** | Specifies whether or not to make an attempt to re-connect automatically after getting a matched connection from a container. Options are:<br><br>▪ **TRUE**: The eWay will discard the invalid matched connection and will attempt to establish another connection automatically.<br>▪ **FALSE**: The eWay will not attempt to to establish a new connection automatically. Instead, control will be deferred to your business rules which will detect this type of failure and perform the desired operations accordingly. | Select **TRUE** or **FALSE**. The default is **TRUE**. |
| **Auto Disconnect Connection** | Specifies whether the eWay disconnects automatically after the work on the connection is completed. Options are:<br><br>▪ **TRUE**: The eWay connection will be disconnected and it will not be re-used.<br>▪ **FALSE**: The connection will be left for reuse. | Select **TRUE** or **FALSE**. The default is **FALSE**. |

## 3.5 eWay Environment Properties

eWay External System properties must be configured from within the Environment. Until you have successfully configured all eWays for your CAPS project, your project cannot be properly executed. The following list identifies the SNA eWay properties. There are six Environment Configuration categories that the SNA eWay implements.

**Property Categories Configured in the Logical Host Environment**

- **SNALU62 Inbound eWay Properties** on page 37
- **SNALU62 Outbound eWay Properties** on page 39

### 3.5.1 SNALU62 Inbound eWay Properties

Before deploying your eWay, you will need to set the Environment properties. The Inbound SNA eWay includes the following configuration sections:

- SNA Settings
- General Settings
- MDB Pool Settings

## SNA Settings

Details for the SNALU62 Inbound eWay SNA Settings are listed in Table 13.

**Table 13** SNALU62 Inbound eWay—SNA Settings

| Name | Description | Required Value |
|---|---|---|
| **Host Name** | Specifies the host name where the LU62 Server runs.<br><br>***Note:*** This parameter is only required for the Brixton LU62 server and is ignored on other platforms. | Any valid string. The default is **localhost**. |
| **Symbolic Dest Name** | Specifies the symbolic destination name associated with a side information entry loaded from the configuration file. Refer to your SNA documentation for more information. | Any valid string.<br><br>***Note:*** This parameter is case-sensitive. |
| **Local LU Name** | Specifies the local LU name defined to the SunLink LU62 server. Refer to your SNA documentation for more information.<br><br>***Note:*** This parameter is required for SunLink P2P LU6.2 9.1 and is ignored on other platforms. | Any valid string.<br><br>***Note:*** This parameter is case-sensitive. |
| **Local TP Name** | Specifies the local Transaction Program (TP) name that is running on the local LU. Refer to your SNA documentation for more information. | Any valid string.<br><br>***Note:*** This parameter is case-sensitive. |

## General Settings

Details for the SNALU62 Inbound eWay General Settings are listed in Table 14.

**Table 14** SNALU62 Inbound eWay—General Settings

| Name | Description | Required Value |
|---|---|---|
| **Persistent Storage Location** | Specifies the Persistent Location (a local folder path and name) that contains the file used to store the persistent data. The base file name will be generated according to the project, deployment, and Collaboration information. | The absolute path and name of the directory. The default is: **/temp/snalu62inbound/persist**. |

## MDB Pool Settings

Details for the SNALU62 Inbound eWay MDB Pool Settings are listed in Table 15.

**Table 15** SNALU62 Inbound eWay—MDB Pool Settings

| Name | Description | Required Value |
|---|---|---|
| **Steady Pool Size** | Specifies the minimum number of physical connections the pool should keep available at all times. **0** (zero) indicates that there should be no physical connections in the pool and the new connections should be created as needed.<br><br>If the pool size is too small, you may experience a longer connection time due to the existing number of physical connections.<br><br>A connection that stays in the pool allows transactions to use it via a logical connection which is faster. | A valid numeric value. The default is **10**. |
| **Max Pool Size** | Specifies the maximum number of physical connections the pool should keep available at all times. **0** (zero) indicates that there is no maximum.<br><br>The pool size you set depends on the transaction volume and response time of the application. If the pool size is too big, you may end up with too many connections to the SNA destination. | A valid numeric value. The default is **60**. |
| **Pool Idle Timeout in Seconds** | Specifies the maximum number of seconds that a physical connection may remain unused before it is closed. **0** (zero) indicates that there is no limit. | A valid numeric value. The default is **600**. |

## 3.5.2 SNALU62 Outbound eWay Properties

Before deploying your eWay, you will need to set the Environment properties. The Outbound SNA eWay includes the following configuration sections:

- SNA Settings
- General Settings
- Connection Pool Settings

### SNA Settings

Details for the SNALU62 Outbound eWay SNA Settings are listed in Table 16.

**Table 16** SNALU62 Outbound eWay—SNA Settings

| Name | Description | Required Value |
|---|---|---|
| **Host Name** | Specifies the host name where the LU62 Server runs.<br><br>***Note:*** This parameter is only required for the Brixton LU62 server and is ignored on other platforms. | Any valid string. The default is **localhost**. |
| **Symbolic Dest Name** | Specifies the symbolic destination name associated with a side information entry loaded from the configuration file. Refer to your SNA documentation for more information. | Any valid string.<br><br>***Note:*** This parameter is case-sensitive. |
| **Local LU Name** | Specifies the local LU name defined to the SunLink LU62 server. Refer to your SNA documentation for more information.<br><br>***Note:*** This parameter is required for SunLink P2P LU6.2 9.1 and is ignored on other platforms. | Any valid string.<br><br>***Note:*** This parameter is case-sensitive. |
| **Local TP Name** | Specifies the local Transaction Program (TP) name that is running on the local LU. Refer to your SNA documentation for more information. | Any valid string.<br><br>***Note:*** This parameter is case-sensitive. |

## General Settings

Details for the SNALU62 Outbound eWay General Settings are listed in Table 17.

**Table 17** SNALU62 Outbound eWay—General Settings

| Name | Description | Required Value |
|---|---|---|
| **Persistent Storage Location** | Specifies the Persistent Location (a local folder path and name) that contains the file used to store the persistent data. The base file name will be generated according to the project, deployment, and Collaboration information. | The absolute path and name of the directory. The default is: **/temp/snalu62outbound/persist**. |

## Connection Pool Settings

Details for the SNALU62 Outbound eWay Connection Pool Settings are listed in Table 18.

**Table 18** SNALU62 Outbound eWay—Connection Pool Settings

| Name | Description | Required Value |
|---|---|---|
| **Steady Pool Size** | Specifies the minimum number of physical connections the pool should keep available at all times. **0** (zero) indicates that there should be no physical connections in the pool and the new connections should be created as needed.<br><br>If the pool size is too small, you may experience a longer connection time due to the existing number of physical connections.<br><br>A connection that stays in the pool allows transactions to use it via a logical connection which is faster. | A valid numeric value. The default is **1**. |
| **Max Pool Size** | Specifies the maximum number of physical connections the pool should keep available at all times. **0** (zero) indicates that there is no maximum.<br><br>The pool size you set depends on the transaction volume and response time of the application. If the pool size is too big, you may end up with too many connections to the SNA destination. | A valid numeric value. The default is **32**. |
| **Pool Idle Timeout in Seconds** | Specifies the maximum number of seconds that a physical connection may remain unused before it is closed. **0** (zero) indicates that there is no limit. | A valid numeric value. The default is **300**. |

## 3.6 Object Type Definitions (OTDs)

Unlike most other eWays, the SNA eWay does not consist of an OTD wizard. OTD wizards typically facilitate the creation of a Collaborations that are used with eWay projects. When an OTD wizard is available, a skeleton Collaboration is created to provide minimal funtionality that you must modify to suit your application's needs. Without the OTD wizard, as in the case of the SNA eWay, you must create your Collaborations completely from scratch.

**To associate the standard SNA eWay OTD to a new Java Collaboration:**

1. From the Project Explorer, right-click the targeted project.
2. Select **New > Collaboration Definition (Java)...**
3. Complete steps 1 and 2 of the **Collaboration Definition Wizard (Java)**.

4. Select the OTD to use in the new Collaboration by traversing the **Look In** drop-down box: SeeBeyond.eWays.SNALU62.
5. Highlight the desired OTD name and click the **Add** button.
6. Optionally, modify the instance name of the OTD that will be used in the Collaboration.
7. Click the **Finish** button.

The new Collaboration that implements the SNA eWay OTD is created. For details about the SNA eWay methods that may be used with Collaborations for the, refer to the associated Javadoc.

Chapter 4

# Implementing the SNA eWay Sample Projects

This chapter provides an introduction to the SNA eWay components, and information on how these components are created and implemented in a Sun Java Composite Application Platform Suite Project. Sample Projects are designed to provide an overview of the basic functionality of the SNA eWay by identifying how information is passed between eGate and supported external systems.

It is assumed that you understand the basics of creating a Project using the Enterprise Designer. For more information on creating an eGate Project, see the *Sun SeeBeyond eGate™ Tutorial* and the *Sun SeeBeyond eGate™ Integrator User's Guide*.

**What's in This Chapter**



## 4.1 About the SNA eWay Sample Project

The SNA sample Project demonstrates how the SNA eWay processes information from a SNA system. The SNA eWay **SNA_eWay_Sample.zip** file contains one sample Project that provides basic instruction on using SNA operations in the Java Collaboration Definition (JCD) environment

The sample Project uses the **SNACPIC_input.txt.~in** input file to pass data into the **jcdSNACPIC_Inbound** and **jcdSNACPIC_Outbound** Collaborations. These two Collaborations demonstrate the capabilities of the SNA eWay using CPiC functions.

The sample Project uses the **SNAHelper_input.txt.~in** input file to pass data into the **jcdSNAHelper_Inbound** and **jcdSNAHelper_Outbound** Collaborations. These two Collaborations demonstrate how to use the SNA eWay with the Helper functions.

The Helper methods should be used if you are not familiar with CPIC methods. They offer less flexibility than CPIC, but are relatively simple to implement and should be considered only for simple project scenarios.

## 4.2 Running the Sample Project

The following steps are required to run the sample Project that is contained in the **SNAeWayDocs.sar** file.

1. From your input directory, paste (or rename) the sample input file to trigger the eWay.
2. Import the sample Project.
3. Build, deploy, and run the sample Project.

   You must do the following before you can run an imported sample Project:

   - Create an Environment
   - Configure the eWays
   - Create a Deployment Profile
   - Create and start a domain
   - Deploy the Project
4. Check the output.

## 4.3 Importing a Sample Project

Sample eWay Projects are included as part of the installation CD-ROM package. To import a sample eWay Project to the Enterprise Designer do the following:

1. Extract the samples from the Sun Java Composite Application Platform Suite Installer to a local file.

   Sample files are uploaded with the eWay's documentation SAR file, and then downloaded from the Installer's **Documentation** tab. The **SNA_eWay_Sample.zip** file contains the various sample Project ZIP files.

**Note:** *Make sure you save all unsaved work before importing a Project.*

2. From the Enterprise Designer's Project Explorer pane, right-click the Repository and select **Import Project** from the shortcut menu. The **Import Manager** appears.
3. Browse to the directory that contains the sample Project ZIP file. Select the sample file and click **Import**.
4. Click **Close** after successfully importing the sample Project.

# 4.4 Building, Deploying, and Running the prjSNA_Sample_JCD Sample Project

The following provides step-by-step instructions for manually creating the prjSNA_Sample_JCD sample Project.

Steps required to create the sample project include:



## 4.4.1 Creating a Project

The first step is to create a new Project in the Enterprise Designer.

1. Start the Enterprise Designer.
2. From the Project Explorer tree, right-click the Repository and select **New Project**. A new Project (**Project1**) appears on the Project Explorer tree.
3. Click twice on **Project1** and rename the Project (for this sample, **prjSNA_Sample_JCD**).

## 4.4.2 Creating a Connectivity Map

The Connectivity Map provides a canvas for assembling and configuring a Project's components.

**Steps required to create a new Connectivity Map:**

1. From the Project Explorer tree, right-click the new **prjSNA_Sample_JCD** Project and select **New > Connectivity Map** from the shortcut menu.
2. The New Connectivity Map appears and a node for the Connectivity Map is added under the Project, on the Project Explorer tree labeled **CMap1**.

Create four additional Connectivity Maps—**CMap2**, **CMap3**. **CMap4**, and **CMap5**— and rename them as follows:

- cmSNACPIC_Inbound
- cmSNACPIC_Outbound
- cmSNAHelper_Inbound
- cmSNAHelper_Outbound

The icons in the toolbar represent the available components used to populate the Connectivity Map canvas.

## Populating the Connectivity Map

Add the Project components to the Connectivity Map by dragging the icons from the toolbar to the canvas.

Each Connectivity Map in the **prjSNA_Sample_JCD** sample Project requires the following components:

- File External Application (2)
- SNA External Application
- Service

Any eWay added to the Connectivity Map is associated with an External System. To establish a connection to SNA, first select SNA as an External System to use in your Connectivity Map.

**Steps required to select a SNA External System:**

1. Click the **External Application** icon on the Connectivity Map toolbar.
2. Select the external systems necessary to create your Project (for this sample, **SNA** and **File**). Icons representing the selected external systems are added to the Connectivity Map toolbar.
3. Rename the following components and then save changes to the Repository:
   - File1 to FileClientIN
   - File2 to FileClientOUT
   - SNA1 to eaSNAOUT
4. Rename each Connectivity Map Service to match the intended operation, as for example:
   - jcdSNACPIC_Inbound
   - jcdSNACPIC_Outbound
   - jcdSNAHelper_Inbound
   - jcdSNAHelper_Outbound

## 4.4.3 Creating the Collaboration Definitions (Java)

The next step is to create Collaboration Definitions (Java) or JCDs using the **Collaboration Definition Wizard (Java)**. Once you create the Collaboration Definitions, you can write the Business Rules of the Collaborations using the Collaboration Editor.

The four Collaborations that you will create as part of this sample Project provide different techniques for using the eWay to perform varying ranges of SNA conversation tasks that you may need to execute. The Collaborations are not meant to demonstrate the only way to perform desired operations. Rather, they should provide insight into how you may use the SNA eWay to develop your applications.

Inbound SNA conversations accept incoming conversation requests from remote transaction programs that initialize conversations. Outbound SNA conversations initialize conversations and relay (send) data to transaction programs that accept the initialize conversation request.

### jcdSNACPIC_Inbound Collaboration

This Collaboration demonstrates how to use the eWay, in conjunction with the CPIC Java methods, to accept an incoming SNA conversation and output the conversation to a file on the local system.

**Steps required to create the jcdSNACPIC_Inbound Collaboration:**

1. From the Project Explorer, right-click the sample Project and select **New** > **Collaboration Definition (Java)** from the shortcut menu. The **Collaboration Definition Wizard (Java)** appears.
2. Enter a Collaboration Definition name (for this sample **jcdSNACPIC_Inbound**) and click **Next**.
3. For Step 2 of the wizard, from the Web Services Interfaces selection window, double-click **Sun SeeBeyond** > **eWays** > **File** > **FileClient** > **receive**. The File Name field now displays **receive.** Click **Next**.
4. For Step 3 of the wizard, from the Select OTDs selection window, double-click **Sun SeeBeyond** > **eWays** > **SNALU62** > **SNALU62eWay**. The **SNALU62eWay** OTD is added to the Selected OTDs field.
5. Click the **Up One Level** button twice to return to the Repository. Double-click **Sun SeeBeyond** > **eWays** > **File** > **FileClient**. The **Selected OTDs** field now lists the **FileClient** OTD.
6. Click **Finish**. The Collaboration Editor with the new **jcdSNACPIC_Inbound** Collaboration appears in the right pane of the Enterprise Designer.

### jcdSNACPIC_Outbound Collaboration

**Steps required to create the jcdSNACPIC_Outbound Collaboration:**

1. From the Project Explorer, right-click the sample Project and select **New** > **Collaboration Definition (Java)** from the shortcut menu. The **Collaboration Definition Wizard (Java)** appears.

2. Enter a Collaboration Definition name (for this sample **jcdSNACPIC_Outbound**) and click **Next**.
3. For Step 2 of the wizard, from the Web Services Interfaces selection window, double-click **Sun SeeBeyond** > **eWays** > **File** > **FileClient** > **receive**. The File Name field now displays **receive.** Click **Next**.
4. For Step 3 of the wizard, from the Select OTDs selection window, double-click **Sun SeeBeyond** > **eWays** > **SNALU62** > **SNALU62eWay**. **SNALU62eWay** is added to the Selected OTDs field.
5. Click the **Up One Level** button twice to return to the Repository. Double-click **Sun SeeBeyond** > **eWays** > **File** > **FileClient**. The **Selected OTDs** field now lists the **FileClient** OTD.
6. Click **Finish**. The Collaboration Editor with the new **jcdSNACPIC_Outbound** Collaboration appears in the right pane of the Enterprise Designer.

## jcdSNAHelper_Inbound Collaboration

**Steps required to create the jcdSNAHelper_Inbound Collaboration:**

1. From the Project Explorer, right-click the sample Project and select **New** > **Collaboration Definition (Java)** from the shortcut menu. The **Collaboration Definition Wizard (Java)** appears.
2. Enter a Collaboration Definition name (for this sample **jcdSNAHelper_Inbound**) and click **Next**.
3. For Step 2 of the wizard, from the Web Services Interfaces selection window, double-click **Sun SeeBeyond** > **eWays** > **File** > **FileClient** > **receive**. The File Name field now displays **receive.** Click **Next**.
4. For Step 3 of the wizard, from the Select OTDs selection window, double-click **Sun SeeBeyond** > **eWays** > **SNALU62** > **SNALU62eWay**. The **SNALU62eWay** OTD is added to the Selected OTDs field.
5. Click the **Up One Level** button twice to return to the Repository. Double-click **Sun SeeBeyond** > **eWays** > **File** > **FileClient**. The **Selected OTDs** field now lists the **FileClient** OTD.
6. Click **Finish**. The Collaboration Editor with the new **jcdSNAHelper_Inbound** Collaboration appears in the right pane of the Enterprise Designer.

## jcdSNAHelper_Outbound Collaboration

**Steps required to create the jcdSNAHelper_Outbound Collaboration:**

1. From the Project Explorer, right-click the sample Project and select **New** > **Collaboration Definition (Java)** from the shortcut menu. The **Collaboration Definition Wizard (Java)** appears.
2. Enter a Collaboration Definition name (for this sample **jcdSNAHelper_Outbound**) and click **Next**.

3. For Step 2 of the wizard, from the Web Services Interfaces selection window, double-click **Sun SeeBeyond** > **eWays** > **File** > **FileClient** > **receive**. The File Name field now displays **receive.** Click **Next**.
4. For Step 3 of the wizard, from the Select OTDs selection window, double-click **Sun SeeBeyond** > **eWays** > **SNALU62** > **SNALU62eWay**. **SNALU62eWay** is added to the Selected OTDs field.
5. Click the **Up One Level** button twice to return to the Repository. Double-click **Sun SeeBeyond** > **eWays** > **File** > **FileClient**. The **Selected OTDs** field now lists the **FileClient** OTD.
6. Click **Finish**. The Collaboration Editor with the new **jcdSNAHelper_Outbound** Collaboration appears in the right pane of the Enterprise Designer.

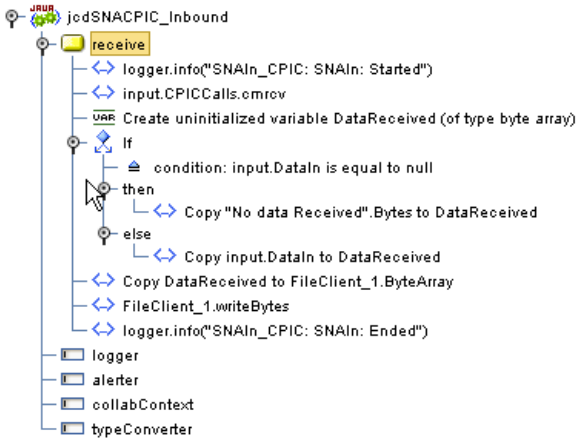
## 4.4.4 Creating the Collaboration Business Rules

The next step in the sample is to create the Business Rules of the Collaboration using the Collaboration Editor.

### Creating the jcdSNACPIC_Inbound Collaboration Business Rules

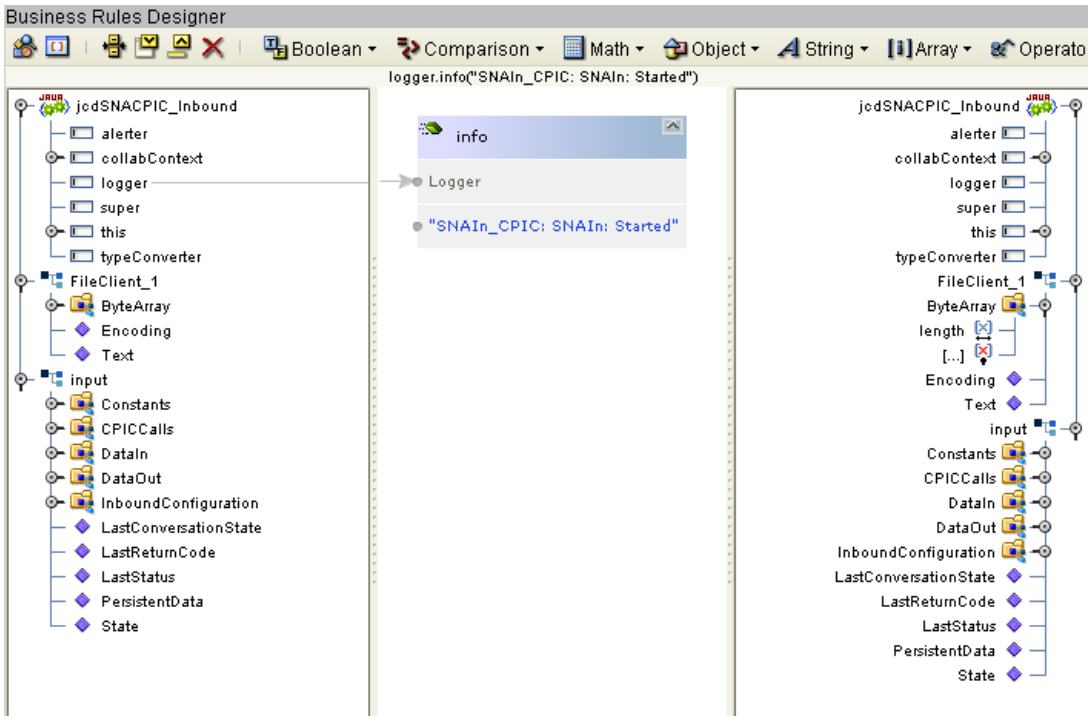
The jcdSNACPIC_Inbound Collaboration contains the Business Rules displayed in Figure 12.

**Figure 12** jcdSNACPIC_Inbound Business Rules



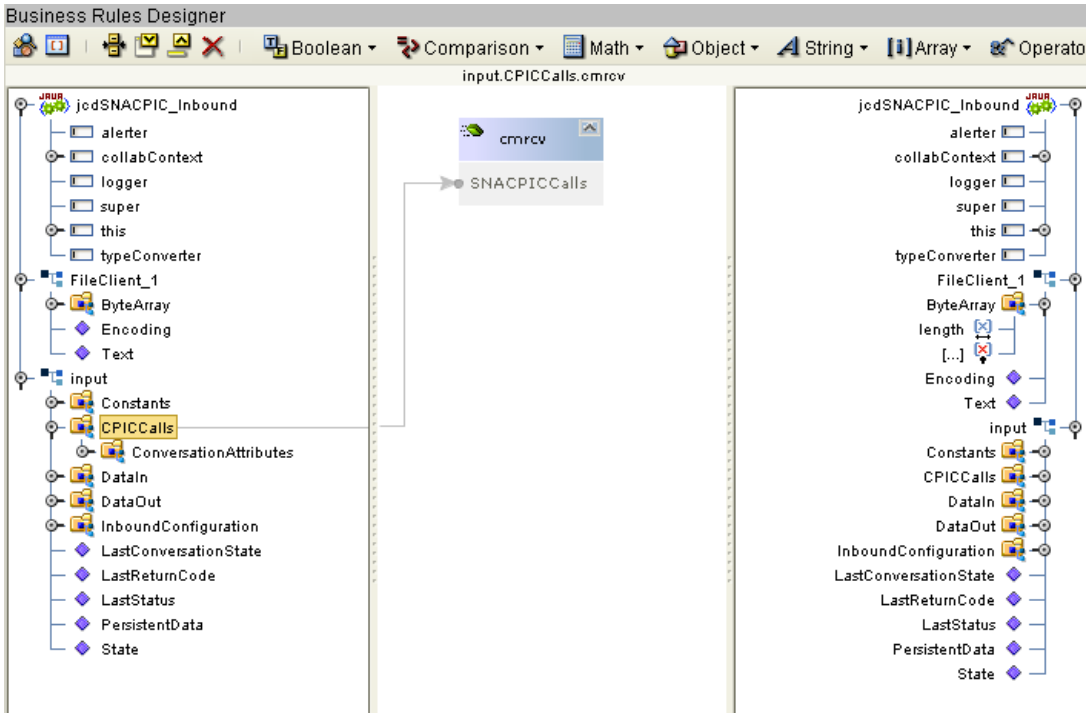

The **logger.info("SNAIn_CPIC: SNAIn: "Started")** Business Rule is displayed in Figure 13.

**Figure 13** jcdSNACPIC_Inbound Business Rule 1



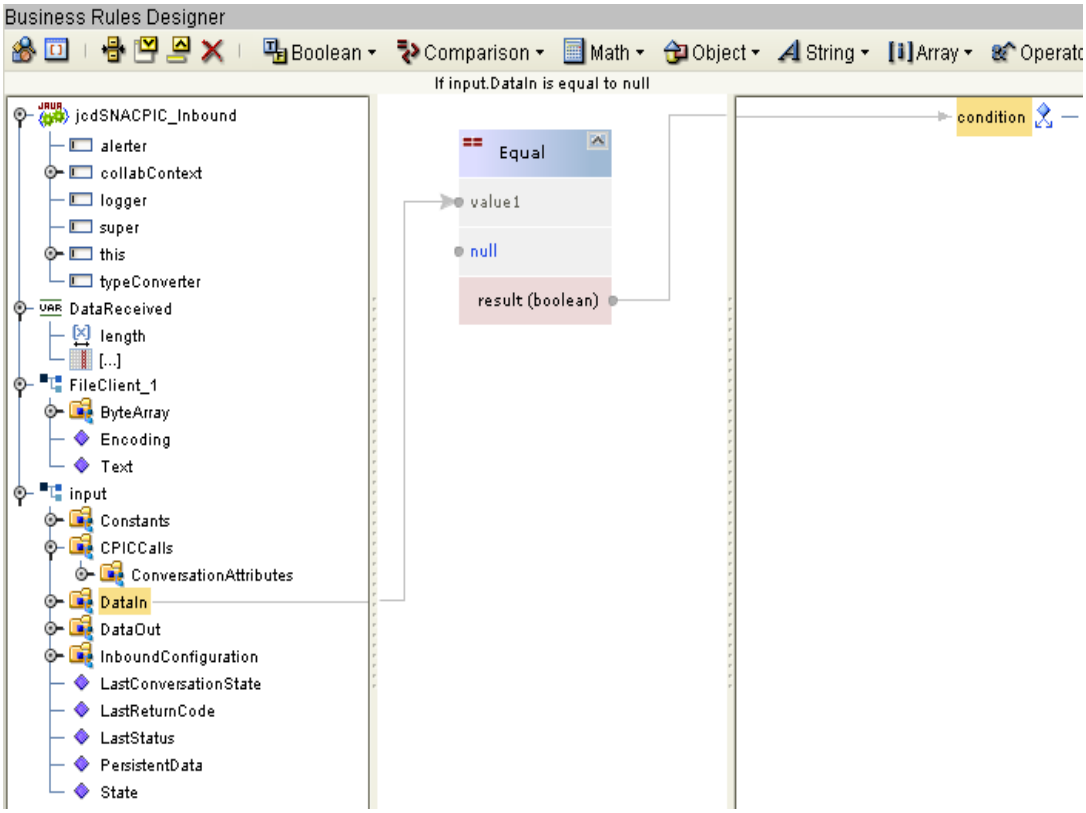

The **input.CPICCalls.cmrcv** Business Rule is displayed in Figure 14.

**Figure 14** jcdSNACPIC_Inbound Business Rule 2



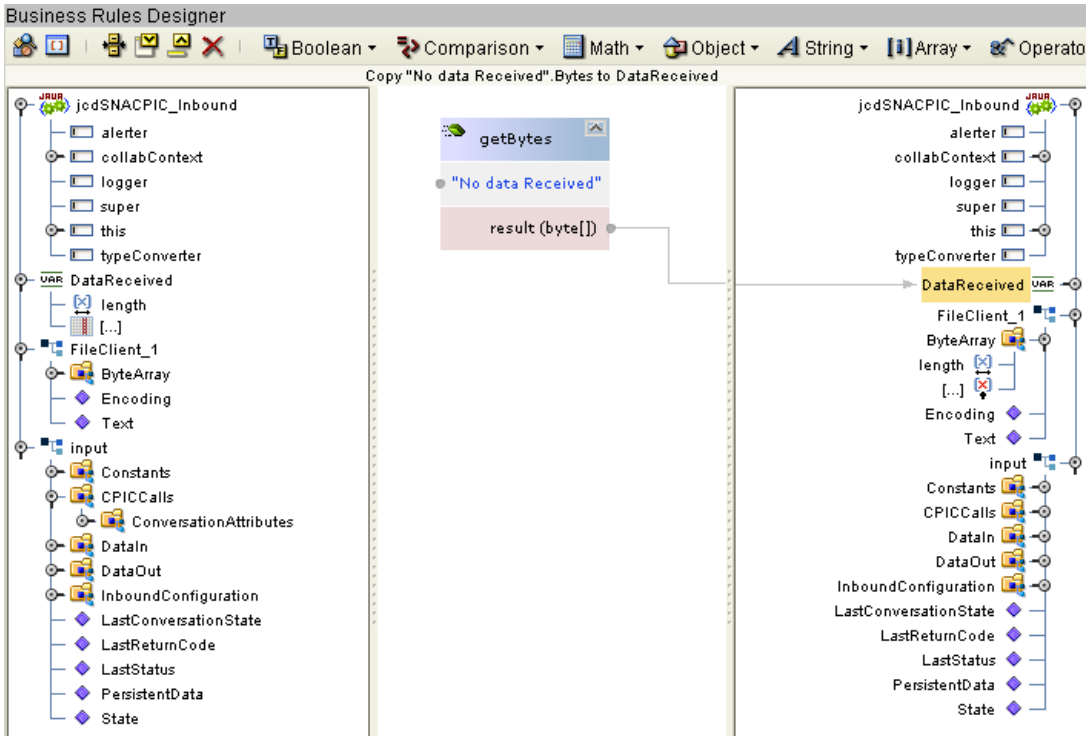

The **input.DataIn is equal to null** condition within the *if* clause is displayed in Figure 15.

**Figure 15** jcdSNACPIC_Inbound Business Rule 3



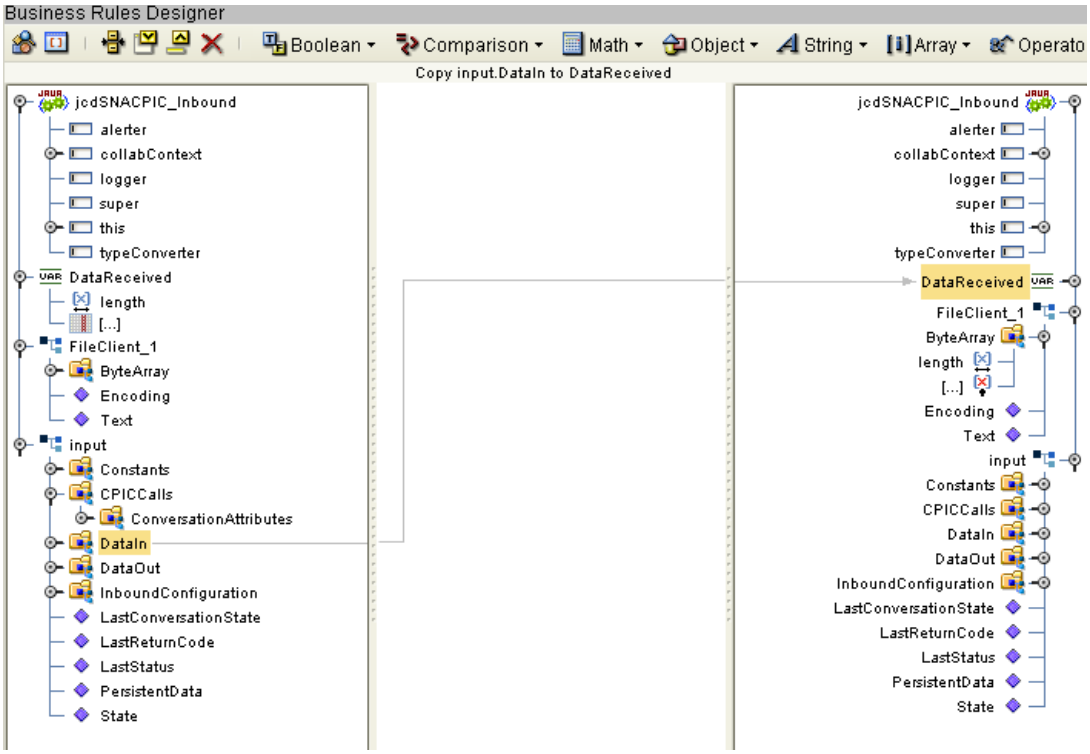

The **Copy "No data Received".Bytes to Data Received** *then* clause within the **input.DataIn is equal to null** condition is displayed in Figure 16.

**Figure 16** jcdSNACPIC_Inbound Business Rule 4



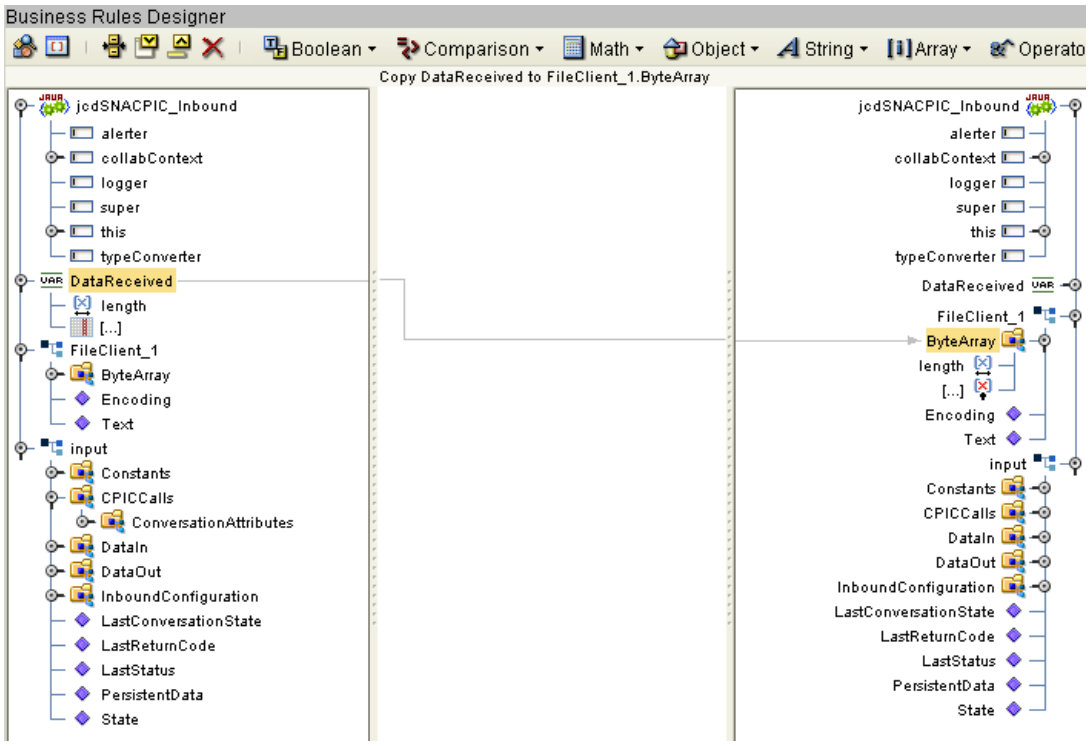

The **Copy input.DataIn to DataReceived** *else* clause within the **input.DataIn is equal to null** condition is displayed in Figure 17.

**Figure 17** jcdSNACPIC_Inbound Business Rule 5



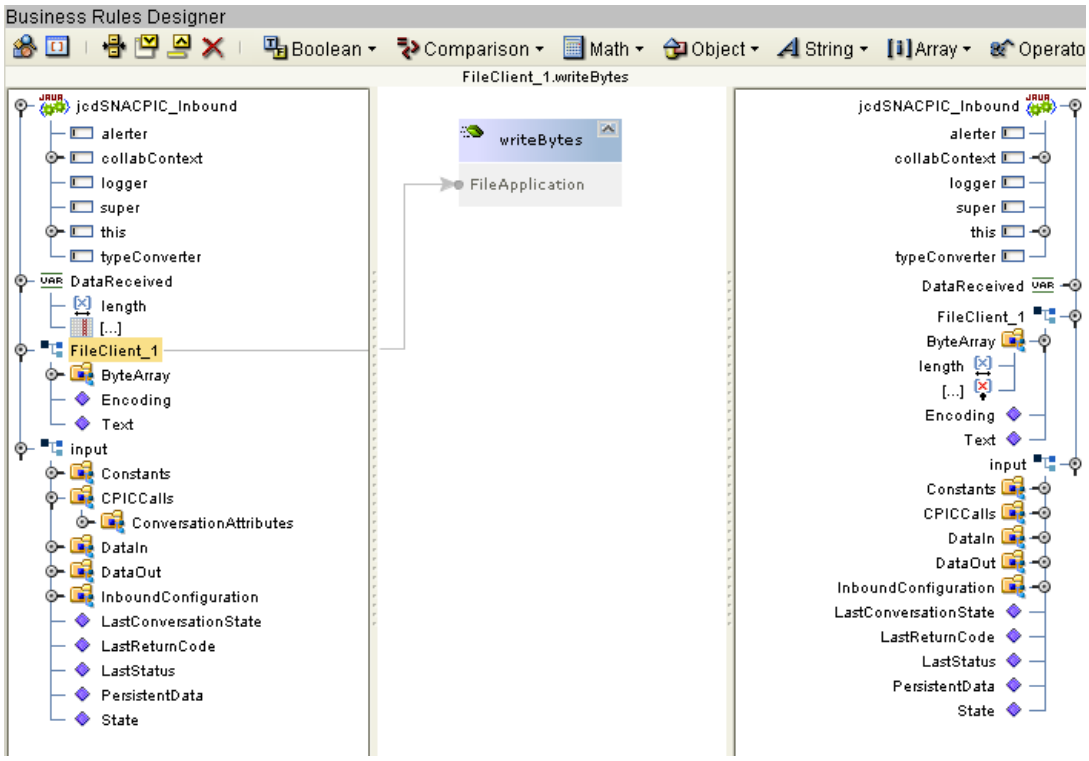
The **Copy DataReceived to FileClient_1.ByteArray** Business Rule is displayed in Figure 18.

**Figure 18** jcdSNACPIC_Inbound Business Rule 6



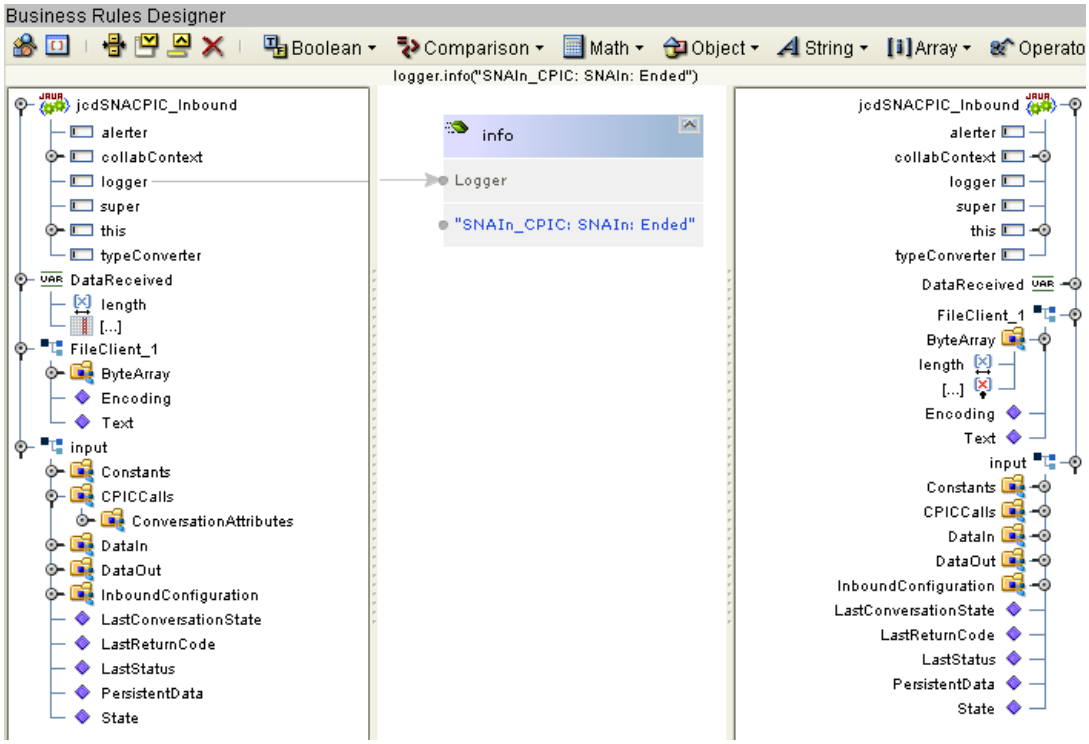

The **FileClient_1.writeBytes** Business Rule is displayed in Figure 19.

**Figure 19** jcdSNACPIC_Inbound Business Rule 7




The **loger.info("SNAIn_CPIC: SNAIn: Ended")** Business Rule is displayed in Figure 20.

**Figure 20** jcdSNACPIC_Inbound Business Rule 8



Sample code from the jcdSNACPIC_Inbound Collaboration includes the following:

```
package prjSNA_Sample_JCD;

public class jcdSNACPIC_Inbound
{

    public com.stc.codegen.logger.Logger logger;
    public com.stc.codegen.alerter.Alerter alerter;
    public com.stc.codegen.util.CollaborationContext collabContext;
    public com.stc.codegen.util.TypeConverter typeConverter;

    public void receive(
com.stc.connector.snalu62.inbound.SNAInboundApplication input,
com.stc.connector.appconn.file.FileApplication FileClient_1 )
        throws Throwable
    {
        logger.info( "SNAIn_CPIC: SNAIn: Started" );
        input.getCPICCalls().cmrcv();
        byte[] DataReceived;
        if (input.getDataIn() == null) {
            DataReceived = "No data Received".getBytes();
        } else {
            DataReceived = input.getDataIn();
        }
        FileClient_1.setByteArray( DataReceived );
        FileClient_1.writeBytes();
        logger.info( "SNAIn_CPIC: SNAIn: Ended" );
    }

}
```

### Anaylyzing the Collaboration Sample Code:

The first six lines of code in this Collaboration are created by the default Collaboration wizard. As such, these line of code will not be discussed here. However, keep in mind that if you are creating a new Java Collaboration, your package name and class name may differ. The next three lines of code tell the Collaboration from where the data should be retrieved and to where the data should be sent.

Since the goal of this Collaboration is to handle incoming conversations, you need to create an instance of the `SNAInboundApplication` interface in order to read incoming conversation traffic. In order for the Collaboration to know to where the data should be sent, you must create an instance of the `FileApplication` interface that belongs to the File eWay. This will allow you to output incoming conversation traffic to a file on the local system. Error handling is handled by the `Throwable` class.

One of the first things you should do before processing any conversation traffic is turn on the logging feature. Here, the logger is instantiated with a specific phrase to include in the log file. Now that limited debugging for the Collaboration is available, the next step is to tell the Collaboration to listen to the incoming traffic. Listening to the conversation traffic is performed by using the CPIC `cmrcv()` method of the exposed Java CPIC calls. Since the SNA CPIC calls belong to `getCPICCalls()`, listening to the input from the looks like this: `input.getCPICCalls().cmrcv()`. In order to process the incoming traffic, a byte array, `recv`, is created.

Depending on the incoming conversation traffic, a logic loop is setup to tell the Collaboration what to do with the incoming byte data. If no data is received (`if (null == input.getDataIn())`), an output message is displayed, `recv = "No data is`

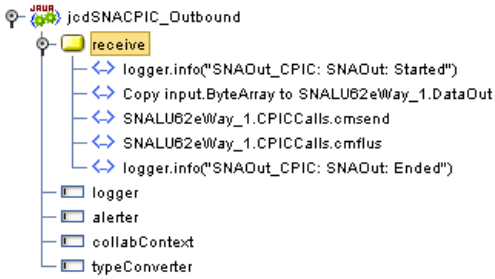
`received.".getBytes()`, before continuing to listen for any change in the data stream. When an incoming data array is finally detected, the loop captures the data, `recv = input.getDataIn()`, and sends it to the `FileApplication` interface for further processing.

Since the goal is to output the collected data array to a file, the File eWay takes the input data, as a byte array (`FileClient_1.setByteArray( recv )`), and writes the data (`FileClient_1.writeBytes()`) to the file specified by the eWay properties. After the data array from the incoming conversation is collected and output to a file, a final log message is displayed, `logger.info( "SNAIn1: Ended." )`.

### Creating the jcdSNACPIC_Outbound Collaboration Business Rules

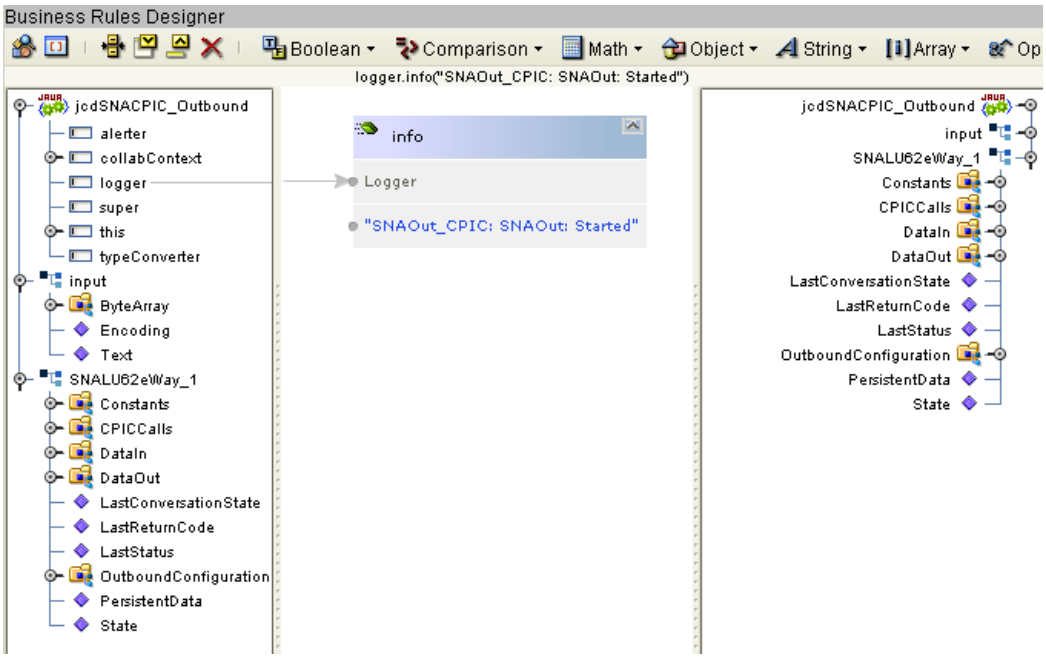
The jcdSNACPIC_Outbound Collaboration contains the Business Rules displayed in Figure 21.

**Figure 21** jcdSNACPIC_Outbound Business Rules



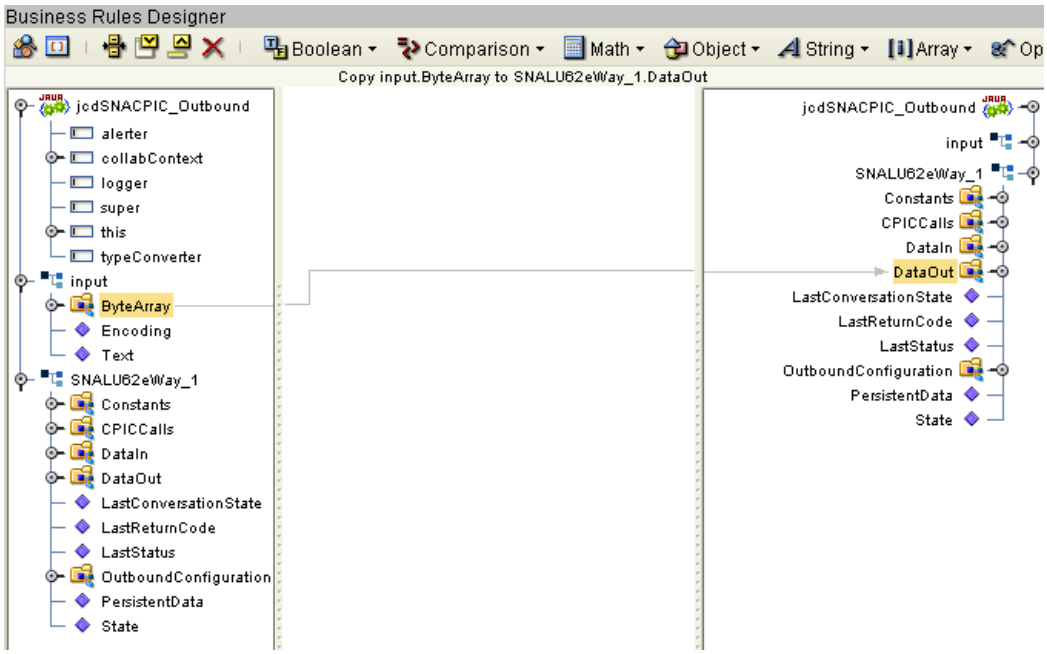

The **logger.info("SNAOut_CPIC: SNAOut: "Started")** Business Rule is displayed in Figure 22.

**Figure 22** jcdSNACPIC_Outbound Business Rule 1



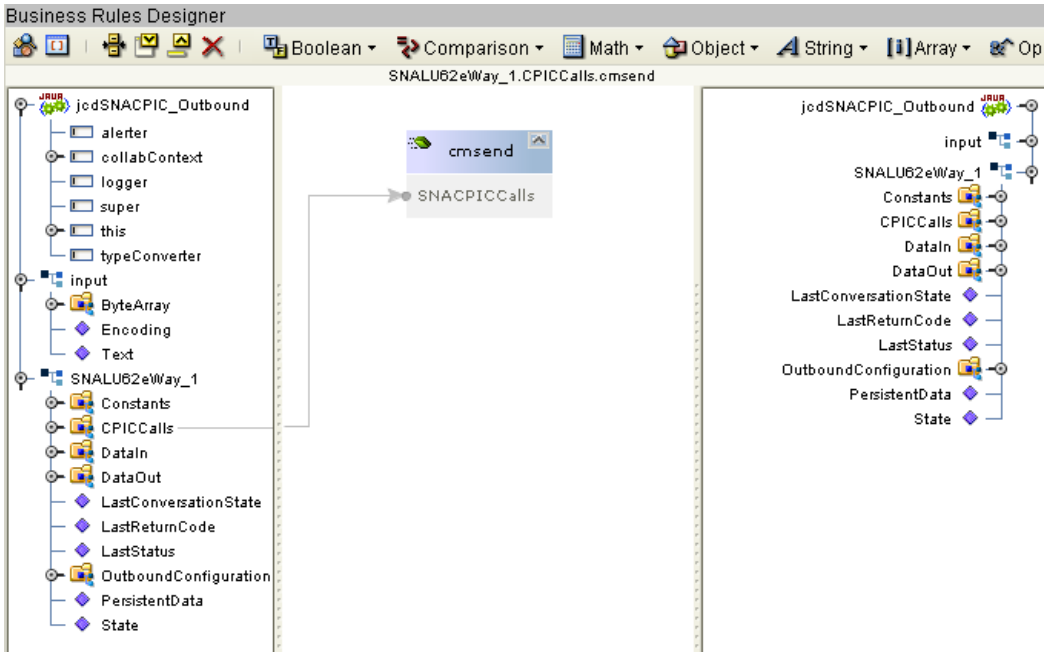
The **Copy input.ByteArray to SNALU62eWay_1.DataOut** Business Rule is displayed in Figure 23.

**Figure 23** jcdSNACPIC_Outbound Business Rule 2



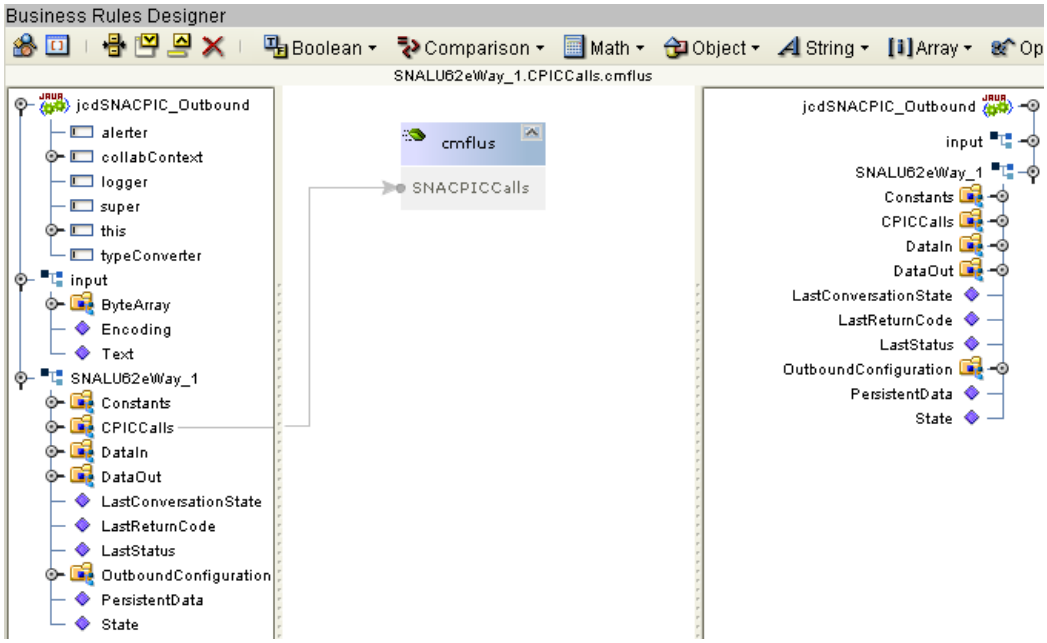

The **SNALU62eWay_1.CPICCalls.cmsend** Business Rule is displayed in Figure 24.

**Figure 24** jcdSNACPIC_Outbound Business Rule 3



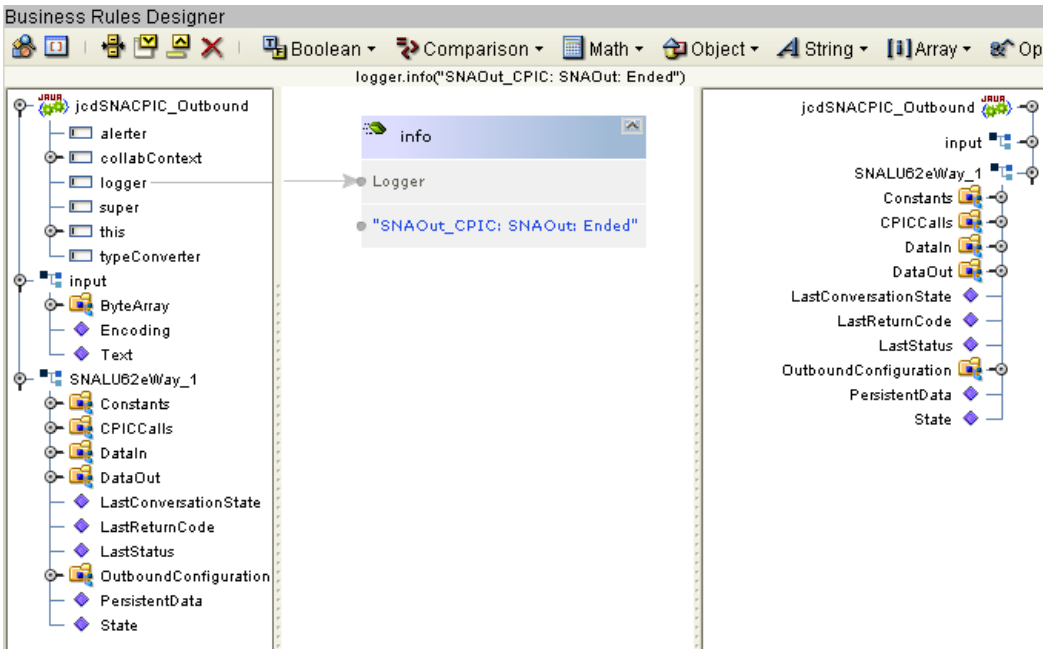

The **SNALU62eWay_1.CPICCalls.cmflus** Business Rule is displayed in Figure 25.

**Figure 25** jcdSNACPIC_Outbound Business Rule 4




The **logger.info("SNAOut_CPIC: SNAOut: "Ended")** Business Rule is displayed in Figure 26.

**Figure 26** jcdSNACPIC_Outbound Business Rule 5




Sample code from the jcdSNACPIC_Outbound Collaboration includes the following:

```
package prjSNA_Sample_JCD;

public class jcdSNACPIC_Outbound
{
```

```
    public com.stc.codegen.logger.Logger logger;
    public com.stc.codegen.alerter.Alerter alerter;
    public com.stc.codegen.util.CollaborationContext collabContext;
    public com.stc.codegen.util.TypeConverter typeConverter;

    public void receive(
com.stc.connector.appconn.file.FileTextMessage input,
com.stc.connector.snalu62.outbound.SNAOutboundApplication
SNALU62eWay_1 )
        throws Throwable
    {
        logger.info( "SNAOut_CPIC: SNAOut: Started" );
        SNALU62eWay_1.setDataOut( input.getByteArray() );
        SNALU62eWay_1.getCPICCalls().cmsend();
        SNALU62eWay_1.getCPICCalls().cmflus();
        logger.info( "SNAOut_CPIC: SNAOut: Ended" );
    }

}
```

### Anaylyzing the Collaboration Sample Code:

The first six lines of code in this Collaboration are created by the default Collaboration wizard. As such, these lines of code will not be discussed here. However, keep in mind that if you are creating a new Java Collaboration, your package name and class name may differ. The next three lines of code tell the Collaboration from where to get the data and to where the data should be sent.

Since the goal of this Collaboration is to handle incoming conversations, you need to create an instance of the `SNAInboundApplication` interface in order to read incoming conversation traffic. In order for the Collaboration to know to where the data should be sent, you must create an instance of the `FileApplication` interface that belongs to the File eWay. This will allow you to output incoming conversation traffic to a file on the local system. Error handling is handled by the `Throwable` class.

One of the first things you should do before processing any conversation traffic is turn on the logging feature. Here, the logger is instantiated with a specific phrase to include in the log file. Now that limited debugging for the Collaboration is available, the next step is to tell the Collaboration to listen to the incoming traffic. Listening to the conversation traffic is performed by using the Helper `recv()` method (refer to the SNA eWay Javadoc). In order to process the incoming traffic, a byte array, `recv`, is created.

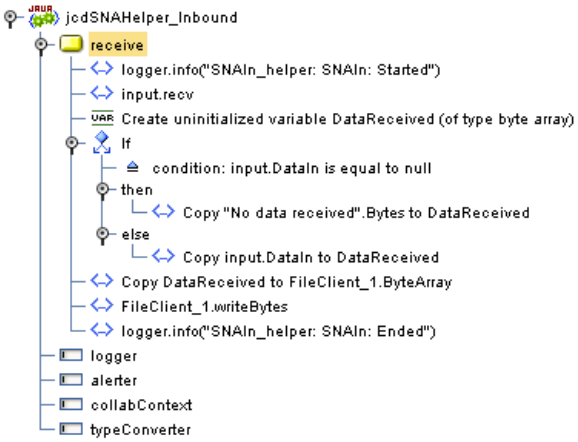
Depending on the incoming conversation traffic, a logic loop is setup to tell the Collaboration what to do with the incoming byte data. If no data is received (`if (null == input.getDataIn())`), an output message is displayed, `recv = "No data is received.".getBytes()`, before continuing to listen for any change in the data stream. When an incoming data array is finally detected, the loop captures the data, `recv = input.getDataIn()`, and sends it to the `FileApplication` interface for further processing.

Since the goal is to output the collected data array to a file, the File eWay takes the input data, as a byte array (`FileClient_1.setByteArray( recv )`), and writes the data (`FileClient_1.writeBytes()`) to the file specified by the eWay properties. After the data array from the incoming conversation is collected and output to a file, a final log message is displayed, `logger.info( "SNAIn1: Ended." )`.

### Creating the jcdSNAHelper_Inbound Collaboration Business Rules

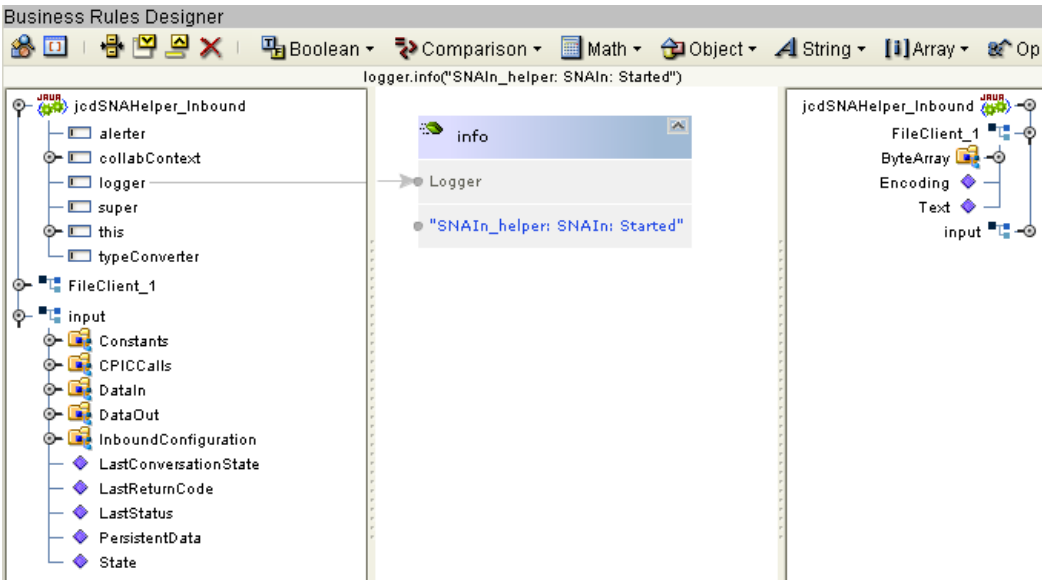
The jcdSNAHelper_Inbound Collaboration contains the Business Rules displayed in Figure 27.

**Figure 27** jcdSNAHelper_Inbound Business Rules



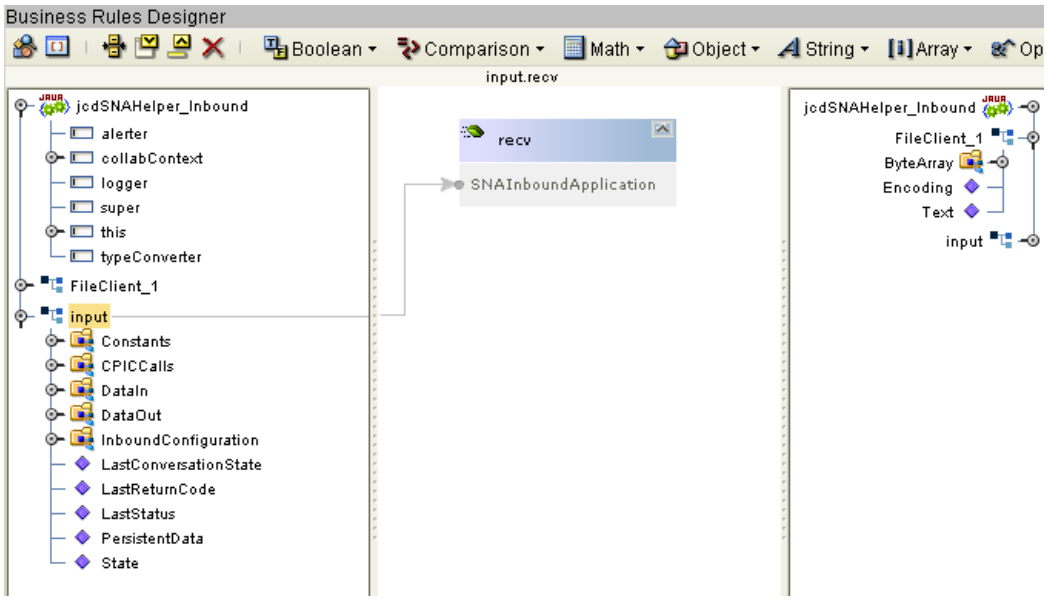

The **logger.info("SNAIn_helper: SNAIn: "Started")** Business Rule is displayed in Figure 28.

**Figure 28** jcdSNAHelper_Inbound Business Rule 1



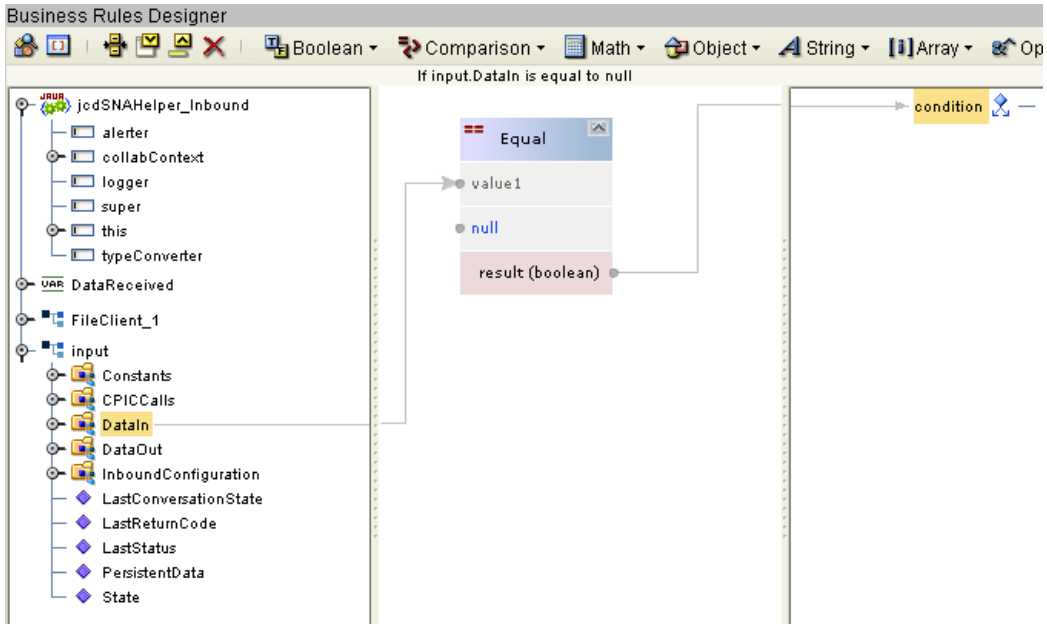

The **input.recv** Business Rule is displayed in Figure 29.

**Figure 29** jcdSNAHelper_Inbound Business Rule 2



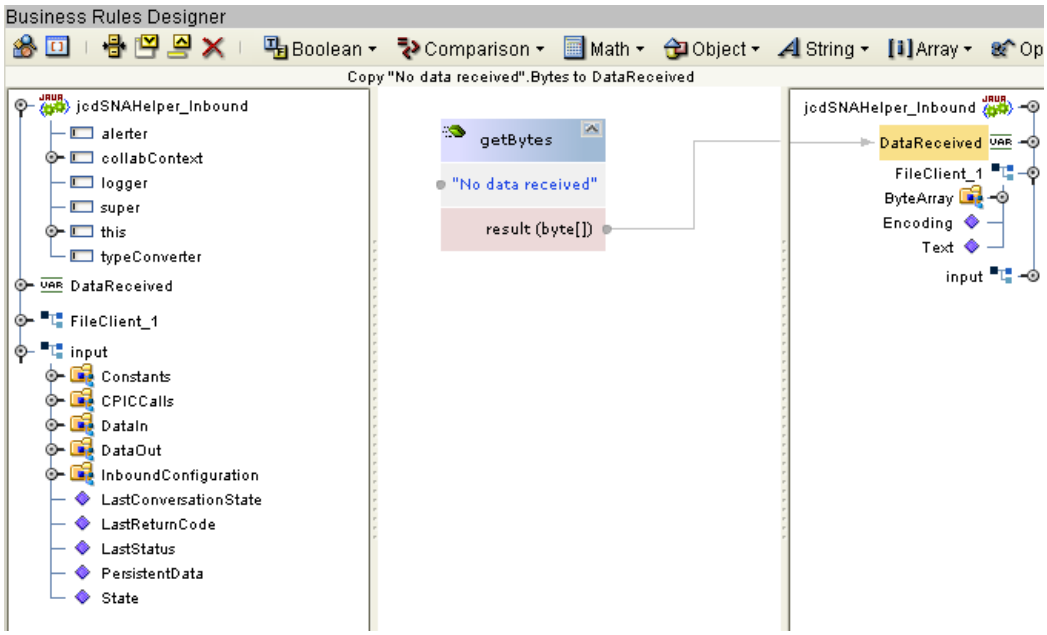

The **input.DataIn is equal to null** condition within the *if* clause is displayed in Figure 15.

**Figure 30** jcdSNAHelper_Inbound Business Rule 3



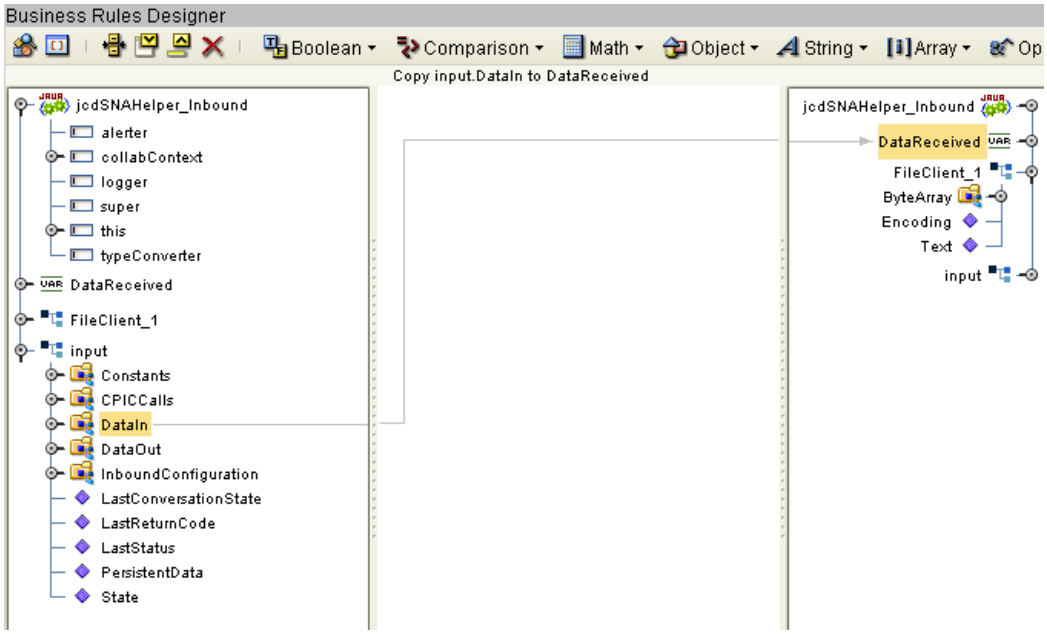

The **Copy "No data Received".Bytes to Data Received** *then* clause within the **input.DataIn is equal to null** condition is displayed in Figure 31.

**Figure 31** jcdSNAHelper_Inbound Business Rule 4



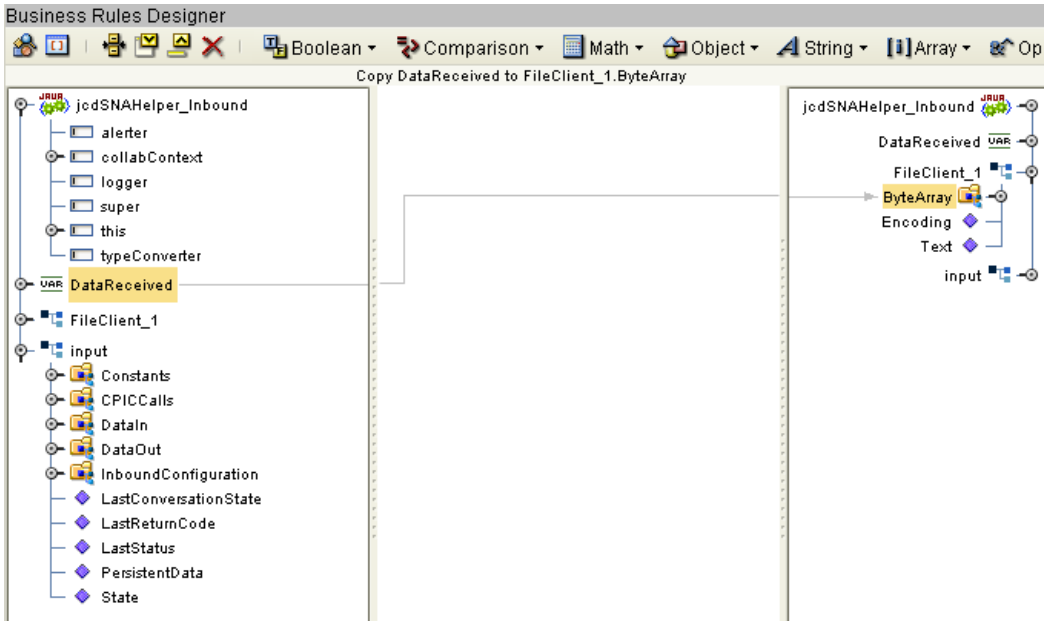

The **Copy input.DataIn to DataReceived** *else* clause within the **input.DataIn is equal to null** condition is displayed in Figure 32.

**Figure 32** jcdSNAHelper_Inbound Business Rule 5



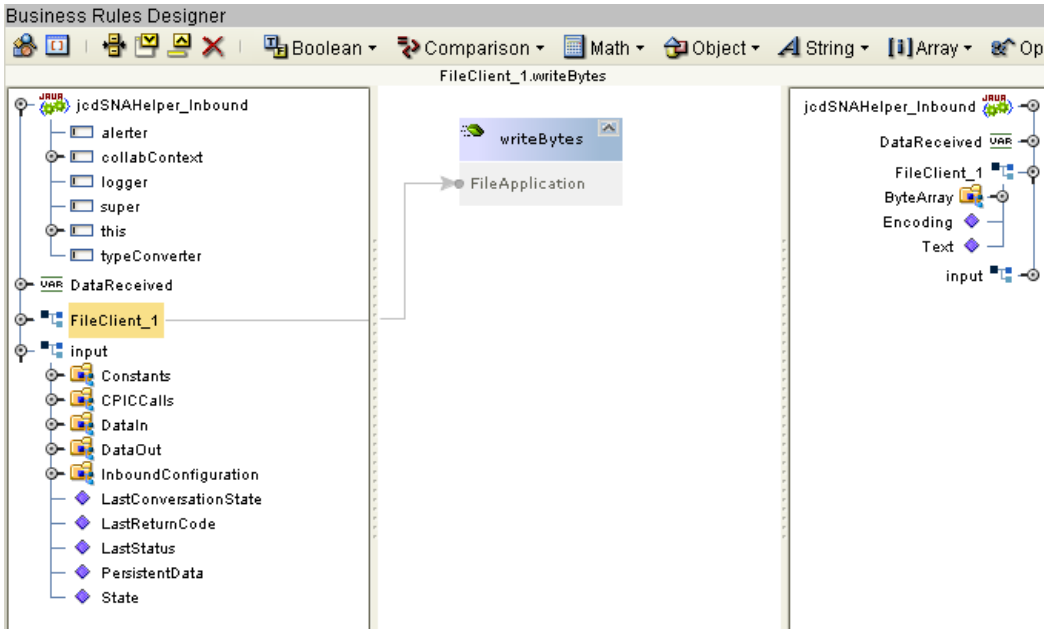

The **Copy DataReceived to FileClient_1.ByteArray** Business Rule is displayed in Figure 33.

**Figure 33** jcdSNAHelper_Inbound Business Rule 6



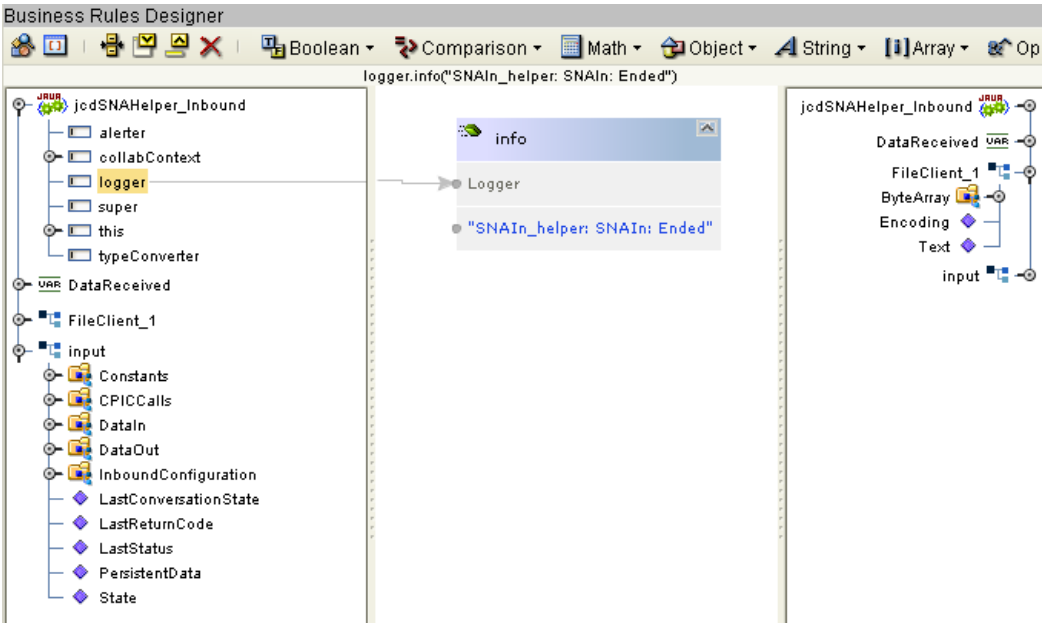

The **FileClient_1.writeBytes** Business Rule is displayed in Figure 34.

**Figure 34** jcdSNAHelper_Inbound Business Rule 7




The **loger.info("SNAIn_helper: SNAIn: Ended")**Business Rule is displayed in Figure 35.

**Figure 35** jcdSNAHelper_Inbound Business Rule 8




Sample code from the jcdSNAHelper_Inbound Collaboration includes the following:

```
package prjSNA_Sample_JCD;

public class jcdSNAHelper_Inbound
{

    public com.stc.codegen.logger.Logger logger;
    public com.stc.codegen.alerter.Alerter alerter;
    public com.stc.codegen.util.CollaborationContext collabContext;
    public com.stc.codegen.util.TypeConverter typeConverter;

    public void receive(
com.stc.connector.snalu62.inbound.SNAInboundApplication input,
com.stc.connector.appconn.file.FileApplication FileClient_1 )
        throws Throwable
    {
        logger.info( "SNAIn_helper: SNAIn: Started" );
        input.recv();
        byte[] DataReceived;
        if (input.getDataIn() == null) {
            DataReceived = "No data received".getBytes();
        } else {
            DataReceived = input.getDataIn();
        }
        FileClient_1.setByteArray( DataReceived );
        FileClient_1.writeBytes();
        logger.info( "SNAIn_helper: SNAIn: Ended" );
    }

}
```

### Anaylyzing the Collaboration Sample Code:

The first six lines of code in this Collaboration are created by the default Collaboration wizard. As such, these lines of code will not be discussed here. However, keep in mind that if you are creating a new Java Collaboration, your package name and class name

may differ. The next three lines of code tell the Collaboration from where the data should be retrieved and to where the data should be sent.

Since the goal of this Collaboration is to retrieve data from a file and send the data, you need to create an instance of the `SNAOutboundApplication` interface in order to deliver conversation traffic. In order for the Collaboration to know from where to retrieve the data that is to be sent via the `SNAOutboundApplication` interface, you must create an instance of the `FileApplication` interface that belongs to the File eWay. This will allow you to read the data file on the local system. Error handling is handled by the `Throwable` class.

One of the first things you should do before processing any conversation traffic is turn on the logging feature. Here, the logger is instantiated with a specific phrase to include in the log file. Now that limited debugging for the Collaboration is available, the next step is to tell the Collaboration from where to retrieve the data that will be transmitted. Using the getByteArray method available for the File eWay, the data can be read with this: `input.getByteArray()`.

A moderately simple construct of the `SNAOutboundApplication` interface can use this getByteArray method to load the data into the send buffer before it is transmitted. In order to load the data to the outbound buffer, you can use something similar to this: `SNALU62eWay_1.setDataOut( input.getByteArray())`. This sets the content of the outgoing payload buffer to the data found in the file.

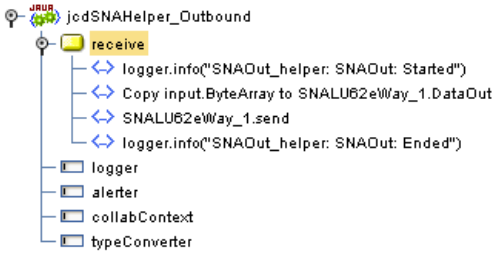
After the payload buffer is filled with the contents of the conversation message to be delivered, you can then send the message. Sending a message to can be performed by using the CPIC `cmsend()` method of the exposed Java CPIC calls. Since the SNA CPIC calls belong to `getCPICCalls()`, listening to the input looks like this: `input.getCPICCalls().cmsend()`. After you initiate the send message activity, you must to flush the contents of the payload buffer to ensure that all the data was sent and to clean the buffer so that it can be used again, if the need arises. To flush the payload buffer, you can use: `SNALU62eWay_1.getCPICCalls().cmflus()`.

After the conversation traffic has been sent, a final log message is displayed, `logger.info( "SNAIn1: Ended." )`.

### Creating the jcdSNAHelper_Outbound Collaboration Business Rules

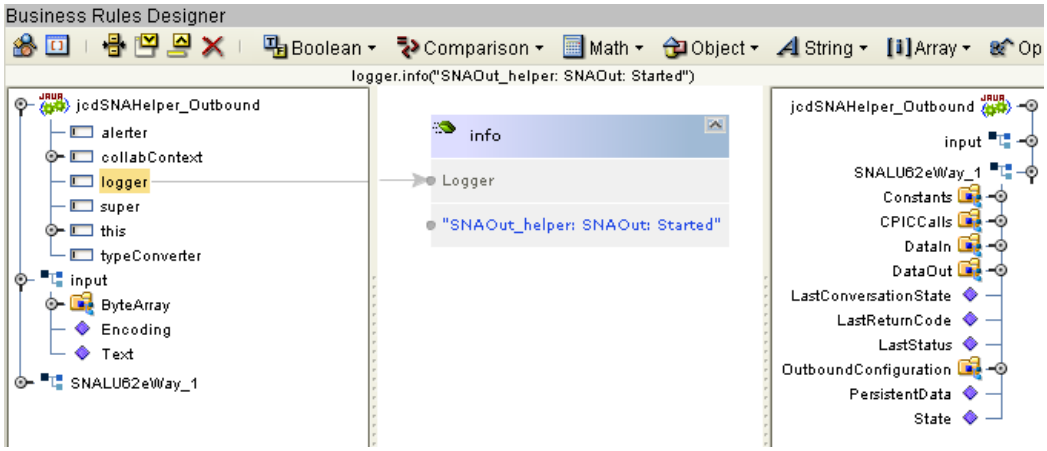
The jcdSNAHelper_Outbound Collaboration contains the Business Rules displayed in Figure 36.

**Figure 36** jcdSNAHelper_Outbound Business Rules



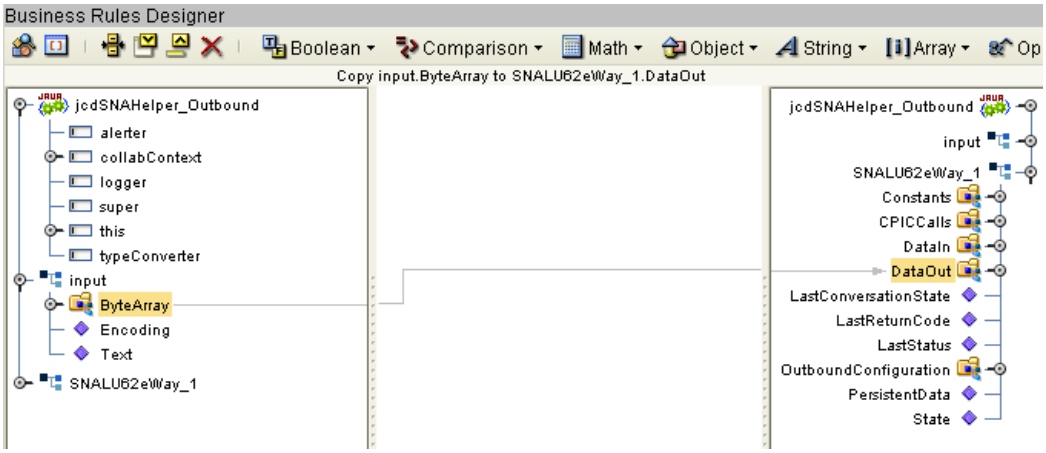

The **logger.info("SNAOut_helper: SNAOut: "Started")** Business Rule is displayed in Figure 37.

**Figure 37** jcdSNAHelper_Outbound Business Rule 1



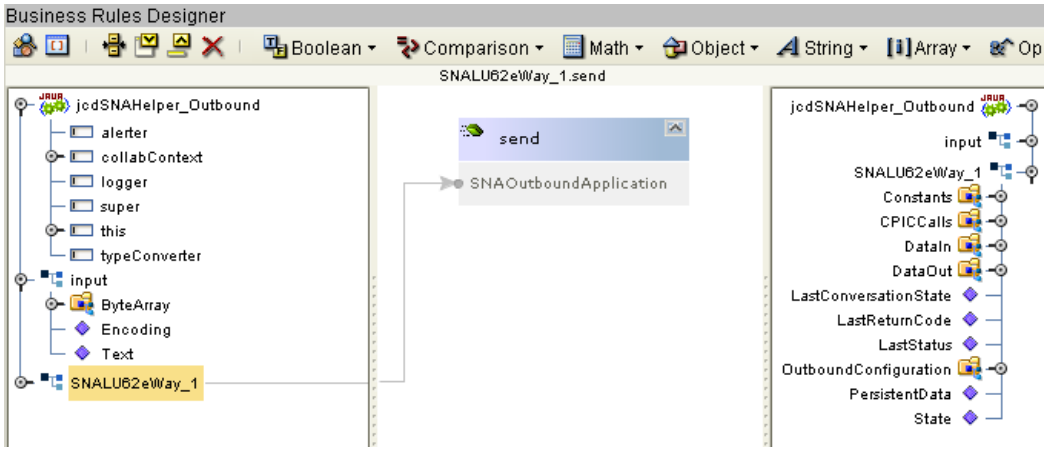

The **Copy input.ByteArray to SNALU62eWay_1.DataOut** Business Rule is displayed in Figure 38.

**Figure 38** jcdSNAHelper_Outbound Business Rule 2



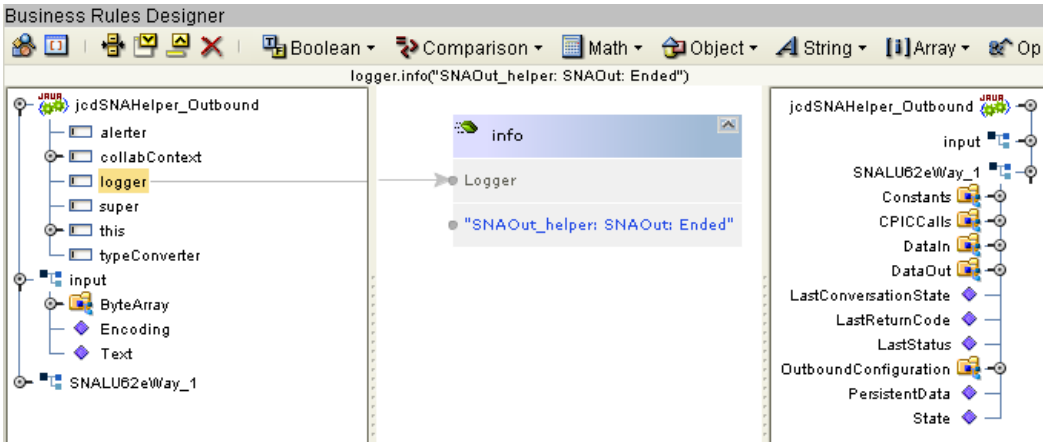

The **SNALU62eWay_1.send** Business Rule is displayed in Figure 39.

**Figure 39** jcdSNAHelper_Outbound Business Rule 3




The **logger.info("SNAOut_helper: SNAOut: "Ended")** Business Rule is displayed in Figure 40.

**Figure 40** jcdSNAHelper_Outbound Business Rule 4




Sample code from the jcdSNAHelper_Outbound Collaboration includes the following:

```
package prjSNA_Sample_JCD;


public class jcdSNAHelper_Outbound
{

    public com.stc.codegen.logger.Logger logger;
    public com.stc.codegen.alerter.Alerter alerter;
    public com.stc.codegen.util.CollaborationContext collabContext;
    public com.stc.codegen.util.TypeConverter typeConverter;

    public void receive(
com.stc.connector.appconn.file.FileTextMessage input,
com.stc.connector.snalu62.outbound.SNAOutboundApplication
SNALU62eWay_1 )
        throws Throwable
    {
        logger.info( "SNAOut_helper: SNAOut: Started" );
        SNALU62eWay_1.setDataOut( input.getByteArray() );
```

```
            SNALU62eWay_1.send();
            logger.info( "SNAOut_helper: SNAOut: Ended" );
        }

    }
```

**Anaylyzing the Collaboration Sample Code:**

The first six lines of code in this Collaboration are created by the default Collaboration wizard. As such, these lines of code will not be discussed here. However, keep in mind that if you are creating a new Java Collaboration, your package name and class name may differ. The next three lines of code tell the Collaboration from where the data should be retrieved and to where the data should be sent.

Since the goal of this Collaboration is to retrieve data from a file and send the data, you need to create an instance of the `SNAOutboundApplication` interface in order to deliver conversation traffic. In order for the Collaboration to know from where to retrieve the data that is to be sent via the `SNAOutboundApplication` interface, you must create an instance of the `FileApplication` interface that belongs to the File eWay. This will allow you to read the data file on the local system. Error handling is handled by the `Throwable` class.

One of the first things you should do before processing any conversation traffic is turn on the logging feature. Here, the logger is instantiated with a specific phrase to include in the log file. Now that limited debugging for the Collaboration is available, the next step is to tell the Collaboration to where the data that will be transmitted should be sent. Using the getByteArray method available for the File eWay, the data can be read with this: `input.getByteArray()`.

A moderately simple construct of the `SNAOutboundApplication` interface can use this `getByteArray` method to load the data into the send buffer before being transmitted. In order to load the data to the outbound buffer, you can use something similar to this: `SNALU62eWay_1.setDataOut( input.getByteArray())`. This sets the content of the outgoing payload buffer to the data found in the file.

After the payload buffer is filled with the contents of the conversation message to be delivered, you can then send the message. Sending a message can be performed by using the Helper `send()` method of the exposed Java methods (refer to the SNA eWay Javadoc). The `send()` method sends the outgoing payload (buffer) represented in the OTD as node DataOut into the local LU's send buffer for transmission to the partner TP with confirmation flag. When the flag is true, a CPIC `cmcfm` will be called after the data is sent. For the general logic flow of this helper function, you may need to refer to the native interface
`com.stc.connector.snalu62.jni.SNAInterface#send(boolean)`.

After the conversation traffic has been sent, a final log message is displayed, `logger.info( "SNAIn1: Ended." )`.

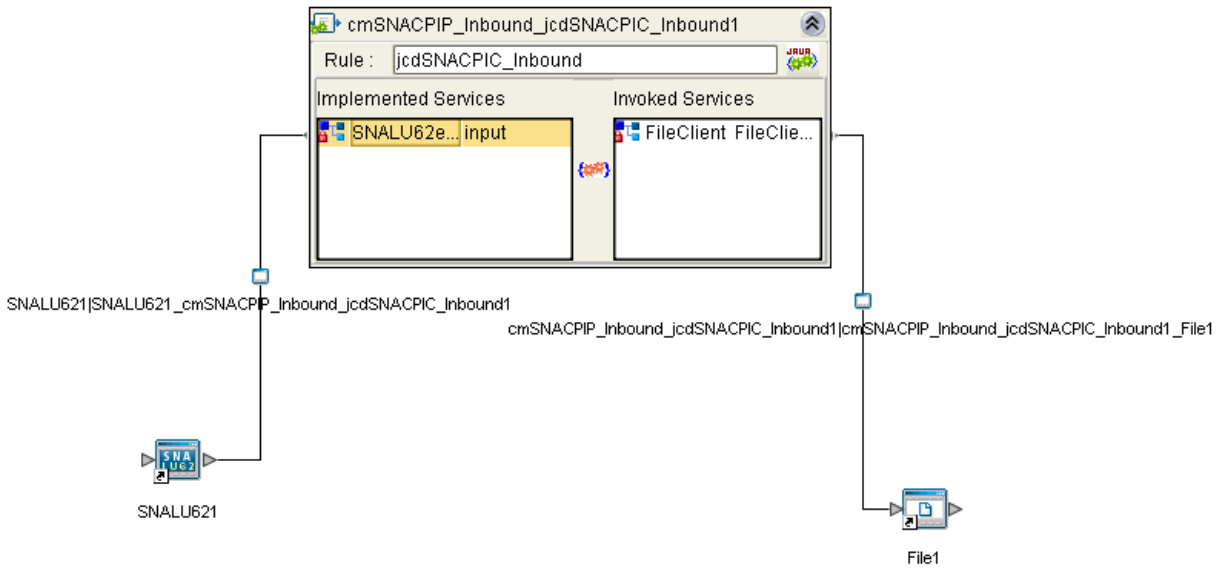
## 4.4.5 Binding the eWay Components

The final step in creating a Connectivity Map is binding the eWay components together.

**Steps required to bind eWay components together:**

1. Double-click a Connectivity Map—in this example **cmSNACPIC_Inbound**—in the Project Explorer tree. The **cmSNACPIC_Inbound** Connectivity Map appears in the Enterprise Designers canvas.
2. Drag and drop the **jcdSNACPIC_Inbound** Collaboration from the Project Explorer to the **jcdSNACPIC_Inbound** Service. The Service icon "gears" change from red to green.
3. Double-click the **jcdSNACPIC_Inbound** Service. The **jcdSNACPIC_Inbound** Binding dialog box appears.
4. Map the input **SNALU62eWay** (under Implemented Services) to the **SNALU62** SNA External Application. To do this, click on **SNALU62eWAy** in the **jcdSNACPIC_Inbound** Binding dialog box, and drag the cursor to the **SNALU62** External Application in the Connectivity Map. A link is now visible between **SNALU62** and **jcdSNACPIC_Inbound**.
5. From the **jcdSNACPIC_Inbound** Binding dialog box, map **FileClient_1** to the **FileClientOUT** External Application, as seen in Figure 41.

**Figure 41** Connectivity Map - Associating (Binding) the Project's Components




6. Minimize the **jcdSNACPIC_Inbound** Binding dialog box by clicking the chevrons in the upper-right corner.
7. Save your current changes to the Repository, and then repeat this process for each of the other Connectivity Maps.

## 4.4.6 Creating an Environment

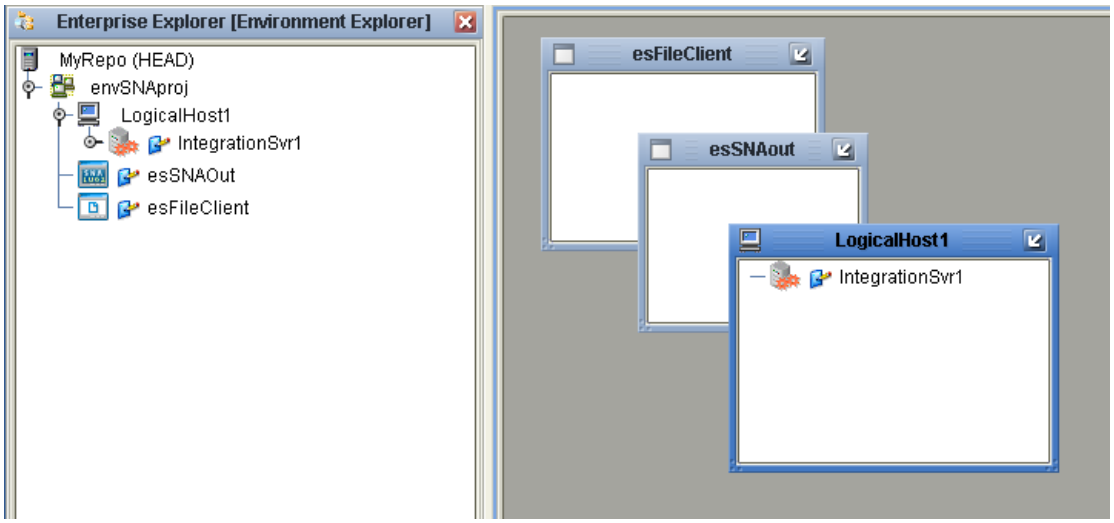
Environments include the external systems, Logical Hosts, Integration Servers and message servers used by a Project and contain the configuration information for these

components. Environments are created using the Enterprise Designer's Environment Editor.

**Steps required to create an Environment:**

1. From the Enterprise Designer's Enterprise Explorer, click the **Environment Explorer** tab.
2. Right-click the Repository and select **New Environment**. A new Environment is added to the Environment Explorer tree.
3. Rename the new Environment to **envSNAProj**.
4. Right-click **envSNAProj** and select **New > SNALU62 External System**. Name the External System **esSNAOut.** Click **OK**. **esSNAOut** is added to the Environment Editor.
5. Right-click **envSNAProj** and select **New > File External System**. Name the External System **esFileClient**. Click **OK**. **esFileClient** is added to the Environment Editor.
6. Right-click **envSNAProj** and select **New > Logical Host**. The **LogicalHost1** box is added to the Environment and **LogicalHost1** is added to the Environment Editor tree.
7. Right-click **LogicalHost1** and select **New > Sun SeeBeyond Integration Server**. A new Integration Server (**IntegrationSvr1**) is added to the Environment Explorer tree under **LogicalHost1**. See Figure 42.

**Figure 42** Environment Editor - envSNAProj



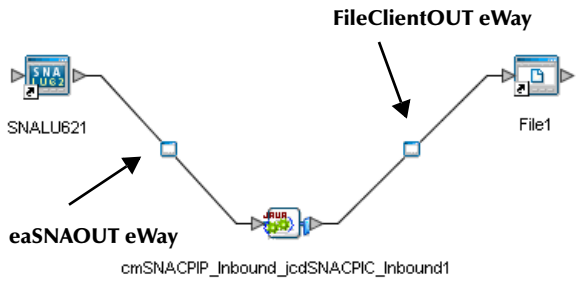

8. Save your current changes to the Repository.

## 4.4.7 Configuring the eWays

eWays facilitate communication and movement of data between the external applications and the eGate system. Each Connectivity Map in the **prjSNA_Sample_JCD** sample Project uses two eWays that are represented as nodes between the External Applications and the Business Process. See Figure 43.

You must configure eWay properties in both the Connectivity Map and the Environment Explorer.

**Figure 43** eWays in the cmSNACPIC_Inbound Connectivity Map




## Configuring the eWay Properties

**Steps required to configure the Inbound SNA eWay properties:**

1. For the **cmSNACPIC_Outbound** and **cmSNAHelper_Outbound** Connectivity Maps, double-click the **FileIn** eWay and modify the following property for your system:

   Input File Name: **SNACPIC_input*.txt** or **SNAHelper_input*.txt**

2. For the **cmSNACPIC_Inbound** and **cmSNAHelper_Inbound** Connectivity Maps, double-click the **FileOut** eWay and modify the following property for your system:

   Output File Name: **SNACPIC_output%d*.dat** or **SNAHelper_output%d*.dat**

**Steps required to configure the Environment Explorer properties:**

1. From the **Environment Explorer** tree, right-click the File External System (**esFileClient** in this sample), and select **Properties**. The Properties Editor opens to the File eWay Environment configuration.
2. Modify the Parameter Settings as required for your Environment, and click **OK**.

## 4.4.8 Configuring the Logical Host

Before you can execute any projects created with the SNA eWay, you must add the SNA eWay Runtime JNI to the Logical Host.

1. If the Logical Host is not already installed, download and install the Logical Host as described in the *Sun Java Composite Application Platform Suite Installation Guide*.
2. From the Environment Explorer, right-click the targeted environment and click **New Logical Host**.
3. Launch the Enterprise Manager.
4. From the Enterprise Manager, click the **DOWNLOADS** tab.
5. Download and save the eWay Runtime JNI file to your local system.

A You are required to copy the the Runtime JNI file to one of the following two folders:

```
<JavaCAPS51>\logicalhost\is\lib\
```

or

```
<JavaCAPS51>\logicalhost\is\domains\domain1\lib
```

where <*JavaCAPS51*> is the directory where the Sun Java Composite Application Platform Suite is installed.

**Note:** *If the Logical Host is running, it must be restarted to pickup the new JNI file.*

### SPARC64 logical host deployment

The SNA eWay is shipped with the Sparc64 Logical Host. The Java Virtual Machine (JVM) for the Sparc64 Logical Host can be started in either 32-bit or 64-bit mode. By default, the JVM is started in 32-bit mode. To properly deploy the logical host for Sparc 64, you must ensure that the JVM bit mode matches the bit size of the JNI bridge shared library for the appropriate Brixton or SNAP-IX library that you installed in **"Installing the SNA eWay on an eGate supported system" on page 16**.

If planning to host large EJB applications, the big EJB applications run more efficiently on Solaris 9 with the JVM configured for 64-bit execution. To modify the logical host so that the JVM starts in 64-bit mode, instead of the default 32-bit mode, you must first perform all the procedures as specified in this section above before continuing.

**To set the JVM to start in 64-bit mode:**

1. Navigate to the Integration Server Administration Console.
2. Click the **User Management** tab.
3. Click the **JVM Settings** tab.
4. Click the **JVM Options** link.
5. Under the **Options** section, edit the appropriate settings to start the JVM in 64-bit mode.
6. When you have completed any necessary changes, click the **Save** button.

## 4.4.9 Configuring for Logical Host Platforms

Specific logical host bootstrap procedures must be adhered to for this eWay. Modifications to the bootstrap procedures are described below and should be implemented according to the Logical Host platform:


- **Windows 2000/XP/Windows Server 2003** on page 73
- **Sparc (32-bit)** on page 73
- **Sparc (64-bit)** on page 74


Refer to the*Sun Java Composite Application Platform Suite Installation Guide* for general instructions on configuring the Logical Host.

## Windows 2000/XP/Windows Server 2003

1. Ensure the `stc_jnisna.dll` resides in a directory specified in the system PATH statement. Refer to **"Installing the SNA eWay on an eGate supported system" on page 16** for instructions on how to download and configure the `stc_jnisna.dll` file.
2. Run the **bootstrap.bat** file.

## IBM AIX 5L versions 5.2 and 5.3 (32-bit)

1. Ensure the `libstc_jnisna.so` file resides in a directory specified in the system LIBPATH statement. Refer to **"Installing the SNA eWay on an eGate supported system" on page 16** for instructions on how to download and configure the `libstc_jnisna.so` file.
2. Add the IBM Communication Server directory path (`usr/lib/sna`) to the system LIBPATH.
3. From a command prompt, execute the following:

```
EXPORT OBJECT_MODE=32

EXPORT LIBPATH
```

4. Execute the `bootstrap.sh` file with the `-32bit` parameter (in addition to other required parameters).

## IBM AIX 5L versions 5.2 and 5.3 (64-bit)

1. Ensure the `libstc_jnisna.so` file resides in a directory specified in the system LIBPATH statement. Refer to **"Installing the SNA eWay on an eGate supported system" on page 16** for instructions on how to download and configure the `libstc_jnisna.so` file.
2. Add the IBM Communication Server directory path (`usr/lib/sna`) to the system LIBPATH.
3. From a command prompt, execute the following:

```
EXPORT OBJECT_MODE=64

EXPORT LIBPATH
```

4. Execute the `bootstrap.sh` file.

## Sparc (32-bit)

For Sparc 32-bit platforms, the SNA eWay supports two third-party SNA servers — SNAP-IX, and Brixton. Both the SNAP-IX and Brixton libraries can be executed in either 32-bit or 64-bit modes. Refer to **"Installing the SNA eWay on an eGate supported system" on page 16** for instructions on how to download and configure the `libstc_jnisna.so` file.

1. Ensure the runtime bridge library, `libstc_jnisna.so`, for either the **SNA eWay - Runtime sparc32 SNAP-IX bridge so** or the **SNA eWay - Runtime sparc32 Brixton**

**bridge so**, resides in a directory that is specified in the system **LD_LIBRARY_PATH**.

2. Ensure that the Brixton or SNAP-IX 32-bit library directory is specified in the system **LD_LIBRARY_PATH**.
3. Ensure the JVM for the Integration Server specified in your logical host is set to 32-bits. Refer to **"Configuring the Logical Host" on page 71** to configure the logical host.
4. Execute the **bootstrap.sh** file.

## Sparc (64-bit)

For bigger EJB applications hosted on Solaris 9, the 64-bit JVM has better performance. If you run the SeeBeyond Integration Server JVM in 64-bit mode, you must also use the 64-bit JNI bridge shared library and the corresponding SNAP-IX or Brixton 64-bit library.

1. Ensure the runtime bridge library, `libstc_jnisna.so`, for either the **SNA eWay - Runtime sparc64 SNAP-IX bridge so** or the **SNA eWay - Runtime sparc64 Brixton bridge so** reside in a directory that is specified in the system **LD_LIBRARY_PATH**.
2. Ensure that the Brixton or SNAP-IX 64-bit library directory is specified in the system **LD_LIBRARY_PATH**.
3. Ensure the JVM for the Integration Server specified in your logical host is set to 64-bits. Refer to **"Configuring the Logical Host" on page 71** to configure the logical host.
4. Execute the **bootstrap.sh** file.

## Configuring the Integration Server

You must set your SeeBeyond Integration Server Password property before deploying your Project.

1. From the Environment Explorer, right-click **IntegrationSvr1** under your **Logical Host**, and select **Properties** from the shortcut menu. The Integration Server Properties Editor appears.
2. Click the **Password** property field under **Sun SeeBeyond Integration Server Configuration**. An ellipsis appears in the property field.
3. Click the ellipsis. The **Password Settings** dialog box appears.
4. Enter **STC** as the **Specific Value** and as the **Confirm Password**, and click **OK**.
5. Click **OK** to accept the new property and close the Properties Editor.

For more information on deploying a Project see the *Sun SeeBeyond Java™ Composite Application Platform Suite Deployment Guide*.

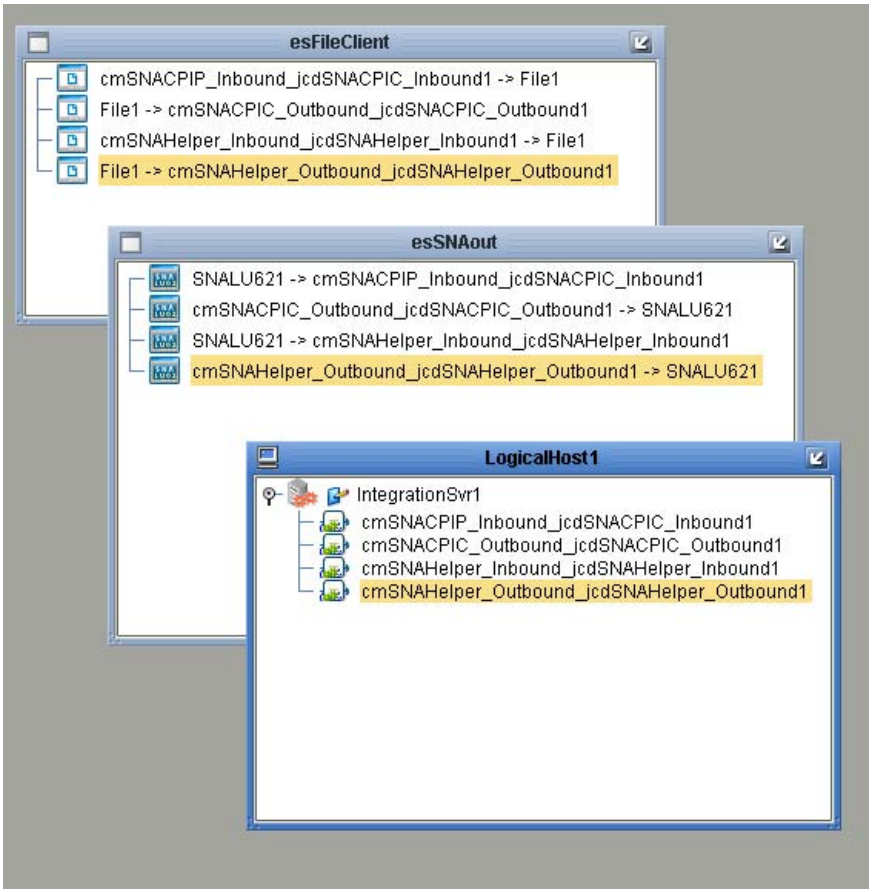
## 4.4.10 Creating the Deployment Profile

A Deployment Profile is used to assign services and message destinations to the Integration Server and message server. Deployment profiles are created using the Deployment Editor.

**Steps required to create the Deployment Profile:**

1. From the Enterprise Explorer's Project Explorer, right-click the **prjSNA_Sample_JCD** Project and select **New** > **Deployment Profile**.
2. Enter a name for the Deployment Profile (for this sample **dpSNA_JCD**). Select **envSNAProj** as the Environment and click **OK**.
3. From the Deployment Editor toolbar, click the **Automap** icon. The Project's components are automatically mapped to their system windows. See Figure 44.

**Figure 44** Deployment Profile




## 4.4.11 Creating and Starting the Domain

To build and deploy your Project, you must first create a domain. A domain is an instance of a Logical Host. After the domain is created, the Project is built and then deployed.

**Note:** *You are only required to create a domain once when you install the Sun Java Composite Application Platform Suite.*

**Steps required to create and start the domain:**

1. Navigate to your **<JavaCAPS51>\logicalhost** directory (where <JavaCAPS51> is the location of your Sun Java Composite Application Platform Suite installation).
2. Double-click the **domainmgr.bat** file. The **Domain Manager** appears.
3. If you have already created a domain, select your domain in the Domain Manager and click the **Start an Existing Domain** button. Once your domain is started, a green check mark indicates that the domain is running. Your domain will continue to run unless you shut it down.
4. If there are no existing domains, a dialog box indicates that you can create a domain now. Click **Yes**. The **Create Domain** dialog box appears.
5. Make any necessary changes to the **Create Domain** dialog box and click **Create**. The new domain is added to the Domain Manager. Select the domain and click the **Start an Existing Domain** button. Once your domain is started, a green check mark indicates that the domain is running.

For more information about creating and managing domains see the *eGate Integrator System Administration Guide*.

## 4.4.12 Building and Deploying the Project

The Build process compiles and validates the Project's Java files and creates the Project EAR file.

**Build the Project**

1. From the Deployment Editor toolbar, click the **Build** icon.
2. If there are any validation errors, a **Validation Errors** pane will appear at the bottom of the Deployment Editor and displays information regarding the errors. Make any necessary corrections and click **Build** again.
3. After the Build has succeeded you are ready to deploy your Project.

**Deploy the Project**

1. From the Deployment Editor toolbar, click the **Deploy** icon. Click **Yes** when the **Deploy** prompt appears.
2. A message appears when the project is successfully deployed. You can now test your sample.

## 4.4.13 Running the Sample

To run your deployed sample Project do the following:

1. From your configured input directory, paste (or rename) the sample input file to trigger the eWay.
2. From your output directory, verify the output data.

Appendix A

# Working with SNA Collaborations

The intention of this appendix is to provide guidelines to using the SNA eWay when you create SNA Java Collaborations and interfaces. Newly created Java Collaborations that implement the SNA OTD consist of skeleton SNA functionality required to use the Collaboration with your project. The Collaboration code generated upon SNA Java Collaboration creation is discussed throughout **Chapter 4**.

An SNA conversation is the connection between the two transaction programs (TP). When a TP want to communicate with another TP, it must first contact each of the other potential TPs and determine their state. Each transaction program in the conversation should be aware of the other TP and work together. If one side is expecting the other side to do perform certain operations, and vise versa, you should ensure that the behaviors are present, accurate, and accounted for; as, any change on one side of the conversation may affect the other side.

## A.1 Checking Conversation State

When writing Collaborations, it is a good programming procedure to check the pre-condition and post-condition for any outstanding function call or major logic check-point. Depending on the returned logic or function state, you can then determine the next course of action. For example, after you call an SNA receive function, you should check the `lastReturnCode`, exposed as an OTD node, as demonstrated below.

```
    public void receive(
        com.stc.connector.snalu62.inbound.SNAInboundApplication
        input, com.stc.connector.appconn.file.FileApplication
FileClient_1 )
        throws Throwable
    {
    // it works along with project SNAOut1_Confirm which sends data
and
    //   requests confirmation.
    // main logic: receive data over SNA, write it into file, then
confirm it.
        logger.info( "SNAIn1_Confirm: Started." );
        // you can do whatever other logic before/after this CPIC
call according
        // to your own actual case
        input.getCPICCalls().cmrcv();
        if (input.getLastReturnCode() !=
             input.getConstants().getReturnCodes().getCM_OK()) {
                logger.error( "SNAIn1_Confirm: cmrcv: Failed." );
```

```
                    throw new Exception( "SNAIn1_Confirm: cmrcv:
Failed." );
                    // or do your own error handling
          }
          //Conversation processing goes here
```

In the above example, a CPIC call, `input.getCPICCalls().cmrcv()`, is used to accept an initiate conversation request. A `getLastReturnCode` is immediately called on the inbound conversation to obtain the return code from the last SNA conversation-related function call. The logic, as implemented here, checks to see if the returned code from the `cmrcv()` call matches the return code value of `getCM_OK`. If the returned codes match, the data from the conversation is captured and output to the target file as specified in the remainder of the Collaboration code. If the return code obtained by the `input.getLastReturnCode()` call does not match, an exception is thrown and logged.

Similar in construct to the previous example, this next inbound conversation code segment uses the Confirmed (`cmcfmd`) CPIC call to send a confirmation reply to the remote program confirmation request. The local and remote programs can use the Confirmed and Confirm calls to synchronize their processing.

```
// you can do whatever other logic before/after this CPIC call
according
     to your design
     input.getCPICCalls().cmcfmd();
     if (input.getLastReturnCode() !=
          input.getConstants().getReturnCodes().getCM_OK()) {
          logger.error( "SNAIn1_Confirm: cmcfmd: Failed." );
          throw new Exception( "SNAIn1_Confirm: cmcfmd: Failed." );
          // or do your own error handling
     }
     //Conversation processing goes here
```

To maintain the cohesion between the inbound conversation Collaboration and the outbound conversation Collaboration, the outbound conversation code below will request a confirmation of the conversation state before proceeding. Without the outbound conversation Collaboration requesting the confirmation from the inbound conversation Collaboration, it is possible that unexpected results could occur since the confirmation codes are being sent to the requestor, even though the requestor didn't ask for confirmation.

```
     SNALU62eWay_1.getCPICCalls().cmcfm();
     if (SNALU62eWay_1.getLastReturnCode() !=
          SNALU62eWay_1.getConstants().getReturnCodes().getCM_OK()) {
          logger.error( "SNAOut1_Confirm: cmcfm: Failed." );
          throw new Exception( "SNAOut1_Confirm: cmcfm: Failed." );
          // or do your own error handling
     }
     //Conversation processing goes here
```

As you can see, the Confirm (`cmcfm`) call is used by the outbound conversation Collaboration to send a confirmation request to the inbound conversation Collaboration and then wait for a reply. The inbound conversation Collaboration replies with a Confirmed (`CMCFMD`) call. The inbound and outbound conversation Collaborations use the Confirm and Confirmed calls to synchronize their processing of data.

If your design requires further conversation synchronization, you can check the `lastStatus` (exposed as an OTD node) and/or check the `lastConversationState` (exposed as an OTD node); in addition to, other confirmation status check calls (see the SNA eWay Javadoc for additional information). Once the state is determined, your program flow can be modified based on these expected or unexpected conversation states. If you are not receiving expected returned values, you will know that something is wrong and can process the conversation accordingly.

Depending on which type of Java calls you use in your Collaboration code, you will need to select the proper calls that are related. A (non-exclusive) list of commly used related confirmation methods is provided below:

- cpic `cmcfm()`
- cpic `cmcfmd()`
- helper `confirm()`
- helper `confirmed()`
- helper `send()` or `send(true)`
- helper `recv()` or `recv(true)`

## A.2 Using CPIC Calls

For the users that want to use CPIC calls (`cmxxxx`) directly, the eWay and Integration Server manage conversation initiation and termination. Normally, it is not necessary to explicitly call the CPIC calls (e.g. `cmaccp`, `cminit`, `cmdeal`, etc.) that manage conversation initiation and termination. Unless your design requires you to manage the conversation on your own logic, of course risk also, you need not implement CPIC calls for conversation handshakes.

Appendix B

# Implementing the SNA Custom Handshake Class

To further utilize the capabilities of the SNA eWay, this appendix provides guidelines for implementing a custom handshake class in a deployed Project. After the default Collaboration is generated, you can then modify the Collaboration to suit your application's needs. While you will need to write your own code for both Inbound and Outbound SNA modes, the following code is also provided as the source for the class that is implemented in the eWay.

**Sample Code for Inbound Mode:**

```
package com.stc.connector.snalu62.api;

import com.stc.connector.logging.LogFactory;
import com.stc.connector.logging.Logger;

import com.stc.connector.snalu62.exception.SNAApplicationException;

/*
 * This is a sample class to implement the interface
SNACustomerHandshake.
 * It implements a simple Accept_Conversation scenario for windows
platform.
*/
public class SNACustomerHandshakeImplSampleAccept implements
SNACustomerHandshake {
   public static final String version = "cvs $Revision: 1.1.2.1.2.2 $
$Date: 2005/11/10 21:40:15 $";
    private Logger logger = LogFactory.getLogger("STC.eWay.SNALU62."
+ getClass().getName());
    private String logMsg;

    /**
     * Constructor
     *
     */
    public SNACustomerHandshakeImplSampleAccept() {
        super();
    }
    /**
     * @see
com.stc.connector.snalu62.api.SNACustomerHandshake#startConversation(
com.stc.connector.snalu62.api.SNACPICCalls)
     */
    public void startConversation(SNACPICCalls cpic) throws
SNAApplicationException {
        try {
            //do whatever checking logics before/after the following
CPIC call on your desires
```

```
            cpic.cmsltp();

            //do whatever checking logics before/after the following
CPIC call on your desires
            cpic.cmaccp();
           if (!cpic.getConversationAttributes().returnCodeIs(0) &&
// 0: CM_OK
               !cpic.getConversationAttributes().returnCodeIs(35)) {
//35: CM_OPERATION_INCOMPLETE
                logMsg =
"SNACustomerHandshakeImplSampleAccept.startConversation(): The return
code is <"
                   + cpic.getConversationAttributes().getReturnCode()
                    + ">.";
                logger.error(logMsg);
                throw new SNAApplicationException(logMsg);
            }

           if (cpic.getConversationAttributes().returnCodeIs(35)) {
//35: CM_OPERATION_INCOMPLETE

logger.info("SNACustomerHandshakeImplSampleAccept.startConversation()
: About to call cmwait ...");
                //do whatever checking logics before/after the
following CPIC call on your desires
                cpic.cmwait();
            }

            if (!cpic.getConversationAttributes().returnCodeIs(0) ||
              !cpic.getConversationAttributes().convReturnCodeIs(0))
{ // 0: CM_OK
                logMsg =
"SNACustomerHandshakeImplSampleAccept.startConversation(): The
return_Code is <"
                   + cpic.getConversationAttributes().getReturnCode()
                    + "> and the conversation_Return_Code is <"
                    +
cpic.getConversationAttributes().getConvReturnCode()
                    + ">. SNA conversation is not established.";
                logger.error(logMsg);
                throw new SNAApplicationException(logMsg);
            }

            //do whatever other logics on your desires here
            //...
        } catch (Exception e) {
            logMsg =
"SNACustomerHandshakeImplSampleAccept.startConversation(): Failed.
Got exception ["
                + e.toString()
                + "].";
            logger.error(logMsg, e);
            throw new SNAApplicationException(logMsg, e);
        }

    }

}
```

## Sample Code for Outbound Mode:

```
package com.stc.connector.snalu62.api;

import com.stc.connector.logging.LogFactory;
```

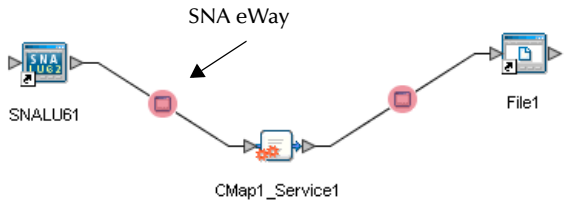
```
import com.stc.connector.logging.Logger;

import com.stc.connector.snalu62.exception.SNAApplicationException;

/**
 * This is a sample class to implement the interface
SNACustomerHandshake.
 * It implements a simple Initialize_Conversation scenario for
windows platform.
*/
public class SNACustomerHandshakeImplSampleInitialize implements
SNACustomerHandshake {
   public static final String version = "cvs $Revision: 1.1.2.1.2.2 $
$Date: 2005/11/10 21:40:15 $";
    private Logger logger = LogFactory.getLogger("STC.eWay.SNALU62."
+ getClass().getName());
    private String logMsg;

    /**
     * Constructor
     *
     */
    public SNACustomerHandshakeImplSampleInitialize() {
        super();
    }

    /**
     * @see
com.stc.connector.snalu62.api.SNACustomerHandshake#startConversation(
com.stc.connector.snalu62.api.SNACPICCalls)
     */
    public void startConversation(SNACPICCalls cpic) throws
SNAApplicationException {
        try {
            //do whatever checking logics before/after the following
CPIC call on your desires
            cpic.cminit();

            //do whatever checking logics before/after the following
CPIC call on your desires
            cpic.cmssl();

            //do whatever checking logics before/after the following
CPIC call on your desires
            cpic.cmallc();
           if (!cpic.getConversationAttributes().returnCodeIs(0)) { /
/ 0: CM_OK
                logMsg =
"SNACustomerHandshakeImplSampleInitialize.startConversation(): The
return_Code is <"
                    + cpic.getConversationAttributes().getReturnCode()
                     + ">. SNA conversation is not established.";
                logger.error(logMsg);
                throw new SNAApplicationException(logMsg);
            }

            //do whatever other logics on your desires here
            //...
        } catch (Exception e) {
            logMsg =
"SNACustomerHandshakeImplSampleInitialize.startConversation():
Failed. Got exception ["
                + e.toString()
                + "].";
```

```
                logger.error(logMsg, e);
                throw new SNAApplicationException(logMsg, e);
            }

        }

    }
```

## B.1 Importing a Custom Class

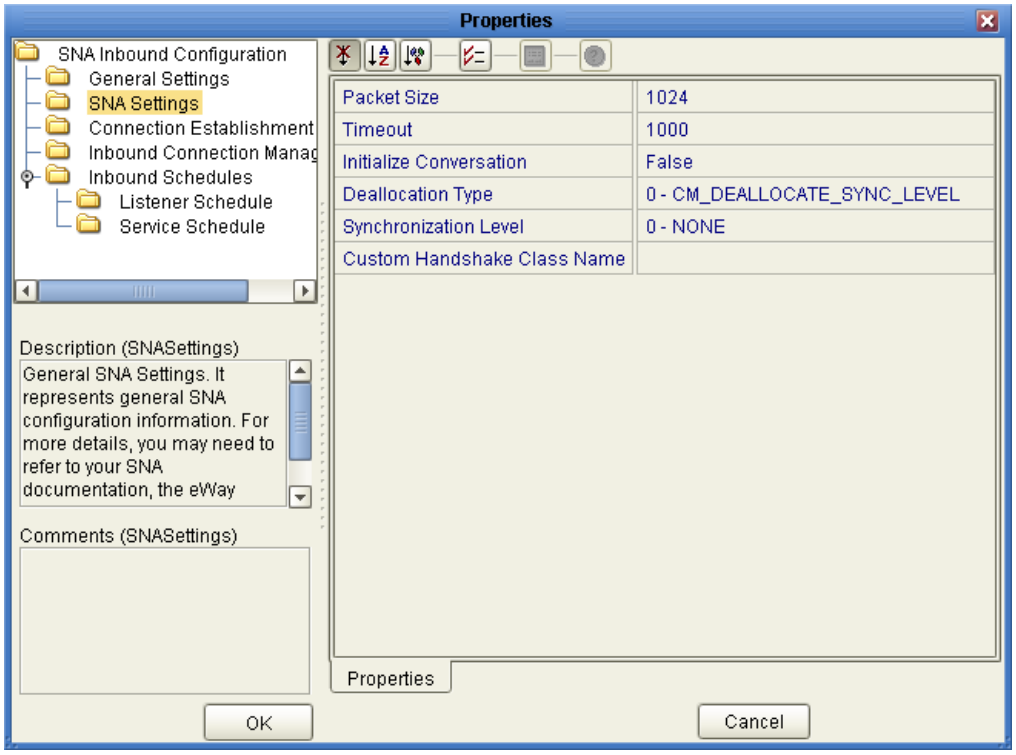
1 On the Enterprise Designer's Inbound SNA Connectivity Map, double-click the SNA eWay icon.

**Figure 45** Connectivity Map with Components - Inbound




The eWay Properties window appears, displaying the default properties for the Inbound eWay.

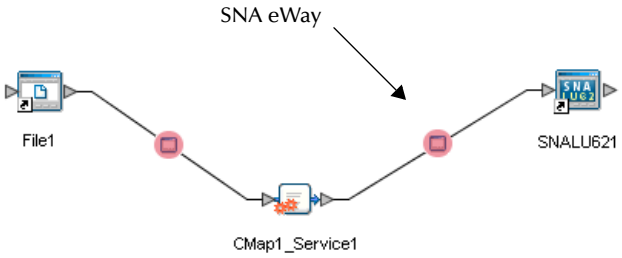
**Figure 46** Inbound eWay Properties




2 Edit the **Custom Handshake Class Name** property in the Inbound **Properties** window. For the sample code provided with the eWay, enter `com.stc.connector.snalu62.api.SNACustomerHandshakeImplSampleAccept`.

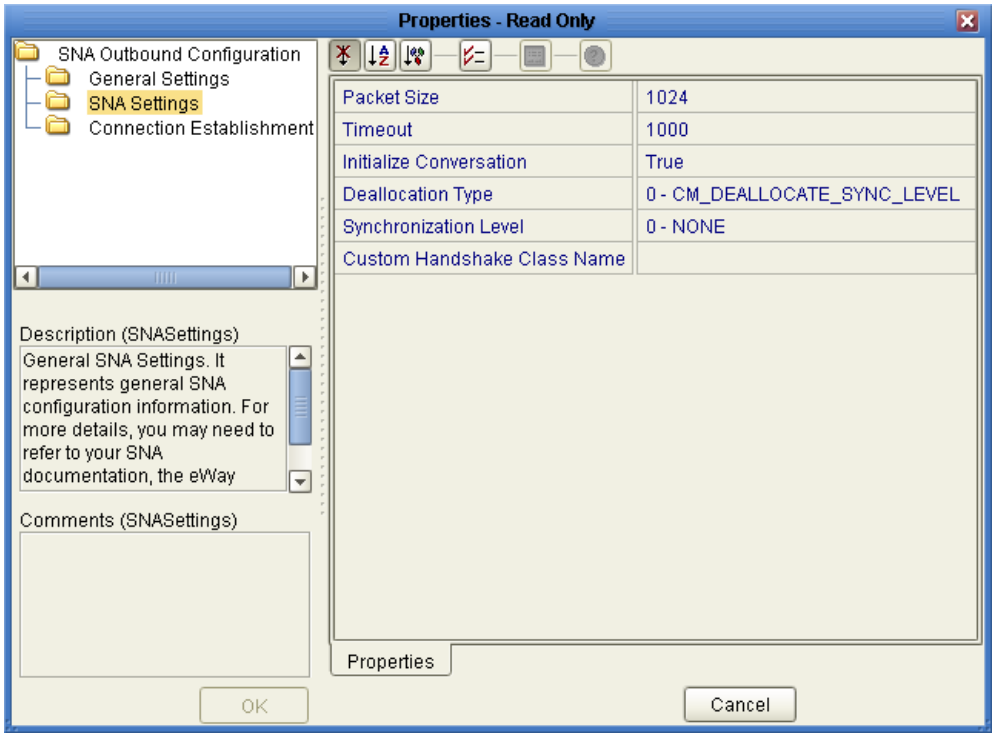
3 On the Enterprise Designer's Outbound SNA Connectivity Map, double-click the SNA eWay icon.

**Figure 47** Connectivity Map with Components - Outbound




The eWay Properties window appears, displaying the default properties for the Oubound eWay.

**Figure 48** Outbound eWay Properties




4 Edit the **Custom Handshake Class Name** property in the Outbound **Properties** window. For the sample code provided with the eWay, enter `com.stc.connector.snalu62.api.SNACustomerHandshakeImplSampleInitialize`.

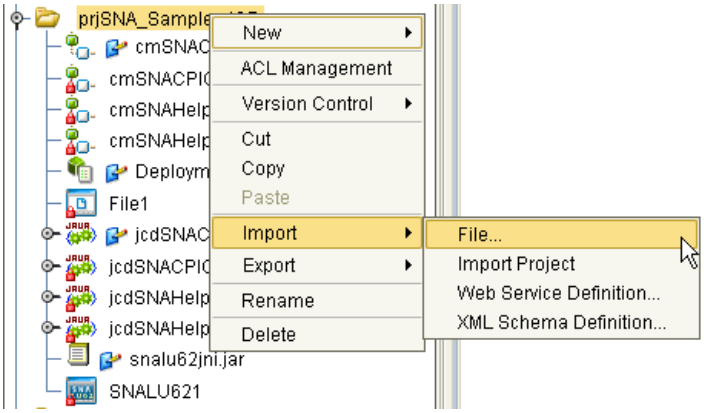
5 Redeploy your project (see **Building and Deploying the Project** on page 76 for further information).

**Steps when Building your own Class:**

1 Prepare a JAR file that includes your built class.

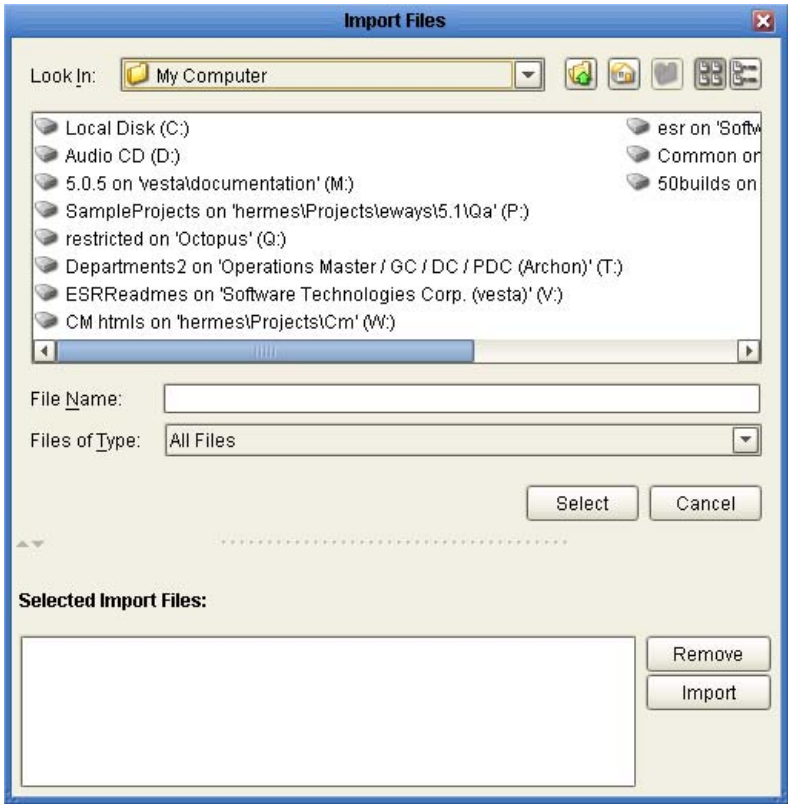
2 From the Project Explorer, right-click the sample Project and select **Import** > **File** from the shortcut menu.

**Figure 49** Importing a JAR File - Project Folder



The **Import Files** window appears.

**Figure 50** Import Files Window

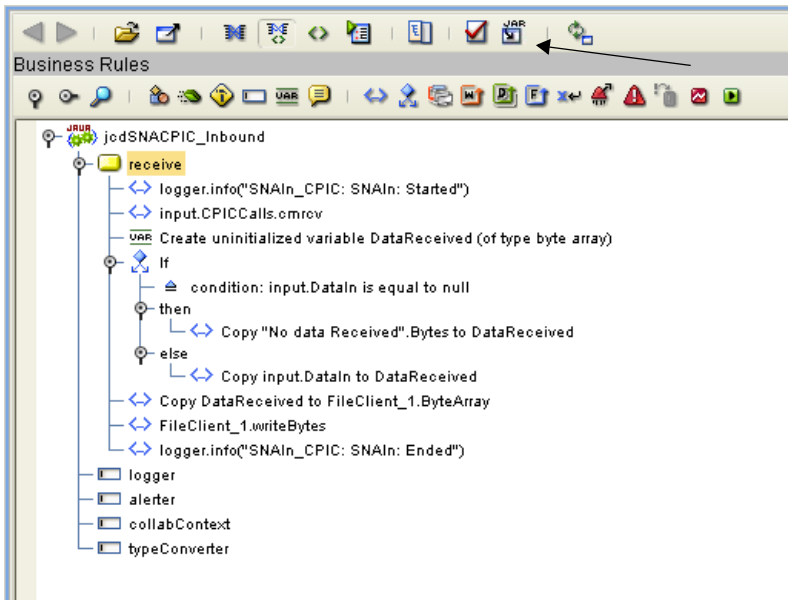



3 Locate your JAR file and click **Select**. Your selected JAR file appears in the **Selected Import Files** pane at the bottom of the **Import Files** window.

4 Click **Import**. Your selected JAR file appears in your sample Project's folder in the left pane of the Enterprise Designer.

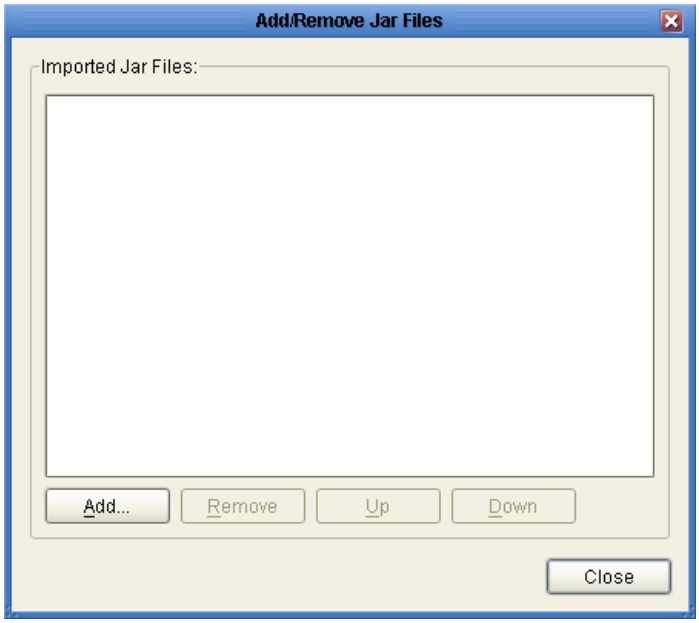
5 Click the **Import JAR file** button on the Business Rules toolbar in the right pane of Enterprise Designer.

**Figure 51** Importing a JAR File - Collaboration




The **Add/Remove JAR Files** window appears.

**Figure 52** Add/Remove JAR Files Window




6 Locate your JAR file and click **Add**. Your selected JAR file appears in the **Imported JAR Files** pane.

7 Click **Close**. Your selected JAR file appears under your sample Project's Collaboration in the left pane of the Enterprise Designer.

**Note:** *If you make any changes to the class, repeat steps 2 through 7.*

The Java Collaboration can handle the SNA connection completely using the sample Class (`com.stc.connector.snalu62.api.SNACustomerHandshakeImplSampleDummy`). This class has been implemented in the SNA eWay. The sample code for this custom class is as follows:

```
package com.stc.connector.snalu62.api;

import com.stc.connector.logging.LogFactory;
import com.stc.connector.logging.Logger;
import com.stc.connector.snalu62.exception.SNAApplicationException;

/**
 * This is a sample class to implement the interface
SNACustomerHandshake.
 * It implements a dummy handshake. That is, the method
startConversation() does not perform a function.
 * No SNA conversation is established inside this implementation
 * class. You should establish the SNA conversation manually (e.g. in
the java Collaboration).
*/

public class SNACustomerHandshakeImplSampleDummy implements
SNACustomerHandshake {
    public static final String version = "cvs $Revision: 1.1.2.2 $
$Date: 2005/11/10 21:40:15 $";
    private Logger logger = LogFactory.getLogger("STC.eWay.SNALU62."
+ getClass().getName());

    /**
     * Constructor
     *
     */
    public SNACustomerHandshakeImplSampleDummy() {
        super();
    }

    /**
     * @see
com.stc.connector.snalu62.api.SNACustomerHandshake#startConversation(
com.stc.connector.snalu62.api.SNACPICCalls)
     */
    public void startConversation(SNACPICCalls cpic) throws
SNAApplicationException {

logger.info("SNACustomerHandshakeImplSampleDummy.startConversation():
Done nothing here.");

    }

}
```

# Index

## Numerics



## A



## B



## C



## D



## E



## I



## J



## L



## M



## O



## P



## R



## S

## T



## W