SUN SEEBEYOND

eWAY™ TCP/IP ADAPTER USER'S GUIDE

Release 5.1.1

Sun microsystems

Version 20060620102537

Contents

Chapter 1

Introduction 6

About the Sun SeeBeyond eWay™ TCP/IP Adapter 6

What's New in This Release 7

About This Document 7
What's in This Document 7
TCP/IP eWay Javadoc 7
Intended Audience 8
Scope 8
Text Conventions 8

Sun Microsystems, Inc. Web Site 8

Documentation Feedback 9

Chapter 2

Installing the TCP/IP eWay 10

TCP/IP eWay System Requirements 10

Installing the TCP/IP eWay 10
Installing the TCP/IP eWay on an eGate Supported System 11
Adding the eWay to an Existing Suite Installation 11
Installing eWay Enterprise Manager plug-ins 12
TCP/IP eWay Alert Codes 12
After Installation 13

ICAN 5.0 Project Migration Procedures 13

Chapter 3

Configuring the TCP/IP eWay 16

Creating and Configuring the TCP/IP eWay 16
Selecting TCP/IP as the External Application 16
Configuring the TCP/IP eWay Properties 17
Using the Properties Editor 18
Inbound and Outbound Properties 19

TCP/IP Inbound eWay Connectivity Map Properties 19

General Inbound Settings 20
TCPIP Inbound Settings 21
TCPIP Inbound Settings - Server Port Binding 23
TCPIP Inbound Settings - Client Connection Establishment 24
TCPIP Inbound Settings - Inbound Connection Management 25
TCPIP Inbound Schedules - Listener Schedule 27
TCPIP Inbound Schedules - Service Schedule 29
Envelope Message 31
Message Envelope Types 33

TCP/IP Outbound eWay Connectivity Map Properties 36
General Outbound Settings 36
TCPIP Outbound Settings 37
TCPIP Outbound Settings - Connection Establishment 39
TCPIP Outbound Settings - Server Port Binding 41
Envelope Message 42

TCP/IP eWay Environment Properties 46
TCPIP Server (inbound) eWay - General Inbound Settings 46
TCPIP Server (inbound) eWay - TCPIP Inbound Settings 47
TCPIP Server (inbound) eWay - MDB Pool Settings 48
TCPIP Client (outbound) eWay - General Outbound Settings 50
TCPIP Client (outbound) eWay - TCPIP Outbound Settings 51
TCPIP Client (outbound) eWay - Connection Pool Settings 52

Chapter 4

Implementing the Sample Project 54

TCP/IP eWay Implementation 54
TCP/IP eWay Components 54

Importing a Sample Project 55

TCP/IP Sample Project Overview 56

Creating the prjTCPIPSample Project 57
Creating a Project 57
Creating the Java Collaboration Definitions 57
Creating the jcdTCPServer Java Collaboration 58
Creating the Business Rules 59
Create the jcdTCPClient Collaboration Business Rules 59
jcdTCPServer Collaboration Business Rules 63
Creating a Connectivity Maps 66
Selecting the External Applications 66
Populating the Connectivity Maps 66
Binding the eWay Components 68
Creating the Project's Environment 69
Configuring the eWays 70
Configuring the TCP/IP eWays 71
Configuring the Integration Server 72
Creating the Deployment Profile 72
Creating and Starting the Domain 72
Building and Deploying the Project 73
Running the Sample 73

Customized Enveloping 74

Chapter 5

Using eWay Java Methods 78

TCP/IP eWay Methods and Classes: Overview 78

eWay Java Classes/Interfaces 78

TCP/IP eWay Javadoc 78

Protected and Unsupported Methods 79

Unsupported Methods 79

Index 80

Chapter 1

Introduction

This guide explains how to install, set properties for, and operate the Sun SeeBeyond eWay™ TCP/IP Adapter, referred to as the TCP/IP eWay throughout this guide.

This chapter provides a brief overview of operations, components, and general features of the TCP/IP eWay.

What's In This Chapter

- **About the Sun SeeBeyond eWay™ TCP/IP Adapter** on page 6
- **What's New in This Release** on page 7
- **About This Document** on page 7
- **Sun Microsystems, Inc. Web Site** on page 8

1.1 About the Sun SeeBeyond eWay™ TCP/IP Adapter

The Sun SeeBeyond eWay™ TCP/IP Adapter (referred to as the TCP/IP eWay throughout this document) enables the eGate™ Integrator to communicate with client applications using TCP/IP, and provides real-time, reliable data transfer for systems that support TCP/IP.

For details on operating and using eGate Integrator and its user interface, see the *Sun SeeBeyond eGate™ Integrator User's Guide.*

The TCP/IP eWay allows you to create a client interface to a server or implement a TCP/IP server in eGate, using a Collaboration framework created using the eGate Enterprise Designer. This interface allows your system to communicate via TCP/IP.

The TCP/IP Object Type Definition (OTD) enables the creation of any messaging protocol capable of running over TCP/IP, and also utilizes the common eWay services available in eGate.

The eWay also allows you to select a desired message envelope, using predefined envelope types. If these types do not meet your needs, you can customize your own envelope, using specialized eWay interfaces designed for this purpose.

1.2 What's New in This Release

The Sun SeeBeyond eWay™ TCP/IP 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.
- 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 to read eWay configurations from LDAP at runtime: eWay configuration properties now support LDAP URL values.
- Connection Retry Support: Allows you to specify the number of attempts to reconnect, and the interval between retry attempts, in the event of a connection failure.

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

1.3 About This Document

This guide explains how to install, configure, and operate the Sun SeeBeyond eWay™ TCP/IP Adapter, referred to as the TCP/IP eWay throughout this guide.

1.3.1 What's in This Document

This document includes the following chapters:

- Chapter 1 **"Introduction"**: Provides an overview of the Sun SeeBeyond eWay™ TCP/IP Adapter as well as high-level information about this document.
- Chapter 2 **"Installing the TCP/IP eWay"**: Describes the process and properties to configure the TCP/IP eWay to run in your Environment.
- **"Configuring the TCP/IP eWay"**: Describes the process and properties to configure the TCP/IP eWay to run in your Environment.
- Chapter 4 **"Implementing the Sample Project"**: Provides instructions for installing and running the sample Projects.
- Chapter 5 **"Using eWay Java Methods"**: Provides API details about exposed Java methods that may be used with collaborations.

TCP/IP eWay Javadoc

A TCP/IP eWay Javadoc is also provided, which documents the Java methods available with the TCP/IP eWay. The Javadoc is uploaded with the eWay's documentation file (TCPIPeWayDocs.sar) and downloaded from the Documentation tab of the Sun Java™ Integrator Suite Installer. To access the full Javadoc, extract the Javadoc to an easily accessible folder, and double-click the index.html file.

1.3.2 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.

1.3.3 Scope

This document describes the process of installing, configuring, and running the TCP/IP eWay. This document does not cover the Java methods exposed by this eWay. For information on the Java methods, download and view the TCP/IP eWay Javadoc files from the Enterprise Manager.

1.3.4 Text Conventions

The following conventions are observed throughout this document.

Table 1 Text Conventions

Text Convention	Used For	Examples
Bold	Names of buttons, files, icons, parameters, variables, methods, menus, and objects	▪ Click **OK**. ▪ On the **File** menu, click **Exit**. ▪ Select the **eGate.sar** file.
`Monospaced`	Command line arguments, code samples; variables are shown in ***bold italic***	`java -jar` ***filename***`.jar`
Blue bold	Hypertext links within document	See **Text Conventions** on page 8
Blue underlined	Hypertext links for Web addresses (URLs) or email addresses	http://www.sun.com

1.4 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

1.5 Documentation Feedback

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

CAPS_docsfeedback@sun.com

Chapter 2

Installing the TCP/IP eWay

This chapter explains how to install the TCP/IP eWay.

What's in This Chapter

- **TCP/IP eWay System Requirements** on page 10
- **Installing the TCP/IP eWay** on page 10
- **Installing eWay Enterprise Manager plug-ins** on page 12
- **ICAN 5.0 Project Migration Procedures** on page 13

2.1 TCP/IP eWay System Requirements

The TCP/IP eWay Readme contains the latest information on:

- Supported Operating Systems
- System Requirements
- External System Requirements
- The TCP/IP eWay Readme is uploaded with the eWay's documentation file (TCPIPeWayDocs.sar) and can be accessed from the Documentation tab of the Sun Java™ Composite Application Platform Suite Installer. Refer to the TCP/IP eWay Readme for the latest requirements before installing the TCP/IP eWay.

2.2 Installing the TCP/IP 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 Enterprise Manager running on a Windows platform connected to the Repository server using Internet Explorer.*

2.2.1 Installing the TCP/IP 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 eGate™ or eInsight™, 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), expand the **eWay** option.
2. Select the products for your Sun Java™ Composite Application Platform Suite and include the following:
   - **FileeWay** (the File eWay is used by most sample Projects)
   - **TCPIPeWay**

   To upload the TCP/IP eWay User's Guide, Help file, Javadoc, Readme, and sample Projects, select the following:
   - **TCPIPeWayDocs**
3. 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.
4. 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.
5. 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 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 finished, 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.

2.2.2 Installing eWay Enterprise Manager plug-ins

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

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

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

1 From the Enterprise Manager:

   A From the **Enterprise Manager**'s Explorer toolbar, click the **Configuration** icon.

   B Click the **Web Applications Manager** tab, go to the **Auto-Install from Repository** tab, and connect to your Repository.

   C Select the application plug-ins you require, and click **Install**. This should include the **eWays Base Enterprise Manager Plug-In** and the **TCPIP eWay Enterprise Manager Plug-In.** The application plug-ins are installed and deployed.

2 From the **Sun Java™ Composite Application Platform Suite Installer:**

   A From the **Sun Java™ Composite Application Platform Suite Installer's Download tab**, select the Plug-Ins you require and save them to a temporary directory.

   B Log onto the **Sun SeeBeyond Enterprise Manager**. From the **Enterprise Manager**'s Explorer toolbar, click the **Configuration** icon.

   C Click the **Web Applications Manager** tab and go to the **Manage Applications** tab.

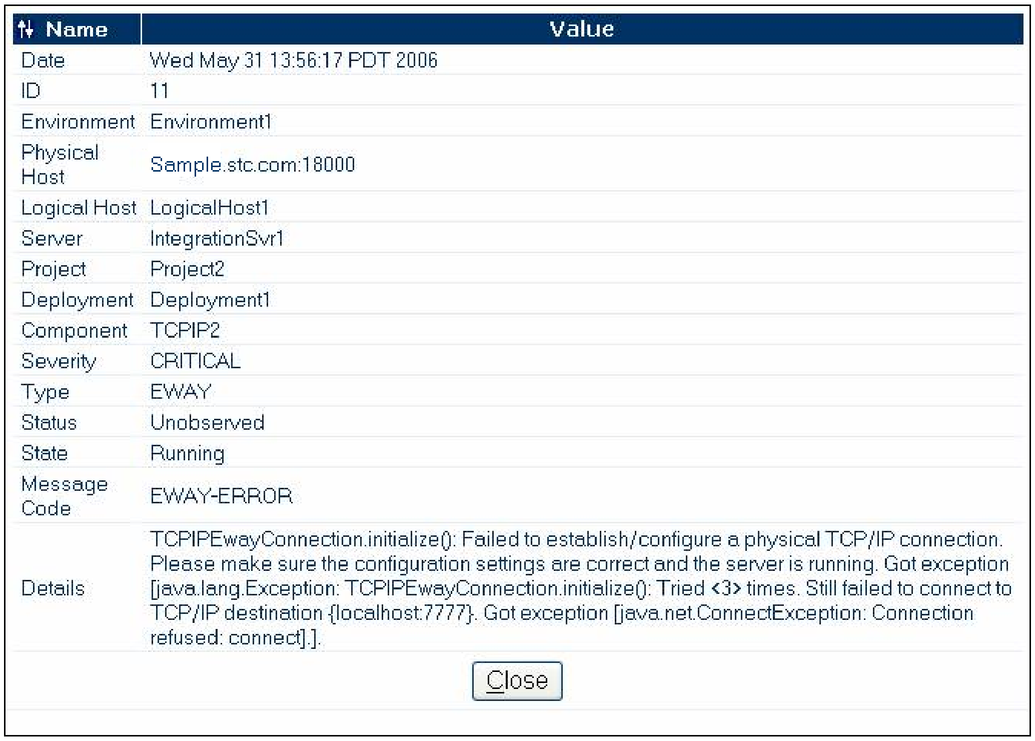
   D Browse for and select the WAR file for the application plug-in that you downloaded, and click **Deploy**. The plug-in is installed and deployed.

TCP/IP eWay Alert Codes

Alerts are viewed from the Enterprise Manager. Alerts are triggered when specified error conditions occur for a Project component. The purpose of the alert is to warn the administrator or user that the condition has occurred.

The TCP/IP eWay utilizes the default eWay alert codes. The alert Message Code displays as **EWAY-ERROR**, followed by a specific dynamic description of the error condition, as displayed in **TCP/IP eWay Alert** on page 13.

Figure 1 TCP/IP eWay Alert



Name	Value
Date	Wed May 31 13:56:17 PDT 2006
ID	11
Environment	Environment1
Physical Host	Sample.stc.com:18000
Logical Host	LogicalHost1
Server	IntegrationSvr1
Project	Project2
Deployment	Deployment1
Component	TCPIP2
Severity	CRITICAL
Type	EWAY
Status	Unobserved
State	Running
Message Code	EWAY-ERROR
Details	TCPIPEwayConnection.initialize(): Failed to establish/configure a physical TCP/IP connection. Please make sure the configuration settings are correct and the server is running. Got exception [java.lang.Exception: TCPIPEwayConnection.initialize(): Tried <3> times. Still failed to connect to TCP/IP destination {localhost:7777}. Got exception [java.net.ConnectException: Connection refused: connect].].

For information on Managing and Monitoring alert codes and logs, see the *Sun SeeBeyond eGate Integrator System Administration Guide.*

2.2.3 After Installation

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

2.3 ICAN 5.0 Project Migration Procedures

This section describes how to transfer your current ICAN 5.0 Projects to Sun Java Composite Application Platform Suite, version 5.1.1. Only Projects developed on ICAN 5.0.2 and above can be migrated successfully to the Sun Java Composite Application Platform Suite. To migrate your ICAN 5.0 Projects, do the following:

Export the Project

1. Before you export your Projects, save your current ICAN 5.0 Projects to your Repository.
2. From the Project Explorer, right-click your Repository and select **Export** from the shortcut menu. The Export Manager appears.
3. From the **Select Projects from the list field** of the Export Manager, select one or more Projects that you want to export and move them 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, from the **Select Environments from the list** field, select the Environments that you want to export and move them 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 Sun Java Composite Application Platform Suite

7. Install Sun Java Composite Application Platform Suite, including all eWays, libraries, and other components used by your ICAN 5.0 Projects.
8. Start the Sun SeeBeyond Enterprise Designer.

Import the Project

9. From the Enterprise Designer's Project Explorer, right-click the Repository and select **Import Project** from the shortcut menu. The Import Manager appears.
10. Browse to and select your exported Project file.
11. 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 Sun Java Composite Application Platform Suite 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.
12. Close the Import Manager after the Project is successfully imported.

Deploy the Project

13. You must create a new Deployment Profile for each of your imported Projects. When you export a Project, the Project's components are automatically *"checked in"* to Version Control to write-protect 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 **Check Out**.

    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 **Check Out**.

14 If your imported Projects include File eWays, they must be reconfigured in your Environment prior to deploying the Project. To reconfigure your File eWays, do the following:

   A The Environment File External System properties can now accommodate both inbound and outbound eWays. If your previous Environment includes both inbound and outbound File External Systems, delete one of these (for example, the outbound File External System).

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

   C The Directory property has been relocated from the Connectivity Map Properties to the Environment Properties. Set the inbound and outbound Directory values, and click **OK**.

15 Deploy your Projects.

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

Chapter 3

Configuring the TCP/IP eWay

This chapter explains how to set the properties for the TCP/IP eWay.

What's in This Chapter

- **Creating and Configuring the TCP/IP eWay** on page 16
- **Using the Properties Editor** on page 18
- **TCP/IP Inbound eWay Connectivity Map Properties** on page 19
- **TCP/IP Outbound eWay Connectivity Map Properties** on page 36
- **TCP/IP eWay Environment Properties** on page 46

3.1 Creating and Configuring the TCP/IP eWay

All eWays contain a set of parameters with properties that are unique to that eWay type. The TCP/IP eWay properties are modified from these 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 TCP/IP External System window.
- **Collaboration**: TCP/IP eWay properties may also be set from your Collaboration, in which case the settings will override the corresponding properties in the eWay's configuration file. Any properties that are not overridden retain their configured default settings.

3.1.1 Selecting TCP/IP as the External Application

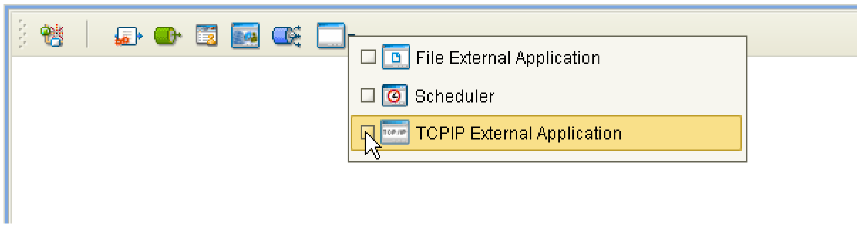
To create a TCP/IP eWay you must first create a TCP/IP External Application in your Connectivity Map. TCP/IP eWays are located between a TCP/IP External Application and a Service. Services are containers for Collaborations, Business Processes, eTL processes, and so forth.

To create the TCP/IP **External Application**

1. From the Connectivity Map toolbar, click the **External Applications** icon.

2 Select the **TCP/IP External Application** from the menu (see Figure 2). The TCP/IP External Application icon now appears on the Connectivity Map toolbar.

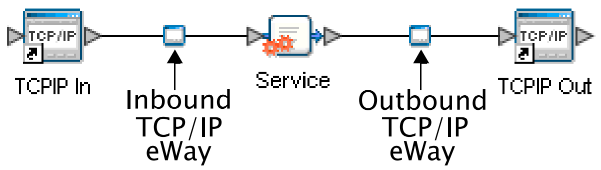
Figure 2 External Applications Selection Menu



3 Drag the new **TCP/IP External Application** from the toolbar onto the Connectivity Map canvas. This icon now represents an external TCP/IP system.

From the Connectivity Map, you can associate (bind) the External Application to the Service to establish an eWay (see Figure 3).

Figure 3 eWay Location.



When TCP/IP is selected as the External Application, it automatically applies the default TCP/IP eWay properties, provided by the OTD, to the eWay that connects it with the Service. These properties can then be or modified for your specific system using the **Properties Editor**.

3.1.2 Configuring the TCP/IP eWay Properties

A Project's eWay properties can be modified after the eWay has been established in the Connectivity Map and the Environment has been created.

Configuring the TCP/IP eWay (Connectivity Map) Properties

1 From the **Connectivity Map**, double click the eWay icon located in the link between the associated External Application and the Service.

2 The eWay **Properties Editor** appears with a template containing the inbound or out bound TCP/IP eWay Connectivity Map properties, depending on the type of component eWay that you double-click. Make any necessary changes to the property values and click **OK** to save the settings.

Configuring the TCP/IP eWay (Environment Explorer) Properties

1 From the **Environment Explorer** tree, right-click the TCP/IP External System. Select **Properties** from the shortcut menu. The **Properties Editor** opens with the TCP/IP eWay Environment properties. This contains the properties for both inbound and outbound component eWays.

2 Make any necessary changes to the Environment property values, and click **OK** to save the settings.

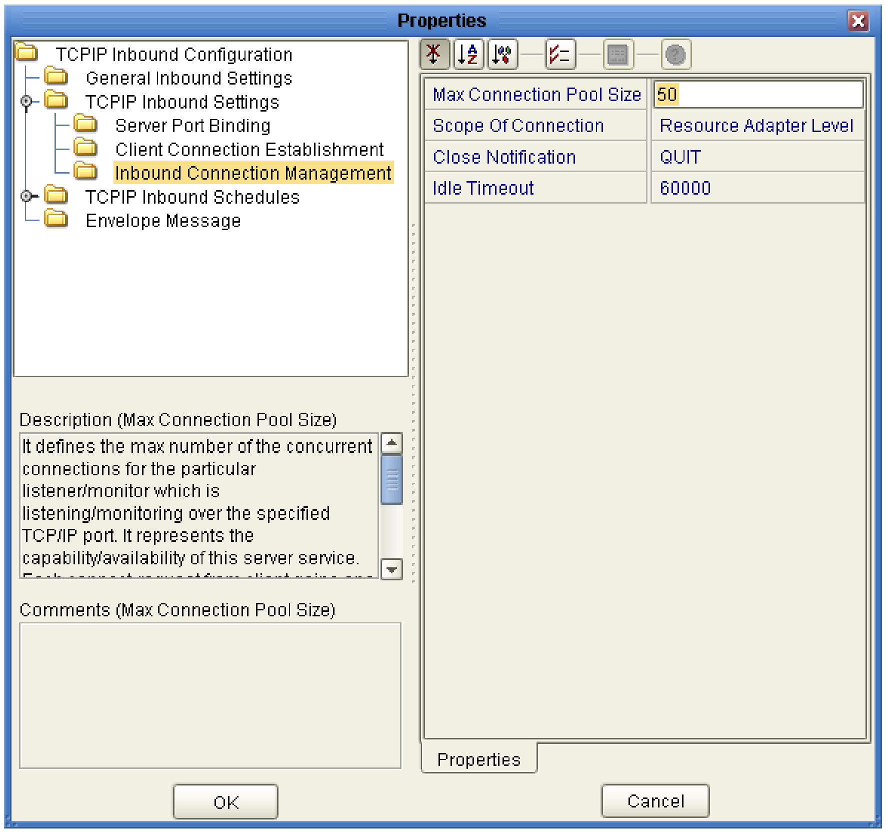
3.1.3 Using the Properties Editor

Modifications to the eWay configuration properties are made from the TCP/IP eWay **Properties Editor**.

Modifying the Default eWay Configuration Properties

1. From the Connectivity Map, double-click the inbound TCP/IP eWay to open the Properties Editor to the eWay's default properties.
2. From the upper-right pane of the Properties Editor, select a subdirectory of the configuration directory. The parameters contained in that subdirectory are now displayed in the Properties pane of the Properties Editor. For example, if you click on the **TCPIP Inbound Settings > Inbound Connection Management** subdirectory, the editable parameters are displayed in the right pane (see Figure 4).

Figure 4 Properties Editor -- Inbound TCP/IP Properties



Max Connection Pool Size	50
Scope Of Connection	Resource Adapter Level
Close Notification	QUIT
Idle Timeout	60000

Description (Max Connection Pool Size)

It defines the max number of the concurrent connections for the particular listener/monitor which is listening/monitoring over the specified TCP/IP port. It represents the capability/availability of this server service.

Comments (Max Connection Pool Size)

3. Click on any property field to make it editable. For example, click on the **Max Connection Pool Size** property to edit its property value. If a property value is true/false or multiple choice, the field displays a submenu of property options.
4. Click on the ellipsis (. . .) in the properties field to open a separate configuration dialog box. This is helpful for large values that cannot be fully displayed in the parameter's property field. Enter the property value in the dialog box and click **OK**. The value is now displayed in the property field.

5. A description of each property is displayed in the **Description** pane when that property is selected. This provides a brief explanation of the required settings or options.
6. The **Comments** pane provides an area to record notes and information regarding the currently selected property. These comments are saved when you close the editor.
7. After modifying the configuration properties, click **OK** to close the Properties Editor and save your changes.

3.1.4 Inbound and Outbound Properties

The TCP/IP eWay operates in two modes, inbound and outbound, each of which contain properties that are set from the Connectivity Map. Clicking on the inbound eWay displays inbound eWay properties in the Property Editor. The eWay's Environment Explorer properties include both inbound and outbound. The rest of this chapter describes the eWay's properties in detail, under the following sections:

- **TCP/IP Inbound eWay Connectivity Map Properties** on page 19
- **TCP/IP Outbound eWay Connectivity Map Properties** on page 36
- **TCP/IP eWay Environment Properties** on page 46

3.2 TCP/IP Inbound eWay Connectivity Map Properties

The inbound property settings determine the eWay's behavior for input operations.

The TCP/IP eWay's Inbound Connectivity Map properties are organized into the following sections:

- **General Inbound Settings** on page 20
- **TCPIP Inbound Settings** on page 21
- **TCPIP Inbound Settings - Server Port Binding** on page 23
- **TCPIP Inbound Settings - Client Connection Establishment** on page 24
- **TCPIP Inbound Settings - Inbound Connection Management** on page 25
- **TCPIP Inbound Schedules - Listener Schedule** on page 27
- **TCPIP Inbound Schedules - Service Schedule** on page 29
- **Envelope Message** on page 31

3.2.1 General Inbound Settings

The **General Inbound Settings** properties provide the **dedicated session mode** and **maximum data size** message settings for the server. This section contains the top-level parameters displayed in Table 2.

Table 2 Connectivity Map - General Inbound Settings Properties

Name	Description	Required Value
Max Data Size	Allows you to define the maximum size of the data that the programs can hold internally.	The valid range is from 1 to 2 GB (the maximum value of the Java integer). The configured default is 2147483647
Scope Of State	Specifies the scope of State object, which is an OTD node. The valid options for this parameter are: ▪ **Resource Adapter Level:** The State has the same life cycle as the resource adapter. ▪ **Connection Level:** The State has the same life cycle as the connection. ▪ **OTD Level:** The State has the same life cycle as the OTD object. This scope represents the life cycle of the State.	**Resource Adapter Level**, **Connection Level**, or **OTD Level**. The configured default is **Resource Adapter Level**.
Dedicated Session Mode	Allows you to enable or disable the eWay's Dedicated Session Mode. When the Dedicated Session Mode is enabled in a server, the current client's request can exclusively hold the server port that it connects to. For example, if this property is enabled, and the client connects to a server, it only serves that client until its work is finished, and the session is disconnected. If another client tries to connect to the server during that time, it cannot until the session is done.	**True** or **False**. **True** enables the option. The configured default is **False**.

3.2.2 TCPIP Inbound Settings

The TCP/IP Inbound Settings properties provide the basic TCP/IP values for the server. The TCP/IP Inbound Settings properties contain the top-level parameters displayed in Table 3.

Table 3 Connectivity Map - TCP/IP Inbound Settings Properties

Name	Description	Required Value
Connection Type	Specifies how the eWay establishes the TCP/IP connection: **Client**: The eWay connects to an external server (host/port) to establish the connection. The eWay is in active mode. **Server**: The eWay waits/listens on a certain port for an incoming connection request from an external client. Once the request is received, the eWay accepts the request and establishes the connection. The eWay is in passive mode.	**Client** or **Server**. **Server** is the default setting. Unless you specifically require **Client** mode, leave **Server** as the default value.
ServerSoTimeout	Allows you to set or get the server **SO_TIMEOUT** value, in milliseconds.	The server's SO_TIMEOUT value in milliseconds. The default is **10,000** milliseconds (10 seconds).
Server Socket Factory Implementation Class Name	Enter the name of the Java class that implements the server socket factory. This class is used for creating the server socket. If you have provided your own server socket implementation, you must enter the name here, of the Java class that contains this implementation. The factory implementation class must implement the following interface: **`com.stc.connector.tcpip.model.factory.TCPIPSocketFactory`**	A valid Java class name. The default is: **`com.stc.connector.tcpip.model.factory.TCPIPSocketFactoryImpl`**
Keep Alive	Specifies whether the server's **SO_KEEPALIVE** option is enabled or disabled; used for the accepted client socket. ***Note:*** For some properties, the server socket itself does not have direct properties settings associated with it. Instead, the properties map to the accepted client socket.	**True** or **False**. **True** enables the option. The configured default is **True**.

Table 3 Connectivity Map - TCP/IP Inbound Settings Properties (Continued)

Name	Description	Required Value
Receive Buffer Size	Allows you to set or get the value of the server's **SO_RCVBUF** option for the current socket, that is, the buffer size used by the platform for input on the socket; used for the accepted client socket.	The receive buffer size. The configured default is **8192**.
Send Buffer Size	Allows you to set or get the value of the server's **SO_SNDBUF** option for the current socket, that is, the buffer size used by the platform for output on the socket; used for the accepted client socket.	The send buffer size. The configured default is **8192**.
SoLinger	Specifies whether the server's **SO_LINGER** option is enabled or disabled; used for the accepted client socket.	**True** or **False**. **True** enables the option. The configured default is **True**.
SoLinger Timeout	Specifies the server's linger time-out in seconds. The maximum time-out value is platform specific. The setting only affects the socket close; used for the accepted client socket.	The linger time-out in seconds. The configured default is **-1** seconds, indicating that the **SO_LINGER** option is disabled.
SoTimeout	Allows you to set or get the server's **SO_TIMEOUT** value, in milliseconds; used for the accepted client socket. A time-out of 0 (zero) is an infinite time-out. If you specify this value, the eWay goes into an infinite read. If this action happens, it is recorded in the eWay's log file.	The **SO_TIMEOUT** value in milliseconds. The default is **10,000** milliseconds (10 seconds).
TcpNoDelay	Specifies whether the server's **TCP_NODELAY** option (that is, Nagle's algorithm) is enabled or disabled; used for the accepted client socket.	**True** or **False**. **True** enables the option, and **False** disables it.

3.2.3 TCPIP Inbound Settings - Server Port Binding

The Server Port Binding properties define configuration parameters used for controlling the server port binding. The TCP/IP Inbound Settings - Server Port Binding properties contain the top-level parameters displayed in Table 4.

Table 4 Connectivity Map - TCP/IP Inbound Settings - Server Port Binding Properties

Name	Description	Required Value
Max Binding Retry	Specifies the maximum number of times the eWay will attempt to bind to the specified TCP/IP port on the localhost.	A number indicating the number of bind attempts. The configured default is **3**.
Retry Binding Interval	Specifies the amount of time (in milliseconds) to wait between attempts to bind to the specified TCP/IP port on the localhost.	A number indicating the length of time in milliseconds that the eWay waits between attempts. The configured default is **30000** (30 seconds).

3.2.4 TCPIP Inbound Settings - Client Connection Establishment

The **Client Connection Establishment** properties define some of the configuration parameters used for controlling the connection establishment. This section is only used when the **Connection Type** is set as **Client**. The **TCP/IP Inbound Settings - Client Connection Establishment** properties contain the top-level parameters displayed in Table 5.

Table 5 TCP/IP Inbound Settings - Client Connection Establishment Properties

Name	Description	Required Value
Time To Wait Before Attempting Connection	Specifies the length of time (in milliseconds) the eWay waits before attempting to connect to the external system.	A number indicating the length of time (in milliseconds) the eWay waits before attempts to connect. The configured default is **3000**.

3.2.5 TCPIP Inbound Settings - Inbound Connection Management

The **Inbound Connection Management** properties define parameters used for inbound Server Connection Management; for example, the connection pool and the life cycle of the accepted connection. The **TCP/IP Inbound Settings - Inbound Connection Management** properties contain the top-level parameters displayed in Table 6.

Table 6 TCP/IP Inbound Settings - Inbound Connection Management

Name	Description	Required Value
Max Connection Pool Size	Specifies the maximum number of the concurrent connections for the particular listener/monitor which is listening/monitoring over a specified TCP/IP port. It represents the capability/ availability of this server service. Each connect-request from a client gains one concurrent connection. This parameter also represents the maximum number of clients who can concurrently connect to this server service and get served by the particular listener/monitor at the same time.	A number indicating the maximum number of concurrent connections available from a listener/ monitor for a specific TCP/ IP port. **0** indicates that there is no limit. The configured default is **50**.
Scope Of Connection	Specifies the scope of the accepted connection which is used by the eWay. The two options are: ▪ **Resource Adapter Level**: The resource adapter will close the connection upon receiving a closure request, so the connection may “keep alive” during multiple executions of the Collaboration. ▪ **Collaboration Level**: The connection is closed once the Collaboration has been executed, so the connection has the same life cycle as the Collaboration.	**Resource Adapter Level** or **Collaboration Level**. The configured default value is **Resource Adapter Level**.
Close Notification	Specifies the close notification value. When the server receives a notification with content that matches this parameter's value, the server safely closes the connection and cancels any corresponding schedules.	A String indicating the trigger value that notifies the server to close the connection. The configured default is **QUIT**.

Table 6 TCP/IP Inbound Settings - Inbound Connection Management (Continued)

Name	Description	Required Value
Idle Timeout	Specifies the length of time (in milliseconds) for inactivity of the requestor (client). The eWay attempts to detect activity on client side (the other side of the connection). If no client activity (no i/o request comes over the connection from the client) for a specified time period, then the connection is closed from the server side to release the resource. The value is in milliseconds.	The length of time (in milliseconds) for inactivity of the requestor (client) before the connection is closed from the server side to release the resource. A value of **0** disables the IdleTimeout. The default value is **60000** (1 minute).

3.2.6 TCPIP Inbound Schedules - Listener Schedule

The **Listener Schedule** properties configure the scheduler used by the inbound TCP/IP Server. The server waits for a new client connection establishment request. These parameters are used to configure the listener/monitor that listens on the specified port.

Two J2EE schedulers are available (see **"Scheduler" on page 27**):

- **Timer Service**: This scheduler is configured using the **At Fixed Rate**, **Delay**, and **Period** properties.
- **Work Manager**: Available for **J2EE** (JCA 1.5 and above). This scheduler is configured using the **Delay** and **Period** properties.

Both schedulers provide the functionality required by the inbound TCP/IP Server.

The **TCPIP Inbound Schedules - Listener Schedule** section of the TCP/IP eWay Connectivity Map properties contains the top-level parameters displayed in Table 7.

Table 7 TCPIP Inbound Schedules - Listener Schedule Properties

Name	Description	Required Value
Scheduler	Specifies the scheduler type for this inbound communication. There are two options: ▪ **Timer Service**: This scheduler is configured using the At **Fixed Rate**, **Delay**, and **Period** properties. ▪ **Work Manager**: The task is scheduled through the J2EE Work Manager. Work Manager is supported by **J2EE** (JCA 1.5 and above). This scheduler is configured using the **Delay** and **Period** properties.	Select **Timer Service** or **Work Manager**. If your container doesn't support JCA Work Manager, select Timer Service.
Schedule Type	This property configuration, though visible from the Properties Editor, is disabled. The only available schedule type is **Repeated**, indicating that the task is scheduled for repeated execution at regular intervals defined by **Period** property in this section (see **"Period" on page 27**).	This property is disabled.
Delay	Applies to both the **Timer Service** or the **Work Manager**. Specifies, in milliseconds, the length of delay time before the task is executed.	The length of time before the task is executed, in milliseconds (1,000 milliseconds is equal to 1 second).
Period	Applies to both the **Timer Service** or the **Work Manager**. Specifies the regular interval, in milliseconds, between successive task executions.	The appropriate length of time between successive task executions, in milliseconds (1,000 milliseconds is equal to 1 second).

Table 7 TCPIP Inbound Schedules - Listener Schedule Properties (Continued)

Name	Description	Required Value
At Fixed Rate	Specific to the **Timer Service** configuration only. Specifies whether a **Fixed-Rate** execution or **Fixed-Delay** execution is used. - **Fixed-Rate:** A fixed-rate execution means that each execution is scheduled relative to the scheduled time of the initial execution. If an execution is delayed for any reason (such as garbage collection or other background activity), two or more executions will occur in rapid succession to "catch up." In the long run, the frequency of execution will be exactly the reciprocal of the specified period (assuming the system clock underlying Object.wait(long) is accurate). - **Fixed-Delay:** A fixed-delay execution means that each execution is scheduled relative to the actual time of the previous execution. If an execution is delayed for any reason (such as garbage collection or other background activity), subsequent executions will be delayed as well. As a result, the frequency of execution will generally be slightly lower than the reciprocal of the specified period (assuming the system clock underlying Object.wait(long) is accurate).	**True** or **False**. **True** indicates that a fixed-rate execution is used. **False** indicates that a fixed-delay execution is used.

3.2.7 TCPIP Inbound Schedules - Service Schedule

The **Service Schedule** properties configure the scheduler used by the TCP/IP Server that executes the business tasks (Collaboration Rules) over the existing connection. This scheduler affects the actual Business Rules defined by the user.

Two J2EE schedulers are available (see **"Scheduler" on page 27**):

- **Timer Service**: This scheduler is configured using the **At Fixed Rate**, **Delay**, **Period**, and **Schedule Type**, properties.
- **Work Manager**: available for **J2EE** (JCA 1.5 and above). This scheduler is configured using the **Delay**, **Period**, and **Schedule Type**, properties.

Both schedulers provide the functionality required by the inbound TCP/IP Server.

The **TCPIP Inbound Schedules - Service Schedule** section of the TCP/IP eWay Connectivity Map properties contains the top-level parameters displayed in Table 8.

Table 8 TCPIP Inbound Schedules - Service Schedule Properties

Name	Description	Required Value
Scheduler	Specifies the scheduler type for this inbound communication. There are two options: ▪ **Timer Service**: This scheduler is configured using the At **Fixed Rate**, **Delay**, and **Period** properties. ▪ **Work Manager**: The task is scheduled through the J2EE Work Manager. Work Manager is supported by **J2EE** (JCA 1.5 and above). This scheduler is configured using the **Delay** and **Period** properties.	Select **Timer Service** or **Work Manager**. If your container doesn't support JCA Work Manager, select Timer Service.
Schedule Type	Applies to both the **Timer Service** or the **Work Manager**. Specifies whether the task is scheduled to occur once or be repeated. ▪ **OneTime**: The task will be scheduled for one-time execution. ▪ **Repeated**: The task will be scheduled for repeated execution at regular intervals defined by **Period** property in this section (see **"Period" on page 30**).	Select **OneTime** or **Repeated**.
Delay	Applies to both the **Timer Service** or the **Work Manager**. Specifies, in milliseconds, the length of delay time before the task is executed.	The length of time before the task is executed, in milliseconds (1,000 milliseconds is equal to 1 second).

Table 8 TCPIP Inbound Schedules - Service Schedule Properties (Continued)

Name	Description	Required Value
Period	Applies to both the **Timer Service** or the **Work Manager**. Specifies the regular interval in milliseconds between successive task executions. This is used for the **Repeated** schedule type (see **"Schedule Type" on page 27**).	The appropriate length of time between successive task executions, in milliseconds (1,000 milliseconds is equal to 1 second).
At Fixed Rate	Specific to the **Timer Service** configuration only. Specifies whether a **Fixed-Rate** execution or **Fixed-Delay** execution is used. This is used for the "Repeated" schedule type by the "Timer Service" scheduler. **Fixed-Rate:** A fixed-rate execution means that each execution is scheduled relative to the scheduled time of the initial execution. If an execution is delayed for any reason (such as garbage collection or other background activity), two or more executions will occur in rapid succession to "catch up." In the long run, the frequency of execution will be exactly the reciprocal of the specified period (assuming the system clock underlying Object.wait(long) is accurate). **Fixed-Delay:** A fixed-delay execution means that each execution is scheduled relative to the actual time of the previous execution. If an execution is delayed for any reason (such as garbage collection or other background activity), subsequent executions will be delayed as well. As a result, the frequency of execution will generally be slightly lower than the reciprocal of the specified period (assuming the system clock underlying Object.wait(long) is accurate).	**True** or **False**. **True** indicates that a fixed-rate execution is used. **False** indicates that a fixed-delay execution is used.

3.2.8 Envelope Message

The **Envelope Message** section of the Inbound TCP/IP eWay Connectivity Map properties contains the top-level parameters displayed in Table 9.

Table 9 Connectivity Map - Envelope Message Properties

Name	Description	Required Value
Envelope Type	Specifies the envelope type. The envelope type defines where a message starts and stops. See **"Message Envelope Types" on page 33** for a description of the available envelope types and the structure of each. **BeginEndMarked** is supported by the properties **"Bytes to Read" on page 32**, **"Ignore Until Char Value" on page 33**, and **"Store Until Char Value" on page 33**. **EndMarked** is supported by the property **"Store Until Char Value" on page 33**. **FixedLength** is supported by the properties **"Bytes to Read" on page 32**. **LengthPrefixed** is supported by the property **"Width of Length" on page 32** and **"Numeric Representation" on page 32**. **MarkedAndFixed** is supported by the property **"Bytes to Read" on page 32**, **"Ignore Until Char Value" on page 33**, and **"Store Until Char Value" on page 33** **Custom** is supported by the properties **"Custom Enveloped Class Name" on page 32** and **"Customer Defined Property" on page 32**. **FixedLength** is supported by the property **"Bytes to Read" on page 32**. ***Note:*** *For all envelope types, except **MarkedAndFixed**, the data is just the payload. See* **"MarkedAndFixed" on page 34** *for an explanation of how the data is handled by that envelope type.*	Enter one of the following properties denoting the envelope type: ♦ **BeginEndMarked** ♦ **EndMarked** ♦ **FixedLength** ♦ **LengthPrefixed** ♦ **MarkedAndFixed** ♦ **PerActiveConnection** ♦ **Custom** The default is **BeginEndMarked**.

Table 9 Connectivity Map - Envelope Message Properties (Continued)

Name	Description	Required Value
Custom Enveloped Class Name	Specifies the Java class name to be used when the **Envelope Type** property is set to **Custom**. If you are using a custom envelope you have created, using a Java class, you can import the Java JAR file containing the class into any desired Collaboration, using the Collaboration Editor's file import feature. The class name should be a full qualified class name, such as "com.abc.MyClass". The class must implement interfaces com.stc.connector.tcpip.ext.msg.EnvelopedMsgReceiver and com.stc.connector.tcpip.ext.msg.EnvelopedMsgSender. For more details, see **Customized Enveloping** on page 74.	A full Java class name. A full qualified class name, or **None** if Custom is not the Envelope Type. The configured default is **None**.
Customer Defined Property	Used when the Envelope Type value is set to **Custom**. Specifies a list of user-defined parameters. You can parse this information, such as delimiters, into your customized envelope message implementation.	A text String.
Bytes to Read	Used with Envelope Types **FixedLength** and **MarkedAndFixed**. Specifies the number of bytes to read. It is assumed that all Events received by the eWay have the same length.	An integer indicting the number of bytes. The configured default is **1**.
Width of Length	Used for Envelope Type value **LengthPrefixed**. Specifies the width of the envelope length. In other words, it dictates the number of digits to be used to represent the Length field.	An integer; the range is 1 to 10. This property must be set to **2** for **Network short** and **4** for **Network long**. The configured default is **1**.
Numeric Representation	Used for Envelope Type value **LengthPrefixed**. Specifies how the number representation of the prefixed length is expressed. This value is expressed in one of the following formats: decimal, hexadecimal, octal, network short, or network long.	Select one of the following**:** ▪ **Decimal** ▪ **Hexadecimal** ▪ **Octal** ▪ **Network short** ▪ **Network long** The configured default is **Decimal**.

Table 9 Connectivity Map - Envelope Message Properties (Continued)

Name	Description	Required Value
Ignore Until Char Value	Used for the Envelope Types **BeginEndMarked** and **MarkedAndFixed**. Specifies the value for the ignore-until (same as begin block) character. All incoming characters are ignored until this character is encountered.	A decimal ascii number. The allowed range is 1 to 127. The configured default is **11.**
Store Until Char Value	Used for Envelope Types **BeginEndMarked**, **EndMarked**, and **MarkedAndFixed**. Specifies the character in the End Block or Marker position of the envelope. All incoming characters are stored until this character is encountered.	A decimal ascii number. The allowed range is 1 to 127. The configured default is **12.**

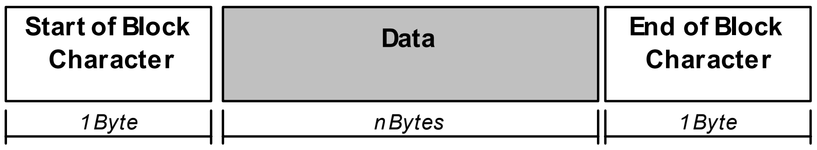
Message Envelope Types

The envelope type defines where a message starts and stops. This section provides a brief explanation of the available envelope types and the structure of each.

Note: *For all envelope types, except **MarkedAndFixed**, the data is just the payload. See* **MarkedAndFixed** *on page 34 for an explanation of how the data is handled by that envelope type.*

BeginEndMarked

The **BeginEndMarked** envelope has the following structure:

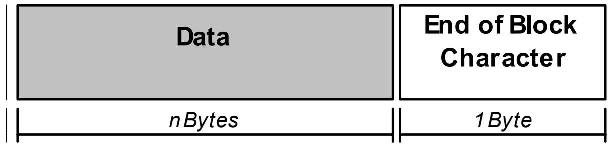


The **Start of Block Character** component of the Begin-end Marked envelope is the same as the editable **Ignore Until Character**. The **End of Block Character** component is the same as the editable **Store Until Character**.

If during the read process, the **Start of Block Character** is encountered, all read bytes are discarded and the read routine starts storing the incoming data from the last **Start of Block Character**.

EndMarked

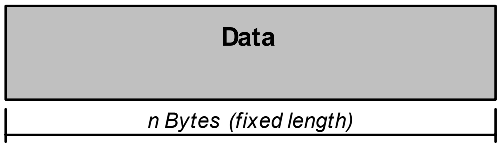
The **EndMarked** envelope has the following structure:



The **End of Block Character** component of the **EndMarked** envelope is the same as the editable **Store Until Character**.

FixedLength

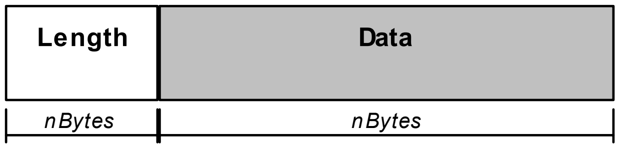
The **FixedLength** envelope has the following structure:



The **FixedLength** envelope type is set using the **Bytes to Read** editable property. It is assumed that all messages are the same length as specified by the **Bytes to Read** editable property.

LengthPrefixed

The **LengthPrefixed** envelope has the following structure:



The **Length** component of the **LengthPrefixed** envelope is an integer indicating the length of the **Data** component. The **Length** component has the following properties:

- **Numeric Representation** of the length
- **Allowed Width** of the length

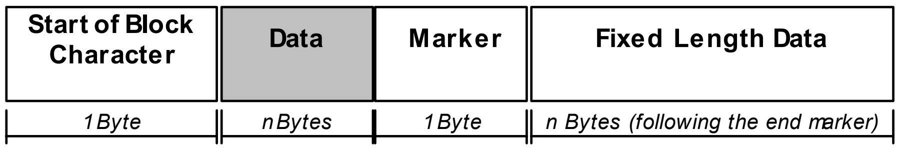
Table 10 displays the available values for each **Numeric Representation** and corresponding **Allowed Width**.

Table 10 Counter Component Values

Numeric Representation	Allowed Width
Decimal	1 to 10
Octal	1 to 8
Hexadecimal	1 to 16
Network short	2
Network long	4

MarkedAndFixed

The **MarkedAndFixed** envelope has the following structure:



The **MarkedAndFixed** envelope type is similar to the **BeginEndMarked** envelope, and is set using the properties **Ignore Until Character**, **Store Until Character**, and **Bytes to Read**.

The **Start of Block Character** in the diagram is the same as the property **Ignore Until Character**, and the **Marker** component is the same as the property **Store Until Character**. The communication client reads the marker before reading the remainder of the envelope as specified by the **Bytes to Read** property (**Fixed Length Data** in the diagram).

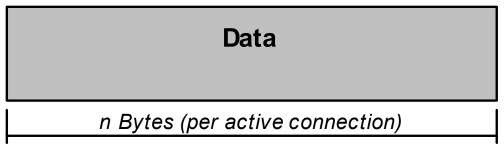
You can express this data structure in the following formula:

> ***Ignore Until Character*** *+ Marked Data +* ***Store Until Character*** *+ Fixed Data (Bytes to Read) =* ***Data Structure***

The **Ignore Until Character**, **Store Until Character**, and **Bytes to Read** values are represented by ASCII number properties settings in the eWay's properties.

PerActiveConnection

The **PerActiveConnection** envelope has the following structure:



The message is not enclosed in an envelope, and the current connection is closed by the eWay when an entire message is sent. The receiver knows the entire message has been received when the sender closes the connection.

If you choose the **PerActiveConnection** envelope type, **sendEnvelopedMsg()** must be last method you use with any corresponding TCP/IP OTD. Since **sendEnvelopedMsg()** sends the message and drops the connection, if you use additional methods afterward, they only return the error message "Connection was already closed."

Custom

The **Custom** envelope type refers to an envelope type you have created yourself, using a Java class. If want to use the class for a custom envelope, that is, a class you specified under the **Custom Enveloped Class Name** property (see **"Custom Enveloped Class Name" on page 32**), you must select this setting.

Note: *For optimum performance, it is recommended that you use the method* ***receiveEnvelopedMsg()*** *with any enveloped messages, because this method uses the envelope as its ending condition. However, the other receiving methods,* ***receiveBytes()*** *and* ***receiveString()****, use a time-out as their ending condition.*

3.3 TCP/IP Outbound eWay Connectivity Map Properties

The outbound TCP/IP eWay properties determine the eWay's behavior for output operations. The outbound TCP/IP eWay Connectivity Map properties are organized into the following sections:

- **General Outbound Settings** on page 36
- **TCPIP Outbound Settings** on page 37
- **TCPIP Outbound Settings - Server Port Binding** on page 41
- **TCPIP Outbound Settings - Connection Establishment** on page 39
- **Envelope Message** on page 42

3.3.1 General Outbound Settings

Specifies the general TCP/IP outbound configuration information. The **TCP/IP Outbound Settings - General Outbound Settings** properties contain the top-level parameters displayed in Table 11.

Table 11 Connectivity Map - General Outbound Settings Properties

Name	Description	Required Value
Max Data Size	Specifies the maximum amount of data that the programs can hold internally	The valid range is from **1** to **2147483647** bytes (2 GB - the maximum value of the Java integer). The configured default value is **2147483647**
Scope Of State	Specifies the scope of State object, which is an OTD node. The valid options for this parameter are: ▪ **Resource Adapter Level**: The State has the same life cycle as the resource adapter. ▪ **Connection Level**: The State has the same life cycle as the connection. ▪ **OTD Level**: The State has the same life cycle as the OTD object. This scope represents the life cycle of the State.	**Resource Adapter Level**, **Connection Level**, or **OTD Level**. The configured default is **Resource Adapter Level**.

3.3.2 TCPIP Outbound Settings

Presents the java Socket options. For more information, see the JDK Javadoc. The **TCPIP Outbound Settings** properties contain the top-level parameters displayed in Table 12.

Note: *For complete information on options referred to by these base settings, for example, SO_KEEPALIVE, see the appropriate Sun Microsystems Java documentation.*

Table 12 Connectivity Map - TCPIP Outbound Settings Properties

Name	Description	Required Value
Connection Type	Specifies how the eWay establishes the TCP/IP connection: **Client**: The eWay connects to an external server (host/port) to establish the connection. The eWay is in active mode. **Server**: The eWay waits/listens on a particular port for an incoming connection request from an external client. Once the request is received, the eWay accepts the request and establishes the connection. The eWay is in passive mode.	**Client** or **Server**. **Server** is the default setting. Unless you specifically require Server mode, leave this value as the default, **Client**.
ServerSoTimeout	Sets or gets the value of the SoTimeout for the ServerSocket, in milliseconds. Used for **ServerSocket.accept()**. When you set this option to a non-zero timeout, calling **accept()** for **ServerSocket** will block for only this period of time. If the timeout expires, a **java.net.SocketTimeoutException** (or **java.net.InterruptedIOException**) is thrown, though the ServerSocket remains valid. Enable this option prior to entering the blocking operation. This parameter is only used when the **Connection Type** is set as **Server**.	An integer that indicates the **SoTimeout** value in milliseconds. The configured default is **60000** (60 seconds). The timeout must be greater than **0** (zero). A timeout value of **0** is interpreted as an infinite timeout.
Keep Alive	Specifies whether the client's **SO_KEEPALIVE** option is enabled or disabled.	**True** or **False**. **True** enables the option. The configured default is **True**.

Table 12 Connectivity Map - TCPIP Outbound Settings Properties (Continued)

Name	Description	Required Value
Receive Buffer Size	Allows you to set or get the value of the client's **SO_RCVBUF** option for the current socket, that is, the buffer size used by the platform for input on the socket. It sets a "hint" as to the size of the underlying buffers used by the platform for incoming network I/O. When used in **set**, this is a suggestion to the kernel, from the application, regarding the buffer sizes to use for the data that will be received over the socket. When used in **get**, this must return the size of the buffer actually used by the platform when receiving in data on this socket.	An integer indicating the receive buffer size. The configured default is **8192**.
Send Buffer Size	Allows you to set or get the value of the client's **SO_SNDBUF** option for the current socket, that is, the buffer size used by the platform for output on the socket.	The send buffer size; the default is **8192**.
SoLinger	Specifies whether the client's **SO_LINGER** option is enabled or disabled.	**True** or **False**. **True** enables the option.
SoLinger Timeout	Specifies the client's linger time-out in seconds. The maximum time-out value is platform specific. The setting only affects the socket close.	The linger time-out in seconds; the default is **-1** seconds, meaning that the **SO_LINGER** option has been disabled.
SoTimeout	Allows you to set or get the client's **SO_TIMEOUT** value, in milliseconds.	The **SO_TIMEOUT** value in milliseconds; the default is **10,000** milliseconds (10 seconds).
TcpNoDelay	Specifies whether the client's **TCP_NODELAY** option (that is, Nagle's algorithm) is enabled or disabled.	**True** enables the option, and **False** disables it.
Socket Factory Implementation Class Name	Enter the name of the Java class that implements the client socket factory. This class is used to create a client socket. If you have provided your own client socket implementation, you must enter the name of the Java class that contains this implementation. The factory implementation class must implement the following interface: **com.stc.connector.tcpip.model.factory.TCPIPSocketFactory**	A valid Java class name; the default is: **com.stc.connector.tcpip.model.factory.TCPIPSocketFactoryImpl**

3.3.3 TCPIP Outbound Settings - Connection Establishment

Specifies configuration parameters that are used to control the connection establishment. The **TCPIP Outbound Settings - Client Connection Establishment** properties contain the top-level parameters displayed in Table 13.

Note: *This section is only used when the Connection Type is Client.*

Table 13 TCPIP Outbound Settings - Client Connection Establishment Properties

Name	Description	Required Value
Time To Wait Before Attempting Connection	Specifies the length of time (in milliseconds) the eWay waits before attempting to connect to the external system.	An integer indicating the length of time (in milliseconds) the eWay waits before attempts to connect. The configured default is **0**.
Always Create New Connection	Specifies whether the eWay always attempts to create a new connection when a connection establishment request is received. ▪ **True** indicates that the eWay always attempts to create a new connection without attempting to match an existing connection. ▪ **False** indicates that the eWay attempts to match an existing connection (managed by the container).	**True** or **False**. The configured default is **False**.
Auto Reconnect Upon Matching Failure	Specifies whether to attempt to re-connect automatically when the eWay gets a matching connection from a container, even though this connection is not valid due to various reasons: for example, the external side of the connection is closed/reset due to the external application's logic. This property only takes effect when the Integration Server has an existing connection in its connection pool, not during an initial triggering when the pool is empty. ▪ **True** indicates that the eWay discards the invalid matching connection and automatically attempts to reconnect using a new connection. ▪ **False** indicates that the eWay does not automatically attempt to reconnect using a new connection: instead, a exception is thrown and the eWay raises the appropriate alert. The user must detect this type of failure and act appropriately.	**True** or **False**. The configured default is **True**.

Table 13 TCPIP Outbound Settings - Client Connection Establishment Properties

Name	Description	Required Value
Max Connection Retry	Specifies the maximum number of times the eWay attempts to connect to a specific external TCP/IP destination (host/port) before giving up.	An integer indicating the number of times the eWay will attempt to connect.
Retry Connection Interval	Specifies the length of time (in milliseconds) the eWay waits between attempts to connect to a specific external TCP/IP destination (host/port).	An integer indicating the length of time (in milliseconds) the eWay waits between attempts to connect. The configured default is **30000** (or 30 seconds).

3.3.4 TCPIP Outbound Settings - Server Port Binding

Specifies configuration parameters used for controlling server port binding. This parameter is only used when the **Connection Type** is set as **Server**. The **TCPIP Outbound Settings - Server Port Binding** properties contain the top-level parameters displayed in Table 14.

Table 14 TCPIP Outbound Settings - Server Port Binding Properties

Name	Description	Required Value
Max Binding Retry	Specifies the maximum number of times the eWay attempts to bind to the specified TCP/IP port on the localhost before giving up.	A number indicating the number of times the eWay attempts to bind to the specified TCP/IP port on the localhost.
Retry Binding Interval	Specifies the number of milliseconds the eWay waits between attempts to bind to the specified TCP/IP port on the localhost.	A number indicating the length of times in milliseconds, between attempts to bind to the specified TCP/IP port. The configured default is **30000** (30 seconds).

3.3.5 Envelope Message

These properties are the envelope message format settings for the Outbound eWay. These properties operate in the same way as those for the inbound eWay. See **"Message Envelope Types" on page 33** for more information on envelope types.

This section explains the envelope message format properties for the server. These properties are all associated with TCP/IP enveloping. The **Envelope Message** section of the Inbound TCP/IP eWay Connectivity Map properties contains the top-level parameters displayed in Table 15.

Table 15 Connectivity Map - Envelope Message Properties

Name	Description	Required Value
Envelope Type	Specifies the envelope type. The envelope type defines where a message starts and stops. See **"Message Envelope Types" on page 33** for a description of the available envelope types and the structure of each. **BeginEndMarked** is supported by the properties **"Bytes to Read" on page 44**, **"Ignore Until Char Value" on page 45**, and **"Store Until Char Value" on page 45**. **EndMarked** is supported by the property **"Store Until Char Value" on page 45**. **FixedLength** is supported by the properties **"Bytes to Read" on page 44**. **LengthPrefixed** is supported by the property **"Width of Length" on page 44** and **"Numeric Representation" on page 45**. **MarkedAndFixed** is supported by the property **"Bytes to Read" on page 44**, **"Ignore Until Char Value" on page 45**, and **"Store Until Char Value" on page 45** **Custom** is supported by the properties **"Custom Enveloped Class Name" on page 44** and **"Customer Defined Property" on page 44**. For optimum performance, use the method **receiveEnvelopedMsg()** with any enveloped messages. This method uses the envelope as its ending condition, while the other receiving methods, **receiveBytes()** and **receiveString()**, use a time-out as their ending condition. ***Note:*** *For all envelope types, except **MarkedAndFixed**, the data is just the payload. See* **"MarkedAndFixed" on page 34** *for an explanation of how the data is handled by that envelope type.*	Enter one of the following properties denoting the envelope type: ♦ **BeginEndMarked** ♦ **EndMarked** ♦ **FixedLength** ♦ **LengthPrefixed** ♦ **MarkedAndFixed** ♦ **PerActiveConnection** ♦ **Custom** The default is **BeginEndMarked**.

Table 15 Connectivity Map - Envelope Message Properties (Continued)

Name	Description	Required Value
Custom Enveloped Class Name	Specifies the Java class name to be used when the **Envelope Type** property is set to **Custom**. If you are using a custom envelope you have created, using a Java class, you can import the Java JAR file containing the class into any desired Collaboration, using the Collaboration Editor's file import feature. The class name should be a full qualified class name, such as "com.abc.MyClass". The class must implement interfaces **com.stc.connector.tcpip.ext.msg.EnvelopedMsgReceiver** and **com.stc.connector.tcpip.ext.msg.EnvelopedMsgSender**. For more details, see **Customized Enveloping** on page 74.	A full Java class name. A full qualified class name, or **None** if Custom is not the Envelope Type. The configured default is **None**.
Customer Defined Property	Used when the Envelope Type value is set to **Custom**. Specifies a list of user-defined parameters. You can parse this information, such as delimiters, into your customized envelope message implementation.	A text String.
Bytes to Read	Used with Envelope Types **FixedLength** and **MarkedAndFixed**. Specifies the number of bytes to read. It is assumed that all Events received by the eWay have the same length.	An integer indicting the number of bytes. The configured default is **1**.
Width of Length	Used for Envelope Type value **LengthPrefixed**. Specifies the width of the envelope length. In other words, it dictates the number of digits to be used to represent the Length field.	An integer; the range is 1 to 10. This property must be set to **2** for **Network short** and **4** for **Network long**. The configured default is **1**.

Table 15 Connectivity Map - Envelope Message Properties (Continued)

Name	Description	Required Value
Numeric Representation	Used for Envelope Type value **LengthPrefixed**. Specifies how the number representation of the prefixed length is expressed. This value is expressed in one of the following formats: decimal, hexadecimal, octal, network short, or network long.	Select one of the following**:** ▪ **Decimal** ▪ **Hexadecimal** ▪ **Octal** ▪ **Network short** ▪ **Network long** The configured default is **Decimal**.
Ignore Until Char Value	Used for the Envelope Types **BeginEndMarked** and **MarkedAndFixed**. Specifies the value for the ignore-until (same as begin block) character. All incoming characters are ignored until this character is encountered.	A decimal ascii number. The allowed range is 1 to 127. The configured default is **11**.
Store Until Char Value	Used for Envelope Types **BeginEndMarked**, **EndMarked**, and **MarkedAndFixed**. Specifies the character in the End Block or Marker position of the envelope. All incoming characters are stored until this character is encountered.	A decimal ascii number. The allowed range is 1 to 127. The configured default is **12**.

3.4 TCP/IP eWay Environment Properties

The TCP/IP eWay Environment properties are organized into the following subfolders:

- **TCPIP Server (inbound) eWay - General Inbound Settings** on page 46
- **TCPIP Server (inbound) eWay - TCPIP Inbound Settings** on page 47
- **TCPIP Server (inbound) eWay - MDB Pool Settings** on page 48
- **TCPIP Client (outbound) eWay - General Outbound Settings** on page 50
- **TCPIP Client (outbound) eWay - TCPIP Outbound Settings** on page 51
- **TCPIP Client (outbound) eWay - Connection Pool Settings** on page 52

3.4.1 TCPIP Server (inbound) eWay - General Inbound Settings

The **General Inbound Settings** properties represents general TCPIP inbound configuration information. The **TCPIPServer (inbound) eWay - General Inbound Settings** properties contain the top-level parameters displayed in Table 16.

Table 16 TCPIPServer (inbound) eWay - General Inbound Settings Properties

Name	Description	Required Value
Persistence State File Location	Specifies the directory location (a local folder name) where the state files, used to persist the state value, are stored. This property is required when the **Scope Of State** is set to **Persistence**.	The file and path. The default value is **C:/temp/ tcpipinbound/state**.

3.4.2 TCPIP Server (inbound) eWay - TCPIP Inbound Settings

Specifies the Java Socket and ServerSocket options. For more information, refer to the JDK Javadoc.

The **TCPIPServer (inbound) eWay - TCPIP Inbound Settings** properties contain the top-level parameters displayed in Table 17.

Table 17 TCPIP Server (inbound) eWay - MDB Properties

Name	Description	Required Value
Host	Specifies the host name or IP address used to establish a TCPIP connection. This parameter is only used when Connection Type is Client.	A TCP/IP host name or IP address.
ServerPort	Specifies the port number of the TCP/IP destination. This is dependent upon the specified Connection Type. If the value for Connection Type is: ▪ **Server**: the ServerPort value is set to the port number on the local host. ▪ **Client**: the ServerPort value is set to the port number of the external host.	The port number of the TCP/IP destination. The default is **8888**.
Backlog	Specifies the maximum length of the queue when creating the ServerSocket. The maximum queue length for incoming connection indications (a request to connect) is set to the backlog parameter. If a connection indication arrives when the queue is full, the connection is refused. ***Note:*** This parameter is only used when **Connection Type** is set to **Server**.	An integer indicting the queue length for incoming connections. The configured default value is **50**.

3.4.3 TCPIP Server (inbound) eWay - MDB Pool Settings

Specific to the MDB bean pool of Sun Java™ System Application Server and Sun SeeBeyond eGate Integration Server only. The parameter settings in this section are applied to sun-ejb-jar.xml.

The **TCPIPServer (inbound) eWay - MDB Pool Settings** properties contain the top-level parameters displayed in Table 18.

Table 18 TCPIPServer (inbound) eWay - MDB Pool Settings Properties

Name	Description	Required Value
Steady Pool Size	Specifies the minimum number of MDB beans to be maintained. When the value is set to a value that is greater than 0, the container not only pre-populates the MDB bean pool with the specified number, but also attempts to ensure that there are always this many MDB beans in the free pool. This ensures that there are enough MDB beans in the "ready to serve" state to process user requests. This parameter does not guarantee that more than the "steady-pool-size" MDB instances will not exist at a given time. It only governs the number of instances that are pooled over a long period of time. For example, suppose an idle stateless session container has a fully-populated pool with a steady-pool-size of 10. If 20 concurrent requests arrive for the MDB bean component, the container creates 10 additional instances to satisfy the burst of requests. This prevents the container from blocking any of the incoming requests. However, if the activity dies down to 10 or fewer concurrent requests, the additional 10 instances are discarded.	An integer indicating the minimum number of Message Driven Beans (MDBs) to be maintained. The configured default is **10**.
Max Pool Size	Specifies the maximum number of MDB beans in the pool.	An integer indicating the maximum number of MDB beans in the pool. A value of **0** means the pool is unbounded. The configured default is **60**.

Table 18 TCPIPServer (inbound) eWay - MDB Pool Settings Properties (Continued)

Name	Description	Required Value
Pool Idle Timeout In Seconds	Specifies the interval at which the "*remove expired MDBs*" thread runs. This thread periodically removes unused MDB beans with expired timeouts. This provides a hint to the server, and allows the user to specify the maximum amount of time that an MDB bean instance can remain idle in the pool. After this period of time, the pool can remove this bean. When **Pool Idle Timeout In Seconds** is set to a value greater than **0**, the removes or destroys any MDB bean instance that is idle for this specified duration. A value of **0** specifies that idle MDB beans can remain in the pool indefinitely.	An integer indicting the Pool Idle Timeout in seconds for unused MDBs. A value of **0** specifies that idle MDBs can remain in the pool indefinitely. The configured default value is **600**.

3.4.4 TCPIP Client (outbound) eWay - General Outbound Settings

The **General Outbound Settings** properties represents general TCPIP outbound configuration information. The **TCPIPClient (outbound) eWay - General Outbound Settings** properties contain the top-level parameters displayed in Table 19.

Table 19 TCPIPClient (outbound) eWay - General Outbound Settings Properties

Name	Description	Required Value
Persistence State File Location	Specifies the File Location (a local folder name). This property is required when the **Scope Of State** value is set to is **Persistence**. This file is used to store the state files which are used to persist the state value.	The file and path. The default value is **/temp/tcpipoutbound/state.**

3.4.5 TCPIP Client (outbound) eWay - TCPIP Outbound Settings

The **TCPIP Outbound Settings** properties represents general TCPIP outbound configuration information. The **TCPIPClient (outbound) eWay - TCPIP Outbound Settings** properties contain the top-level parameters displayed in Table 20.

Table 20 TCPIPClient (outbound) eWay - TCPIP Outbound Settings Properties

Name	Description	Required Value
Host	Specifies the host name or IP address used to establish a TCPIP connection. This parameter is only used when the **Connection Type** is set to **Client**.	A TCP/IP host name or IP address. The configured default is **localhost**.
ServerPort	Specifies the port number of the TCP/IP destination. This is dependent upon the specified Connection Type. If the value for Connection Type is: ▪ **Server**: the ServerPort value is set to the port number on the local host. ▪ **Client**: the ServerPort value is set to the port number of the external host.	The port number of the TCP/IP destination. The default is **7777**.
Backlog	Specifies the maximum length of the queue when creating the ServerSocket. The maximum queue length for incoming connection indications (a request to connect) is set to the backlog parameter. If a connection indication arrives when the queue is full, the connection is refused. ***Note:*** This parameter is only used when **Connection Type** is set to **Server**.	An integer indicting the queue length for incoming connections. The configured default value is **50**.

3.4.6 TCPIP Client (outbound) eWay - Connection Pool Settings

Specific to the RA connection pool of Sun Java™ System Application Server or the Sun SeeBeyond eGate Integration Server only. The parameter settings in this section are applied to sun-ra.xml.

The **TCPIPClient (outbound) eWay - Connection Pool Settings** properties contain the top-level parameters displayed in Table 21.

Table 21 TCPIPClient (outbound) eWay - Connection Pool Settings Properties

Name	Description	Required Value
Steady Pool Size	Specifies the minimum number of RA connections to be maintained. When the value is set to a value that is greater than 0, the container not only pre-populates the RA connection pool with the specified number, but also attempts to ensure that there are always this many RA connections in the free pool. This ensures that there are enough RA connections in the “ready to serve” state to process user requests. For example, suppose an idle stateless session container has a fully-populated pool with a steady-pool-size of 10. If 20 concurrent requests arrive for the RA Connection component, the container creates 10 additional instances to satisfy the burst of requests. This prevents the container from blocking any of the incoming requests. However, if the activity dies down to 10 or fewer concurrent requests, the additional 10 instances are discarded.	An integer indicating the minimum number of RA connections to be maintained. The configured default is **1**.
Max Pool Size	Specifies the maximum number of RA connections in the pool.	An integer indicating the maximum number of RA connections in the pool. A value of **0** indicates that the pool is unbounded. The configured default is **32**.

Table 21 TCPIPClient (outbound) eWay - Connection Pool Settings Properties (Continued)

Name	Description	Required Value
Pool Idle Timeout In Seconds	Specifies the interval at which the "*remove expired RA connections*" thread runs. This thread periodically removes unused RA connections with expired timeouts. This provides a hint to the server, and allows you to specify the maximum amount of time that an RA connection instance can remain idle in the pool. After this period of time, the pool can remove this bean. When **Pool Idle Timeout In Seconds** is set to a value greater than **0**, the container removes or destroys any RA connection instance that is idle for this specified duration. A value of **0** specifies that idle RA connections can remain in the pool indefinitely.	An integer indicting the Pool Idle Timeout in seconds for unused RA connections. A value of **0** specifies that idle RA connections can remain in the pool indefinitely. The configured default value is **600**.

Chapter 4

Implementing the Sample Project

This chapter describes how to implement the TCP/IP eWay by way of reviewing sample Project included with the eWay.

For a complete explanation of eGate terminology and concepts, including how to use the Sun SeeBeyond eGate™ Enterprise Designer to create and set properties for the components of an eGate Project, see the *Sun SeeBeyond eGate™ Integrator User's Guide*.

What's in This Chapter

- **TCP/IP eWay Components** on page 54
- **Importing a Sample Project** on page 55
- **TCP/IP Sample Project Overview** on page 56
- **Creating the prjTCPIPSample Project** on page 57
- **Customized Enveloping** on page 74

4.1 TCP/IP eWay Implementation

This chapter presents sample TCP/IP eWay Projects that use Collaboration Definitions (Java) to implement the Business Logic. These Projects are created using the same procedures as the sample end-to-end Project provided in the *Sun SeeBeyond eGate™ Tutorial*.

4.1.1 TCP/IP eWay Components

TCP/IP eWay components that are unique to this eWay include the following:

TCP/IP eWay Properties File

The Properties file for the TCP/IP eWay contains the parameters that are used to connect with a specific external system. These parameters are set using the **Properties Editor**. For more information about the TCP/IP eWay Configuration File and the **Properties Editor** see **"Configuring the TCP/IP eWay" on page 16**.

TCPIP.TCPIPClient OTD

The TCPIP.TCPIPClient OTD contains methods and attributes used to create the Business Rules that invoke TCP/IP protocol. The TCP/IP eWay OTD provides the following features:

- **Java Socket Functions**: Allows you to expose a Java socket object via J2EE connection management, using whatever the **java.net.Socket** interface provides.
- **Predefined Message Envelope Types**: Several types of enveloped messages (over a TCP/IP connection) are provided with the eWay.eWay. See **"Message Envelope Types" on page 33** for details.
- **Extensibility for Enveloped Messages**: If the predefined message envelope types (over a TCP/IP connection) cannot meet your demands, you can provide your own customized enveloping. The eWay has corresponding defined interfaces for this purpose. See **"Customized Enveloping" on page 74** for details. Also, a sample customized envelope is provided.
- **Socket Property Options**: You can set these properties (for example, time-out and buffer size) statically using the eWay **Properties** dialog box or dynamically using methods in an OTD.
- **Message Property Options**: You can set these properties (for example, envelope type and beginning marker) statically using the eWay **Properties** dialog box or dynamically via methods in an OTD.
- **Single-port Connections**: The TCP/IP server OTD can handle multiple, virtually unlimited connections asynchronously, on a single port. Each work is a separate thread.
- **Dedicated Session Mode**: This feature, when enabled in a server, allows the current client's request to exclusively hold the server port that it connects to. See **"Dedicated Session Mode" on page 20** for details.
- **Server Endpoint Property Options**: You can set the eWay's server socket and message properties as desired.

4.2 Importing a Sample Project

A Sample eWay Project is included with the TCPIP eWay documentation file. To import a sample eWay Project to the Enterprise Designer do the following:

1. The sample files are uploaded with the eWay's documentation SAR file and downloaded from the Sun Java™ Composite Application Platform Suite Installer's Documentation tab. The **TCP_IP_eWay_Sample.zip** file contains the sample Project zip file. Extract the samples to a local file.
2. Save all unsaved work before importing a Project.
3. From the Enterprise Designer's Project Explorer pane, right-click the Repository and select **Import** from the shortcut menu. The **Import Manager** appears.

4. Browse to the directory that contains the sample Project zip file. Select the sample file (for this sample, **prjTCPIPSample.zip**) and click **Import**. After the sample Project is successfully imported, click **Close**.
5. Before an imported sample Project can be run you must do the following:
   - Create an **Environment** (see **"Creating the Project's Environment" on page 69**)
   - Configure the eWays for your specific system (see **"Configuring the eWays" on page 70**)
   - Create a **Deployment Profile** (see **"Creating the Deployment Profile" on page 72**)
   - Create and start a domain (see **"Creating and Starting the Domain" on page 72**)
   - Build and deploy the Project (see **"Building and Deploying the Project" on page 73**)

4.3 TCP/IP Sample Project Overview

The TCP/IP eWay sample Project demonstrates how the TCP/IP eWay processes information to and from a client, as well as how it processes information from a server. The resulting information is then written to a text file.

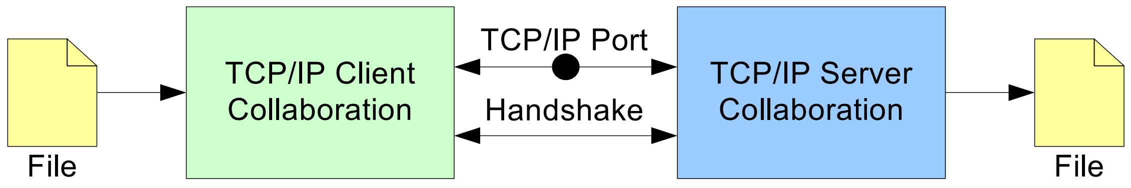
This section provides an overview of what the sample Project demonstrates, operating in both client and server mode.

The Sample Project demonstrates the following:

- The client and server do a handshake by exchanging a pre-determined, formatted String.
- After a successful handshake, the client reads data from a file and sends it to the server.
- The server reads the data and writes it back to a file (see **Figure 5 on page 56**).

Both the client and server must be configured for the same TCP/IP port. The TCP/IP port is configured from the Environment properties.

Figure 5 TCP/IP Sample Project Scenario



4.4 Creating the prjTCPIPSample Project

The following pages provide step by step directions for manually creating the **prjTCPIPSample** Project.

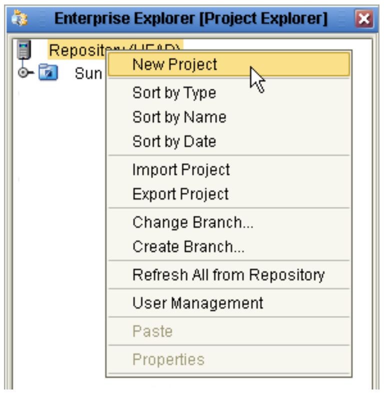
4.4.1 Creating a Project

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

1 Start the Enterprise Designer.

2 From the Enterprise Explorer's Project Explorer tab, right-click the Repository and select **New Project** (see Figure 6). A new Project (Project1) appears on the Project Explorer tree.

Figure 6 Enterprise Explorer - New Project



3 Rename the Project (for this sample, **prjTCPIPSample**).

4.4.2 Creating the Java Collaboration Definitions

The next step in the sample is to create the Java Collaboration Definitions using the **Collaboration Definition Wizard (Java)**. Once the Collaborations have been created, the Business Rules of the Collaborations are written using the Collaboration Editor. The prjTCPIPSample sample Project includes two Java Collaboration Definitions:

- **jcdTCPClient**
- **jcdTCPServer**

Creating the jcdTCPClient Collaboration

The **jcdTCPClient** Collaboration defines transactions between the inbound File eWay and the TCP/IP eWay.

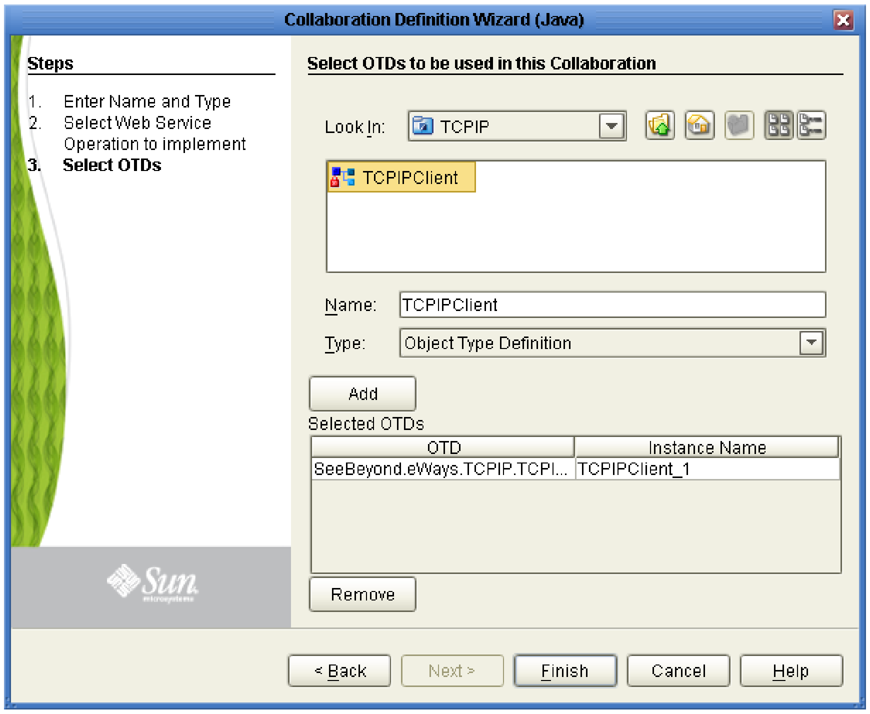
1 From the Project Explorer, right-click the **prjTCPIPSample** Project and select **New** > **Collaboration Definition (Java)** from the shortcut menu. The **Collaboration Definition Wizard** appears.

2 Enter a Collaboration name (for this sample **jcdTCPClient**) 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 Name field now displays **receive**. Click **Next**.

4 For Step 3 of the wizard, double-click **Sun SeeBeyond > eWays > TCPIP > TCPIPClient.** The **TCPIPClient_1** OTD is added to the Selected OTDs.

Figure 7 Collaboration Definition Wizard (Java) - Select OTD


Collaboration Definition Wizard (Java)
Steps
1. Enter Name and Type
2. Select Web Service Operation to implement
3. Select OTDs
Select OTDs to be used in this Collaboration
Add
Remove
< Back
Next >
Finish
Cancel
Help

Look In: TCPIP

TCPIPClient

Name: TCPIPClient

Type: Object Type Definition

Selected OTDs

OTD	Instance Name
SeeBeyond.eWays.TCPIP.TCPI...	TCPIPClient_1

5 Click **Finish**. The Collaboration Editor (Java) with the new **jcdTCPClient** Collaboration appears in the right pane of the Enterprise Designer.

Creating the jcdTCPServer Java Collaboration

The **jcdTCPServer** Collaboration (Java) defines transactions made between the inbound TCP/IP eWay and the outbound File eWay.

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

2 Enter a Collaboration Definition name (for this sample **jcdTCPServer**) and click **Next**.

3 For Step 2 of the wizard, from the Web Services Interfaces selection window, double-click **Sun SeeBeyond** > **eWays** > **TCPIP** > **TCPIPClient** > **receive**. The Name field now displays **receive.** Click **Next**.

4 For Step 3, **Select OTDs**, from the Select OTDs selection window, double-click **Sun SeeBeyond > eWays > File > FileClient**. The Selected OTDs field now lists the **FileClient_1** OTD.

5 Click **Finish**. The Collaboration Editor with the new **jcdTCPServer** Collaboration appears.

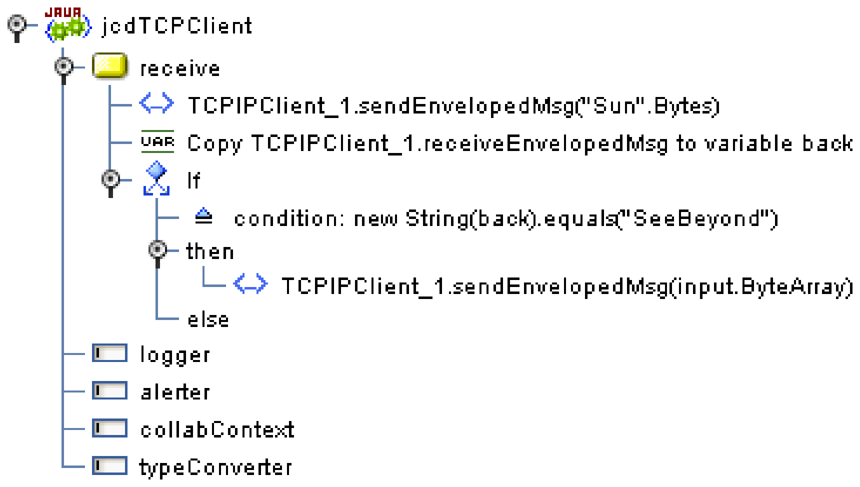
4.4.3 Creating the Business Rules

The next step in the sample is to create the Business Rules of the Collaborations using the Collaboration Editor (Java). The Business Rules for the jcdTCPClient and jcdTCPServer Collaborations are created using the Collaboration Editor (Java).

Create the jcdTCPClient Collaboration Business Rules

The jcdTCPClient Collaboration contains the Business Rules displayed in Figure 8.

Figure 8 jcdTCPClient Collaboration Business Rules



```
jcdTCPClient
  receive
    TCPIPClient_1.sendEnvelopedMsg("Sun".Bytes)
    Copy TCPIPClient_1.receiveEnvelopedMsg to variable back
    If
      condition: new String(back).equals("SeeBeyond")
      then
        TCPIPClient_1.sendEnvelopedMsg(input.ByteArray)
      else
  logger
  alerter
  collabContext
  typeConverter
```

To create the **jcdTCPClient** Collaboration Business Rules do the following:

1 From the Project Explorer tree, double-click **jcdTCPClient** to open the Collaboration Editor (Java) to the **jcdTCPClient** Collaboration.

2 To create comments for the Business Rules, click the comment icon on the Business Rules toolbar. The **Enter a Comment** dialog box appears. Enter the comment and click **OK**. The comment is placed on the Business Rules tree under the last selected item. Once the Comment is created, it can be moved by clicking the comment and dragging it up or down the Business Rules tree to a new location.

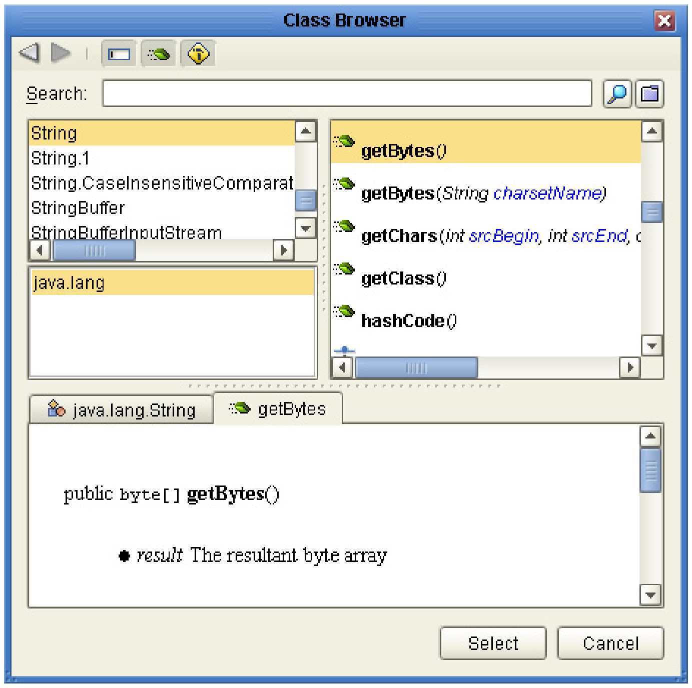
3 Create the **TCPIPClient_1.sendEnvelopedMsg("Sun".Bytes)** Business Rule

A Right-click **TCPIPClient_1** in the left pane of the Business Rules Designer, and click **Select method to call** from the shortcut menu.

B From the method selection window, select **sendEnvelopedMsg(byte[] arg0)**. The **sendEnvelopedMsg** method box appears.

C From the Business Rules Designer toolbar, click the **Class Browser** icon. The **Class Browser** dialog box appears (see Figure 9).

Figure 9 Class Browser



public byte[] getBytes()

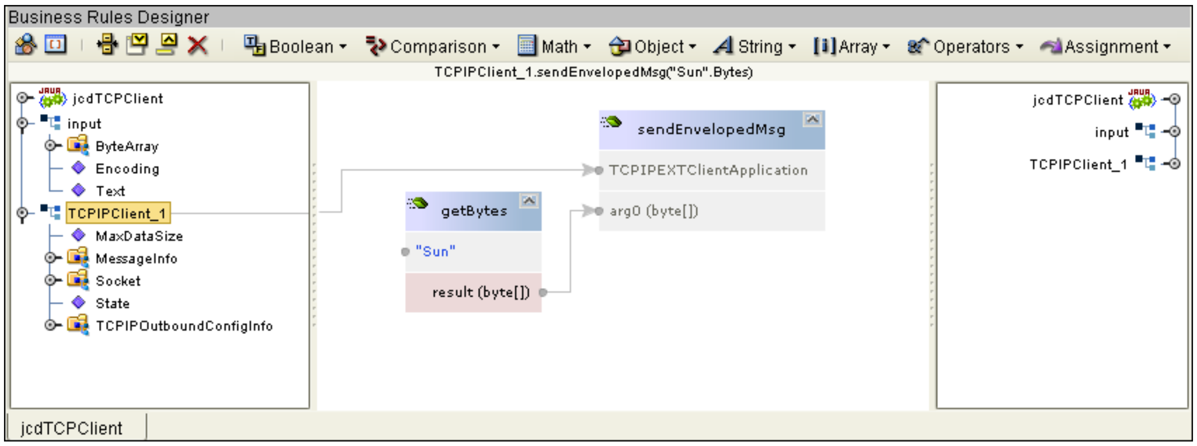
- *result* The resultant byte array

D From the **Class Browser** dialog box, select **String** as the Class, and select **getBytes()** in the right pane, as the method. Click **Select** to close the Class Browser. The **getBytes** method box appears.

E Double click the **String** node of the **getBytes** method box to change the String value. Enter **Sun** as the value.

F Map the **result(byte[])** output node of the **getBytes** method box, to **arg0 (byte[])** input node of the **sendEnvelopedMsg** method box. To do this, click on the **result(byte[])** output node of the **getBytes** method box, and drag your cursor to the input node of the **sendEnvelopedMsg** method box. A link now connects the two nodes (see Figure 10 on page 60).

Figure 10 Collaboration Editor - Business Rules Designer

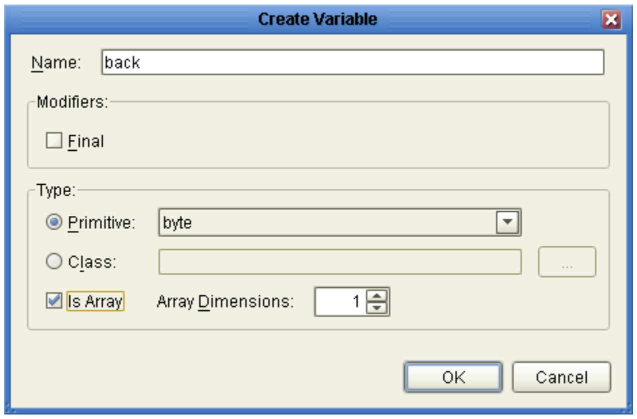


4 Create the **Copy TCPIPClient_1.receiveEnvelopedMsg to variable back** variable:

A From the Business Rules toolbar, click the **Local Variable** icon. The **Create Variable** dialog box appears.

B From the Create Variable dialog box, enter **back** as the name, and for Type, select **Primitive** and **byte**. Select the **Is Array** option, and click OK (see Figure 11).

Figure 11 Create Variable dialog box

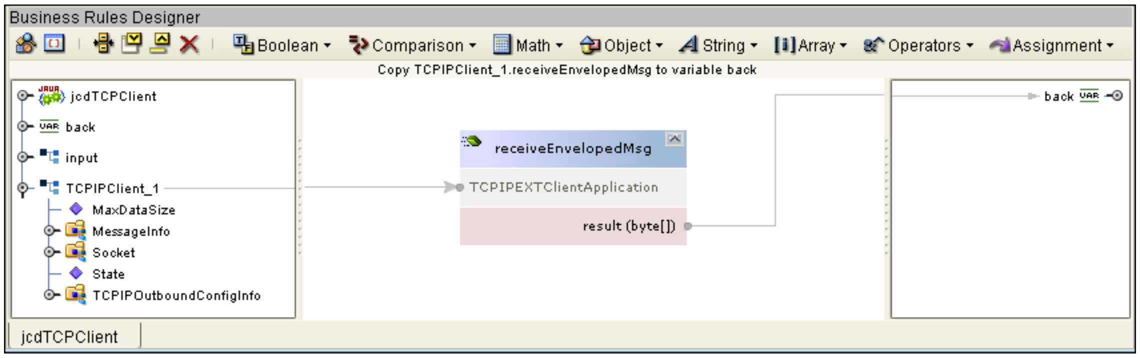


A Right-click **TCPIPClient_1** in the left pane of the Business Rules Designer, and click **Select method to call** from the shortcut menu.

B From the method selection window, select **receiveEnvelopedMsg()**. The **receiveEnvelopedMsg** method box appears.

C Map the **result(byte[])** output node of the **receiveEnvelopedMsg** method box, to **back** (variable) in the right pane of the Business Rules Designer (see **Figure 12 on page 61**).

Figure 12 Collaboration Editor - Business Rules Designer



5 Create the **If-then** statement and **condition: new String(back).equals("SeeBeyond")**:

A From the Business Rules toolbar, click the **If-then** icon. An **If** statement is added to the Business Rules tree.

B Select **condition:** under the **If** statement.

C From the Business Rules Designer toolbar, click the **Class Browser** icon. The **Class Browser** dialog box appears.

D From the **Class Browser** dialog box, select **String** as the Class, and select ***String(byte[] bytes)*** in the right pane as the constructor (a constructor is marked by the yellow diamond-hammer symbol). Click **Select** to close the Class Browser. The **String** constructor box appears.

E From the Business Rules Designer's **String** menu, select **Equals**. The **Equals** method box appears.

F From the Business Rules Designer's **String** menu, select **Literal String**. The **String** Literal box appears. Enter **SeeBeyond** as the Literal String value.

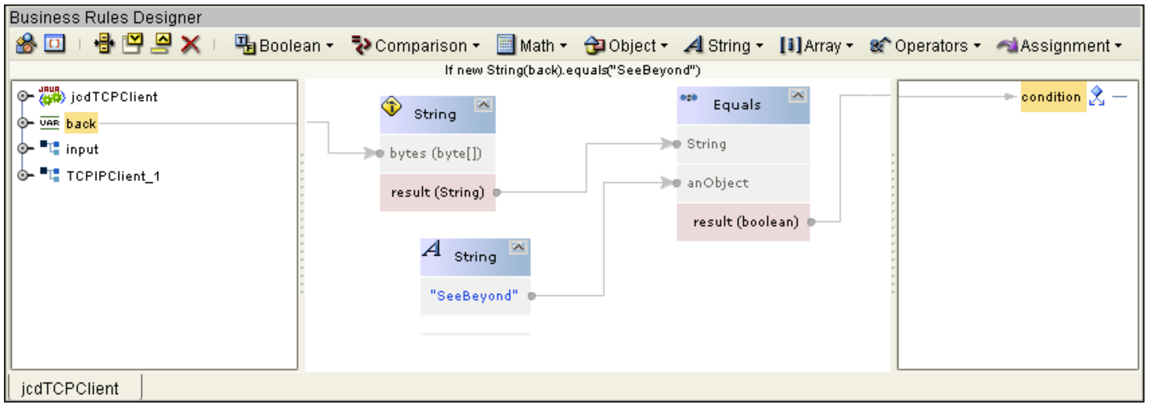
G Map **back** (variable) in the left pane of the Business Rules Designer, to the **bytes (byte[])** input node of the **String** constructor box.

H Map the **result (String)** output node of the **String** construction box, to the **String** input node of the **Equals** method box.

I Map the **"SeeBeyond"** output node of the **String** Literal box, to the **anObject** input node of the **Equals** method box.

J Map the **result (boolean)** output node of the **Equals** method box to **condition** in the right pane of the Business Rules Designer (see **Figure 13 on page 62**).

Figure 13 Collaboration Editor - Business Rules Designer



6 Create the **TCPIPClient_1.sendEnvelopedMsg(input.ByteArray)** Business Rule under the **If/then** statement:

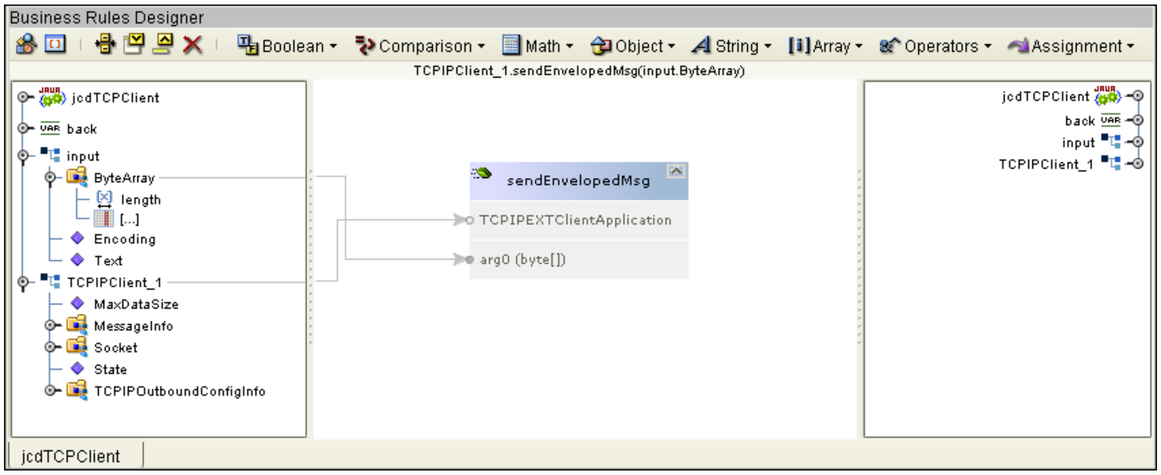
A From the Business Rules tree, select **then** under the **If** statement. From the Business Rules toolbar, click the Rule icon. A new rule is added under the **then** statement.

A Right-click **TCPIPClient_1** in the left pane of the Business Rules Designer, and click **Select method to call** from the shortcut menu.

B From the method selection window, select **sendEnvelopedMsg(byte[] arg0)**. The **sendEnvelopedMsg** method box appears.

C Map **ByteArray** under **input** (OTD) in the left pane of the Business Rules Designer, to the **arg0 (byte[])** input node of the **sendEnvelopedMsg** method box (see Figure 14).

Figure 14 Collaboration Editor - Business Rules Designer



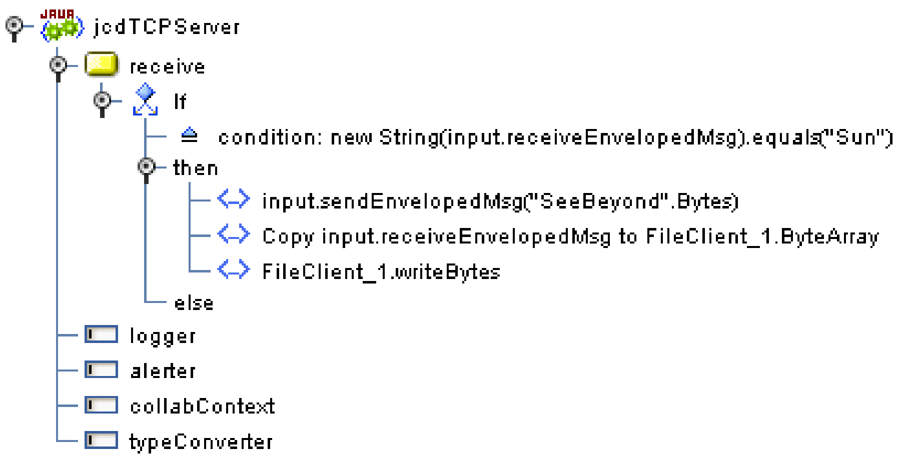
7 From the editor's toolbar, click **Validate** to check the Collaboration for errors.

8 Save your current changes to the repository.

jcdTCPServer Collaboration Business Rules

The **jcdTCPServer** Collaboration contains the Business Rules displayed in Figure 15.

Figure 15 jcdTCPServer Collaboration Business Rules



```
jcdTCPServer
  receive
    If
      condition: new String(input.receiveEnvelopedMsg).equals("Sun")
      then
        input.sendEnvelopedMsg("SeeBeyond".Bytes)
        Copy input.receiveEnvelopedMsg to FileClient_1.ByteArray
        FileClient_1.writeBytes
      else
  logger
  alerter
  collabContext
  typeConverter
```

To create the **jcdTCPServer** Collaboration Business Rules do the following:

1 From the Project Explorer tree, double-click **jcdTCPServer** to open the Collaboration Editor (Java) to the **jcdTCPServer** Collaboration.

2 Create the **If-then** statement and **condition: new String(input.receiveEnvelopedMsg).equals("Sun")**:

A From the Business Rules toolbar, click the **If-then** icon. An **If** statement is added to the Business Rules tree.

B Select **condition:** under the **If** statement.

A Right-click **input** (OTD) in the left pane of the Business Rules Designer, and click **Select method to call** from the shortcut menu.

B From the method selection window, select **receiveEnvelopedMsg()**. The **receiveEnvelopedMsg** method box appears.

C From the Business Rules Designer toolbar, click the **Class Browser** icon. The **Class Browser** dialog box appears.

D From the **Class Browser** dialog box, select **String** as the Class, and select **String*(byte[] bytes)*** in the right pane as the constructor (a constructor is marked by the yellow diamond-hammer symbol). Click **Select** to close the Class Browser. The **String** constructor box appears.

E From the Business Rules Designer's **String** menu, select **Equals**. The **Equals** method box appears.

F From the Business Rules Designer's String menu, select **Literal String**. The **String** Literal box appears. Enter **Sun** as the Literal String value.

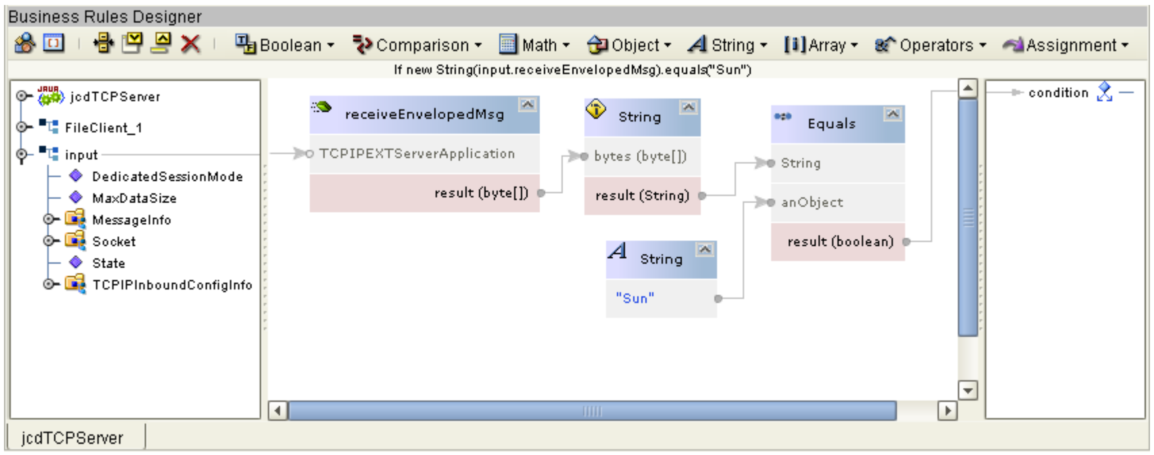
G Map the **result (byte[])** output node of the **receiveEnvelopedMsg** method box, to the **bytes (byte[])** input node of the **String** constructor box.

H Map the **result (String)** output node of the **String** construction box, to the **String** input node of the **Equals** method box.

I Map the **"Sun"** output node of the **String** Literal box, to the **anObject** input node of the **Equals** method box.

J Map the **result (boolean)** output node of the **Equals** method box to **condition** in the right pane of the Business Rules Designer (see Figure 16).

Figure 16 Collaboration Editor - Business Rules Designer



If new String(input.receiveEnvelopedMsg).equals("Sun")

3 Create the **input.sendEnvelopedMsg("SeeBeyond".Bytes)** rule under the If/then statement.

A Select then under the If/then statement. From the Business Rules toolbar, click the Rule icon. A new rule is added to the Business Rules tree.

B Right-click **input** (OTD) in the left pane of the Business Rules Designer, and click **Select method to call** from the shortcut menu.

C From the method selection window, select **sendEnvelopedMsg(byte[] arg0)**. The **sendEnvelopedMsg** method box appears.

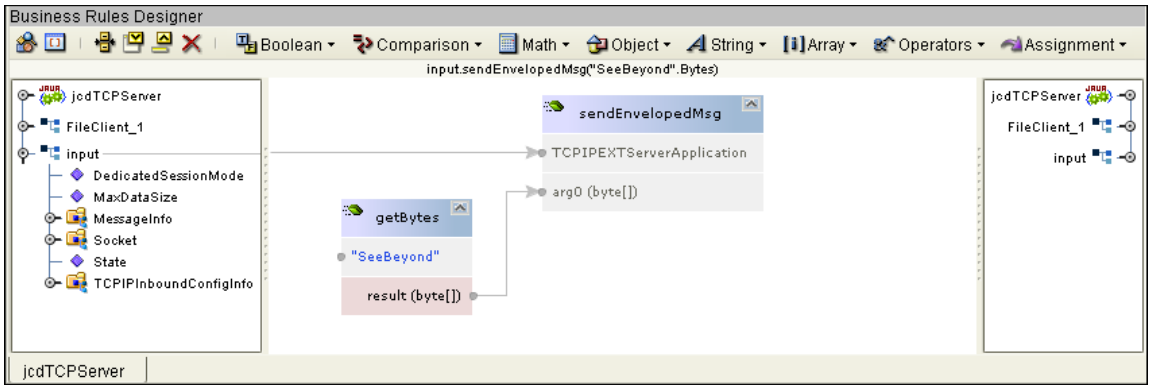
D From the Business Rules Designer toolbar, click the **Class Browser** icon. The **Class Browser** dialog box appears.

E From the **Class Browser** dialog box, select **String** as the Class, and select **getBytes()** in the right pane, as the method. Click **Select** to close the Class Browser. The **getBytes** method box appears.

F Double click the **String** node of the **getBytes** method box to change the String value. Enter **SeeBeyond** as the value.

G Map the **result (byte[])** output node of the **getBytes** method box, to **arg0 (byte[])** input node of the **sendEnvelopedMsg** method box (see **Figure 17 on page 65**).

Figure 17 Collaboration Editor - Business Rules Designer



input.sendEnvelopedMsg("SeeBeyond".Bytes)

"SeeBeyond"

4 Create the **Copy input.receiveEnvelopedMsg to FileClient_1.ByteArray** rule under the If/then statement.

A From the Business Rules toolbar, click the Rule icon. A new rule is added to the Business Rules tree.

B Right-click **input** (OTD) in the left pane of the Business Rules Designer, and click **Select method to call** from the shortcut menu.

C From the method selection window, select **receiveEnvelopedMsg()**. The **receiveEnvelopedMsg** method box appears.

D Map the **result (bytes)** output node of the **receiveEnvelopedMsg** method box, to **ByteArray** under **FileClient_1** in the right pane of the Business Rules Designer.

5 Create the **FileClient_1.writeBytes** rule under the **If/then** statement:

A From the Business Rules toolbar, click the Rule icon. A new rule is added under the **then** statement.

A Right-click **FileClient_1** in the left pane of the Business Rules Designer, and click **Select method to call** from the shortcut menu.

B From the method selection window, select **writeBytes()**. The **writeBytes** method box appears.

6 From the editor's toolbar, click **Validate** to check the Collaboration for errors.

7 Save your current changes to the repository.

For more information on using the Collaboration Editor, see the *Sun Seebeyond eGate™ Integrator User's Guide*.

4.4.4 Creating a Connectivity Maps

The Connectivity Map provides a canvas for assembling and configuring a Project's components. The prjTCPIPSample Project includes two Connectivity Maps.

To create the Connectivity Maps, do the following:

1 From the Project Explorer tree, right-click the new **prjTCPIPSample** Project and select **New > Connectivity Map** from the shortcut menu.

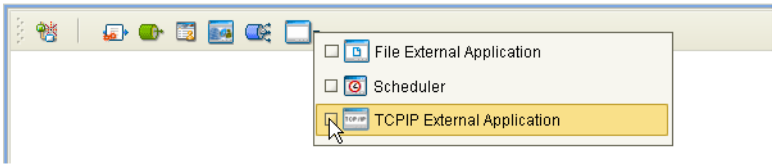
2 The New Connectivity Map appears and a node for the Connectivity Map is added under the Project on the Project Explorer tree labeled **CMap1**. Rename the Connectivity Map **cmTCPClient**.

3 Repeat this process to create a second Connectivity Map named **cmTCPServer**.

Selecting the External Applications

In the Connectivity Maps, eWays are associated with External Systems. For example, to establish a connection to an external TCP/IP application, you must first select TCP/IP as an External Application to use in your Connectivity Map (see Figure 18).

Figure 18 Connectivity Map - External Applications



1 Click the **External Application** icon on the Connectivity Map toolbar,

2 Select the External Applications you require to create your Project (for this sample, **File** and **TCP/IP**). Icons representing the selected External Applications are added to the Connectivity Map toolbar.

Populating the Connectivity Maps

For the **cmTCPClient** Connectivity Map, add the Project components by dragging the icons from the Connectivity Map Editor toolbar to the Connectivity Map canvas.

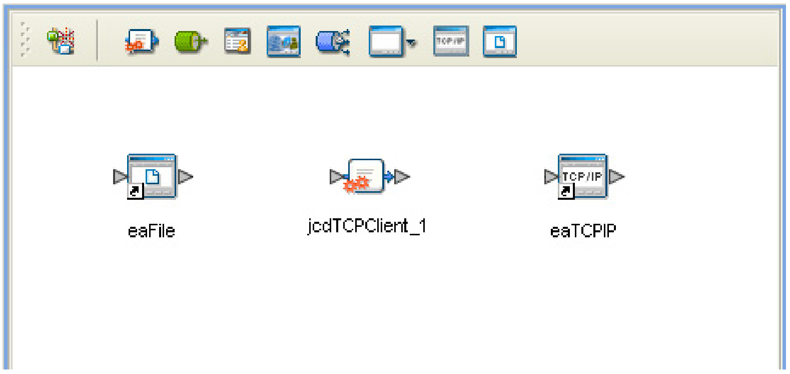
1 Populate the Connectivity Map with the following components as displayed in **Figure 19 on page 67**:

- File External Application
- TCP/IP External Application
- Service (A service is a container for Java Collaborations, Business Processes, eTL processes, and so forth)

2 Rename the components as displayed in Figure 19:

- **File1** to **eaFile**
- **TCPIP1** to **eaTCPIP**
- **cmTCPClient_Service1** to **jcdTCPClient_1**

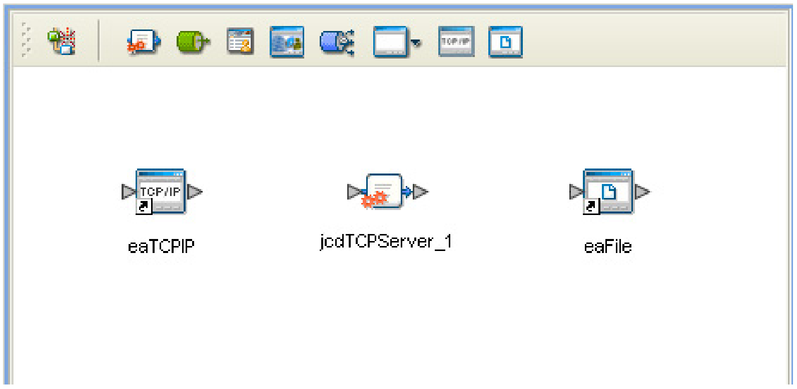
Figure 19 cmTCPClient Connectivity Map with Components



3 Follow these same procedures to create a second Connectivity Map named cmTCPServer containing the following components (as displayed in Figure 20):

- File External Applications (rename as **eaFile**)
- Service (rename as **jcdTCPServer_1**)
- TCP/IP External Application (rename as **eaTCPIP**)

Figure 20 cmTCPServer Connectivity Map with Components



4 Save your current changes to the Repository.

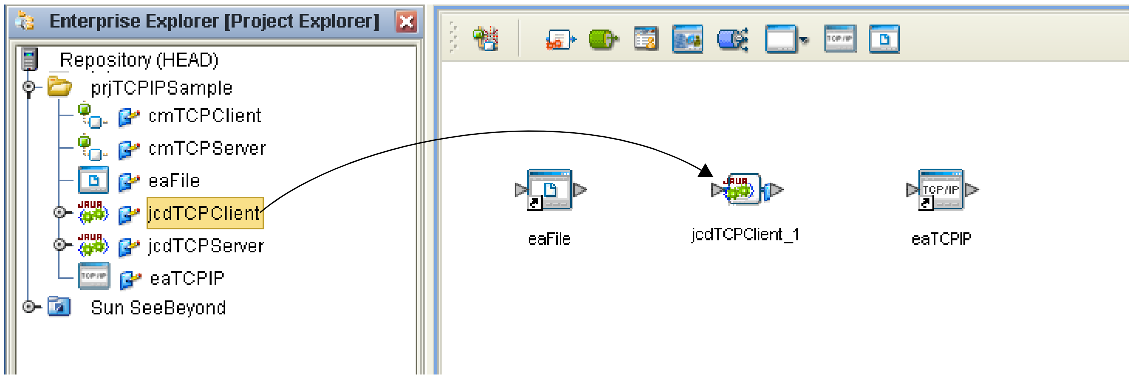
4.4.5 Binding the eWay Components

After the Collaborations have been written, the components are associated and bindings are created in the Connectivity Map.

1 From the Project Explorer, double-click **cmTCPClient** to display the **cmTCPClient** Connectivity Map.

2 Drag and drop the **jcdTCPClient** Collaboration from the Project Explorer to the **jcdTCPClient_1** Service. If the Collaboration was successfully associated, the Service's "gears" icon changes from red to green (see **Figure 21 on page 68**).

Figure 21 Connectivity Map - Binding the Components

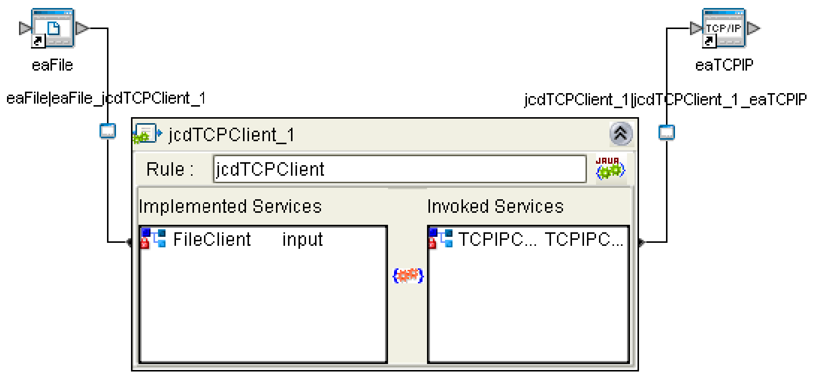


3 From the Connectivity Map canvas, double-click **jcdTCPClient_1**. The **jcdTCPClient_1** binding dialog box appears using the **jcdTCPClient** Rule.

4 From the **jcdTCPClient_1** binding dialog box, map **FileClient input** (under Implemented Services) to the output node of the inbound **eaFile** External Application. To do this, click on **FileClient input** under Implemented Services in the **jcdTCPClient_1** binding box, and drag your cursor to the output node of the **eaFile** External Application. A link now joins the two components.

5 From the **jcdTCPClient_1** binding dialog box, map **TCPIPClient_1** (under Invoked Services) to the input node of the **eaTCPIP** External Application (see Figure 22).

Figure 22 Connectivity Map - jcdTCPClient_1



6 Minimize the **jcdTCPClient_1** binding dialog box.

7 From the Project Explorer, double-click **cmTCPServer** to display the **cmTCPServer** Connectivity Map.

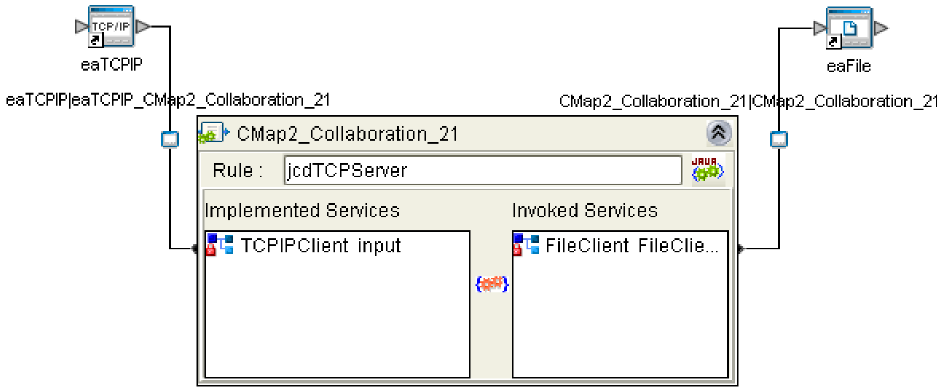
8 Drag and drop the **jcdTCPServer** Collaboration from the Project Explorer to the **jcdTCPServer_1** Service.

9 Double-click the **jcdTCPServer_1** service. The **jcdTCPServer_1** binding dialog box appears using the **jcdTCPServer** Rule.

10 From the **jcdTCPServer_1** binding dialog box, map **TCPIPClient input** (under Implemented Services) to the output node of the inbound **eaTCPIP** External Application.

11 From the **jcdTCPServer_1** binding dialog box, map **FileClient_1** (under Invoked Services) to the input node of the **eaFile** External Application (see Figure 23).

Figure 23 Connectivity Map - jcdTCPServer_1



12 Minimize the **jcdTCPServer_1** binding dialog box.

13 Save your current changes to the Repository.

4.4.6 Creating the Project’s Environment

This section provides general procedures for creating an Environment for your Project. Environments include the external systems, Logical Hosts, integration servers and message servers used by a Project and contain the configuration information for these components. For a complete explanation, see the *Sun SeeBeyond eGate™ Integrator User’s Guide.*

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 **envTCPIPSample**.

4 Right-click **envTCPIPSample** and select **New > TCPIP External System**. Name the External System **esTCPIP** and click **OK**. The **esTCPIP** window is added to the Environment Editor.

5 Right-click **envTCPIPSample** and select **New > File External System**. Name the External System **esFile** and click **OK**. The **esFile** window is added to the Environment Editor.

6 Right-click **envTCPIPSample** and select **New > Logical Host**. The **LogicalHost1** box is added to the Environment.

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.

8 Save your current changes to the repository.

4.4.7 Configuring the eWays

The **prjTCPIPSample** Project uses four eWays, each represented in the Connectivity Map as a node between an External Application and a Collaboration.

The eWay properties are set from both the Project Explorer's Connectivity Map and the Environment Explorer. To configure the eWays do the following:

Configure the File eWays Connectivity Map Properties

Open the File eWay property files by double-clicking the eWay in the Connectivity Map. When the Properties Editor appears, modify the **eaFile** File eWay Connectivity Map properties for your system, including the settings in the following tables, and click **OK**.

Table 22 eaFile eWay cmTCPClient Connectivity Map Properties

eaFile eWay cmTCPClient Connectivity Properties	
Parameter Settings - Set as directed, otherwise use the default settings.	
Input file name	*Enter the input file name such as input.txt*

Table 23 eaFile eWay cmTCPServer Connectivity Map Properties

eaFile eWay cmTCPServer Connectivity Properties	
Parameter Settings - Set as directed, otherwise use the default settings.	
Output file name	TCPIPoutput%d.dat

Configure the File eWay Environment Explorer Properties

9 From the **Environment Explorer** tree, right-click the File External System (**esFile** in this sample), and select **Properties**. The Properties Editor opens to the File eWay Environment configuration.

10 Modify the File eWay Environment configuration properties for your system, including the settings in Table 24, and click **OK**.

Table 24 File eWay Environment Explorer Properties

File eWay Environment Explorer Properties	
Inbound File eWay - Set as directed, otherwise use the default settings	
Parameter Settings > Directory	*C:/temp or the input directory of your choice*
Outbound File eWay - Set as directed, otherwise use the default settings.	
Parameter Settings > Directory	*C:/temp or the output directory of your choice*

Configuring the TCP/IP eWays

Configure the TCP/IP eWay properties from both the Connectivity Map and Environment Explorer. For more information on the TCP/IP eWay properties, see **"Creating and Configuring the TCP/IP eWay" on page 16**.

Configure the TCP/IP eWay Connectivity Map Properties

The **prjTCPIPSample** may use the default TCP/IP eWay Connectivity Map properties for the TCP/IP eWays. These default properties must still be saved prior to deployment. Open each of these TCP/IP eWay property files by double-clicking the eWays in the Connectivity Map. When the Properties Editor appears, click **OK** to save the current configuration.

Configure the TCP/IP eWay Environment Explorer Properties

11 From the **Environment Explorer** tree, right-click the TCP/IP External System (**esTCPIP** in this sample), and select **Properties**. The Properties Editor opens to the TCP/IP eWay Environment configuration.

12 Modify the TCP/IP eWay Environment configuration properties for your system, including the settings in Table 25, and click **OK**.

Table 25 TCP/IP eWay Environment Properties

eaTCPIP eWay Environment Properties	
Set as directed, otherwise use the default settings.	
TCPIPServer (inbound) eWay > TCPIP Inbound Settings > Host	*The host name or IP address to use for establishing a TCPIP connection.*
TCPIPServer (inbound) eWay > TCPIP Inbound Settings > Port	*The port number of the TCP/IP destination - indicates the port number on the local host.*
TCPIPClient (Outbound) eWay > TCPIP Outbound Settings > Host	*The host name or IP address to use for establishing a TCPIP connection (client).*
TCPIPClient (Outbound) eWay > TCPIP Outbound Settings > Port	*The port number of the TCP/IP destination - indicates the port number of the external Host.*

Note: *To run this sample, the inbound and outbound TCP/IP Environment properties must use the same TCP/IP port.*

4.4.8 Configuring the Integration Server

You must set your Sun 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. Enter **STC** as the **Specific Value** and as the **Confirm Password**, and click **OK**.
4. 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*.

4.4.9 Creating the Deployment Profile

A Deployment Profile is used to assign Collaborations and message destinations to the integration server and message server. Deployment profiles are created using the Deployment Editor.

1. From the Enterprise Explorer's Project Explorer, right-click the **prjTCPIPSample** and select **New** > **Deployment Profile**.
2. Enter a name for the Deployment Profile (for this sample **dpTCPIPSample**). Select **envTCPIPSample** 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.

4.4.10 Creating and Starting the Domain

To 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.

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.
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 *Sun SeeBeyond eGate™ Integrator System Administration Guide*.

4.4.11 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.

Note: *Projects can also be deployed from the Enterprise Manager. For more information about using the Enterprise Manager to deploy, monitor, and manage your projects, see the Sun SeeBeyond eGate™ Integrator System Administration Guide.*

4.4.12 Running the Sample

To run your deployed sample Project do the following

1 Copy the sample input file (input.txt) to your configured input directory to trigger the eWay.

2 From your output directory, verify the output data.

4.5 Customized Enveloping

The TCP/IP eWay properties provide ready-made message enveloping you can use if desired. However, if these predefined envelopes do not meet your needs, you can create your own Java class defining a custom message envelope.

The class name should be a full qualified class name, such as “com.abc.MyClass”. The class must implement interfaces **com.stc.connector.tcpip.ext.msg.EnvelopedMsgReceiver** and **com.stc.connector.tcpip.ext.msg.EnvelopedMsgSender**.

Note: *See the Javadoc provided with this eWay for complete information on the eWay’s Java classes, interfaces, and methods.*

For more information on the eWay’s enveloping feature, see **“Envelope Type” on page 31**.

The eWay allows you to set properties to implement your customized enveloping. These properties are explained under:

- **“Custom Enveloped Class Name” on page 32**
- **“Customer Defined Property” on page 32**

Creating a custom envelope

1. Using the Java Software Developer’s Kit (SDK), create the class that defines your custom message envelope, including the desired custom features.
2. For your implemented Java interface, you can import/use the following Java classes:
   - **com.stc.connector.appconn.tcpip.ext.TCPIPEXTClientApplication**
   - **com.stc.connector.message.envelope.EnvelopedMsg**
   - **com.stc.connector.tcpip.model.exception.TCPIPApplicationException**

   You can find the JAR files containing these classes in the following eGate installation directory:

   ```
   <JavaCAPS51>\edesigner\usrdir\modules\ext\tcpipextadapter
   ```

   where <*JavaCAPS51*> is your Sun Java™ Composite Application Platform Suite installation directory.

Note: *These classes are provided with the eWay. You can create your own classes, if desired.*

3. Be sure to include the JAR files discussed under step 2 in the Java classpath to compile the class.
4. Create, compile, and package a JAR file that contains your custom envelope class, for example, **customenvelope.jar**.
5. In the eWay Properties Editor for the eWay, you must set the following properties:

- **Envelope Type**: **Custom**
- **Custom Property**: Any property that needs to be defined in order for your custom envelope to work, for example, delimiters.
- **Custom Enveloped Class Name**: For example:

  `com.stc.customenvelope.CustomEnvelope1`

You must enter the full path location of the custom envelope class.

Next, you must include your custom envelope in the current eGate Project.

To import and use a custom envelope in a Project

Note: *Custom envelopes created to execute with a Logical Host prior to version 5.0.4 must be recompiled into byte-code without any code change before the jar file may be imported into version 5.0.4 or newer of the Logical Host. Alternatively, you can incorporate the source code of the custom envelope into your collaboration so that it is compiled each time the project is activated.*

1. From the Enterprise Designer's **Project Explorer** pane, right-click your Project and choose **Import > File** from the shortcut menu. The **Import Files** dialog box appears.
2. From the **Import > Files** dialog box, select your custom envelope's JAR file and click **Import**. Your custom envelope's JAR file is now available from the Project Explorer tree.

Note: *If there is an unwanted JAR file in the **Project Explorer**, right-click on the file name and choose **Delete** from the shortcut menu to delete the unwanted file.*

3. Open any Collaboration that uses your custom envelope, in the Collaboration Editor (Java). To do this, double-click the Collaboration in the Project Explorer tree.
4. From the Collaboration Editor toolbar, click the **Import JAR File** icon. The **Add/ Remove JAR Files** dialog box appears.
5. Click **Add** and select your custom envelope's JAR file. This action adds your customized envelope's class to the current Collaboration's classpath.

Note: *If you delete a file from the **Project Explorer**, you must also remove the file from every Java-based Collaboration where it is used. Click the* **Import JAR File** icon and use the **Add/Remove JAR Files** dialog box to remove the files.

Custom Envelope Sample

The following text provides a sample of a custom envelope:

```
package com.stc.customenvelope;
import java.io.IOException;
import java.util.*;
import org.apache.log4j.Logger;

import com.stc.connector.appconn.tcpip.ext.TCPIPEXTClientApplication;
import com.stc.connector.message.envelope.EnvelopedMsg;
import com.stc.connector.tcpip.model.exception.TCPIPApplicationException;
import com.stc.connector.tcpip.ext.msg.*;

/**
 * A custom envelope message sample:
```

```
 * Read data on socket from position "offset" until length "len".
 * The "offset" and "len" are defauled as 4 and 10.
 * Note: This sample is only used to demonstrate the extensibility of envelope type.
 *       You can define your configuration parameters on your demand (for excample,
 *       you can make "offset" and "len" configurable).
 * @version cvs revision: $Revision: 1.4 $ Last Modified: $Date: 2003/07/19 21:17:19 $
 */
public class CustomEnvelopedSample
    implements EnvelopedMsgReceiver, EnvelopedMsgSender {
    private Logger logger = Logger.getLogger("STC.eWay.TCPIP." + getClass().getName());
    private String logMsg;

    // class variables
    private int offset;
    private int len;

    /**
     * Constructor for CustomEnvelopedMsgSample03.
     */
    public CustomEnvelopedSample() {
        // you can make them configurable (for example, get them from a property file).
        // now we just give the fixed value to make the case easy.
        this.offset = 0;
        this.len = 0;
    }


    /**
     * @see com.stc.connector.tcpip.ext.msg.EnvelopedMsgReceiver#receiveEnvelopedMsg
     * (TCPIPEXTClientApplication)
     */
    public EnvelopedMsg receiveEnvelopedMsg(TCPIPEXTClientApplication app)
        throws TCPIPApplicationException, IOException {
        // do additional validation if needed.
        /*
        if (this.offset < 0) {
            logMsg =
                "CustomEnvelopedMsgSample01.receiveEnvelopedMsg(): "
                    + "Invalid input, the offset shouldn't be less than 0.";
            logger.error(logMsg);
            throw new TCPIPApplicationException(logMsg);
        }
        */
     /* The follwing code parses the eWay custom property given as 4&10
     * where 4=offset and 10 = len and & = delimeter
     */

     String config = app.getMessageInfo().getCustomerDefinedProperty();
     StringTokenizer st = new StringTokenizer(config,"&");
     if (st.hasMoreTokens()) {
               this.offset = Integer.parseInt(st.nextToken());
               }
               if (st.hasMoreTokens()) {
               this.len = Integer.parseInt(st.nextToken());
          }

        int readLen = 0;


        byte[] ignore = new byte[this.offset];
        readLen = app.getSocket().getInputStream().read(ignore);

        if (readLen != this.offset) {
            logMsg =
                "CustomEnvelopedMsgSample01.receiveEnvelopedMsg(): "
                   + "Invalid data/offset, no enough data is available on the socket.";
            logger.error(logMsg);
            throw new TCPIPApplicationException(logMsg);
        }


        byte[] bytes = new byte[this.len];
        readLen = app.getSocket().getInputStream().read(bytes);

        if (readLen != this.len) {
            logMsg =
                "CustomEnvelopedSample.receiveEnvelopedMsg(): "
                    + "Invalid data/len, no enough data is available on the socket.";
            logger.error(logMsg);
            throw new TCPIPApplicationException(logMsg);
        }
```

```
            return this.new MyEnvelopedMsgImpl(bytes);
        }

        /**
         * @see com.stc.connector.tcpip.ext.msg.EnvelopedMsgSender#sendEnvelopedMsg
         * (EnvelopedMsg, TCPIPEXTClientApplication)
         */
        public void sendEnvelopedMsg(
            EnvelopedMsg msg,
            TCPIPEXTClientApplication app)
            throws TCPIPApplicationException, IOException {
            // do additional validation if needed.
            app.sendBytes(msg.getBytes());
        }

        /**
         * An inner classe that implements an enveloped message.
         *
         * @version cvs revision: $Revision: 1.4 $   Last Modified: $Date: 2003/07/19
    21:17:19 $
         */
        private class MyEnvelopedMsgImpl extends EnvelopedMsg {
            private byte[] data;

            /**
             * Constructor for MyEnvelopedMsgImpl.
             * @param data byte[]
             */
            public MyEnvelopedMsgImpl(byte[] data) {
                super(null);
                this.data = data;
                //this.setEnvelopeType("Custom");
            }

            /**
             * @see com.stc.connector.message.common.AbstractMsg#getBytes()
             */
            public byte[] getBytes() throws IOException {
                return this.data;
            }
        }
    }
```

Chapter 5

Using eWay Java Methods

This chapter provides an overview of the Java classes/interfaces and methods contained in the TCP/IP eWay. These methods are used to extend the functionality of the eWay.

What's in This Chapter

- **TCP/IP eWay Methods and Classes: Overview** on page 78
- **TCP/IP eWay Javadoc** on page 78
- **Protected and Unsupported Methods** on page 79

5.1 TCP/IP eWay Methods and Classes: Overview

The TCP/IP eWay exposes various Java methods to add extra functionality to the eWay. These methods make it easier to set information in the TCP/IP eWay Object Type Definitions (OTDs), as well as get information from them.

eWay Java Classes/Interfaces

The TCP/IP eWay provides the following Java classes/interfaces:

- **TCPIPEXTClientApplication**: Extends **TCPIPClientApplication** with the message envelope interface for TCP/IP.
- **TCPIPEXTServerApplication**: Extends **TCPIPServerApplication** with the message envelope interface for TCP/IP.
- **TCPIPClientApplication**: Represents the TCP/IP client application interface.
- **TCPIPServerApplication**: Represents the TCP/IP server application interface.

5.1.1 TCP/IP eWay Javadoc

- A TCP/IP eWay Javadoc is also provided, which documents the Java methods available with the TCP/IP eWay. The Javadoc is uploaded with the eWay's documentation file (TCPIPeWayDocs.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.

5.2 Protected and Unsupported Methods

The Collaboration Editor's Java Source Editor, features a code completion helper pull-down list that allows the user to see and select available methods to add to your code. This list may also display a number of methods that are not available through the configuration object. These are "protected" methods, not "public" methods, and cannot be invoked against a configuration object in Collaboration code. Attempting to invoke any of these "protected" methods will cause a compiling error during activation.

5.2.1 Unsupported Methods

The Collaboration Editor may allow you to access some method under the **Socket** node of the TCP/IP Object Type Definition (OTD) which cannot be used by the Collaboration (Java). Though these methods may be accessible, they are not supported by the eWay or eGate. See the TCP/IP eWay Javadoc for a list of supported methods.

Index

A

Automap **72**

B

basic features **6**
binding
  eWay components **68**
Business Rule
  comments
    creating **59**
Business Rules
  Collaboration Editor (Java) **59**

C

Collaboration Definitions
  wizard **57**
Collaboration Editor (Java) **59**
comments
  creating **59**
Connectivity Map **66**
conventions, text **8**
Custom Enveloped Class Name **32**, **44**

D

Dedicated Session Mode **20**
Deployment Profile
  Automap **72**
document
  scope **8**

E

Envelope Message (Client) **42**
Envelope Type **31**, **43**
  Begin-End Marked **33**
  Custom **35**
  End Marked **33**
  Fixed Length **34**
  Length Prefixed **34**
  Marked and Fixed **34**
  Per Active Connection **35**
eWay operation, general **6**
eWays
  creating **17**
External Application
  creating **16**

G

General Client Settings **36**
general operation, eWay **6**

I

Ignore Until Char Value **33**, **45**

J

Java methods and classes
  overview **78**

K

Keep Alive (client) **37**
Keep Alive (server) **21**

M

Max Data Size (client) **36**
Max Data Size (server) **20**
methods not used by eWay **79**

O

operating systems
  requirements **10**
  supported **10**

P

platforms
  requirements **10**
  supported **10**
Project
  External Environment **69**
  importing **55**
Project sample
  operation **56**
  overview **56**
properties
  Always Create New Connection **39**
  At Fixed Rate **28**, **30**
  Auto Reconnect Upon Matching Failure **39**
  Bytes to Read **32**, **44**

Close Notification **25**
Connection Type **21**, **37**
Connectivity Map properties **16**
  modifying **17**
Custom Enveloped Class Name **32**, **44**
Customer Defined Property **32**, **44**
Dedicated Session Mode **20**
Delay **27**, **29**
Envelope Type **31**, **43**
Environment Explorer properties **16**
Environment properties
  modifying **17**
IdleTimeout **26**
Ignore Until Char Value **33**, **45**
Keep Alive **21**
Keep Alive (client) **37**
Keep Alive (server) **21**
Max Binding Retry **23**, **41**
Max Connection Pool Size **25**
Max Connection Retry **40**
Max Data Size **20**, **36**
Max Data Size (client) **36**
Max Data Size (server) **20**
modifying **18**
  Connectivity Map properties **17**
Numeric Representation **32**, **45**
Period **27**, **30**
Receive Buffer Size **22**
Receive Buffer Size (client) **38**
Receive Buffer Size (server) **22**
Retry Binding Interval **23**, **41**
Retry Connection Interval **40**
Schedule Type **27**, **29**
Scheduler **27**, **29**
Scope Of Connection **25**
Scope Of State **20**, **36**
Send Buffer Size **22**
Send Buffer Size (client) **38**
Send Buffer Size (server) **22**
Server Socket Factory Implementation Class **21**
Server Socket Factory Implementation Class Name **21**
ServerSoTimeout **21**, **23**, **37**
Socket Factory Implementation Class **38**
SoLinger **22**
SoLinger (client) **38**
SoLinger (server) **22**
SoLinger Timeout **22**
SoLinger Timeout (client) **38**
SoLinger Timeout (server) **22**
SoTimeout **22**
SoTimeout (client) **38**
SoTimeout (server) **22**
Store Until Char Value **33**, **45**
TCPIP Inbound Settings - Client Connection Establishment **24**
TCPIP Inbound Settings - Connection Management **25**
TCPIP Outbound Settings - Connection Establishment **39**
TCPIP Outbound Settings - Server Port Binding **41**
TCPIP Server Inbound Schedules - Service Schedule **29**
TcpNoDelay **22**
TcpNoDelay (client) **38**
TcpNoDelay (server) **22**
Time To Wait Before Attempting Connection **24**, **39**
Width of Length **32**, **44**
Properties Editor **18**

R

Receive Buffer Size (client) **38**
Receive Buffer Size (server) **22**

S

scope **8**
Send Buffer Size (client) **38**
Send Buffer Size (server) **22**
Server Socket Factory Implementation Class **21**
ServerSoTimeout **21**, **23**
setting eWay properties
  client mode properties **36**
  server mode properties **19**, **46**
  values
    Envelope Message (Client) **42**
    General Client Settings **36**
    TCP/IP Base Settings **37**
Socket Factory Implementation Class **38**
SoLinger (client) **38**
SoLinger (server) **22**
SoLinger Timeout (client) **38**
SoLinger Timeout (server) **22**
SoTimeout (client) **38**
SoTimeout (server) **22**
supported operating systems **10**

T

TCP/IP Base Settings **37**
TCP/IP eWay
  properties
    Connectivity Map properties **16**
    Environment Explorer properties **16**

TCPIP.TCPIPClient **55**
TcpNoDelay (client) **38**
TcpNoDelay (server) **22**
text conventions **8**