SeeBeyond ICAN Suite

TCP/IP eWay Intelligent Adapter User's Guide

Release 5.0.1

Version 20031119145336.

Contents

Chapter 1

Introducing the TCP/IP eWay

This guide explains how to install, set properties for, and operate the SeeBeyond Technology Corporation's (SeeBeyond) TCP/IP eWay Intelligent Adapter, referred to as the TCP/IP eWay throughout this document.

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

Chapter Topics

1.1 Overview

The TCP/IP eWay 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 and its user interface, see the *eGate Integrator User's Guide*.

TCP/IP eWay

The TCP/IP eWay allows you to create a client interface to the server or implement a server in eGate, using an eGate 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 your desired message enveloping, using predefined envelope types. If these types do not meet your needs, you can customize your own enveloping, using specialized eWay interfaces designed for this purpose.

The TCP/IP eWay properties provide the necessary parameters for the operation of the TCP/IP eWay. These properties are adopted into the OTD's functions.

See **Chapter 3** for details on how to set the TCP/IP eWay properties.

1.2 Supported Operating Systems

The TCP/IP eWay is available for the following operating systems:

- Windows Server 2003, Windows XP SP1a, and Windows 2000 SP3
- HP-UX 11.0 and 11i (RISC)
- IBM AIX 5.1 and 5.2
- Red Hat Linux 8 (Intel)
- Red Hat Linux Advanced Server 2.1 (Intel)
- Sun Solaris 8 and 9

1.3 System Requirements

To use the TCP/IP eWay, you need:

- eGate Logical Host, version 5.0 or later.
- TCP/IP network connection.

Logical Host requirements

The eWay must be set up and administered using the Enterprise Designer. For complete information on the eGate Enterprise Designer system requirements, see the *SeeBeyond ICAN Suite Installation Guide*.

1.4 External System Requirements

To enable the eWay to communicate properly with the TCP/IP system, you need:

- Host on which the server is running (host name)
- Port on which the server is listening (port number)

Chapter 2

Installing the TCP/IP eWay

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

Chapter Topics

2.1 Installing on Windows and UNIX Operating Systems

During the eGate Integrator installation process, the Enterprise Manager, a Web-based application, is used to select and upload eWays (**.sar** files for eWays) from the eGate installation CD-ROM to the Repository.

When the Repository is running on a UNIX operating system, eGate and the eWays are installed using the Enterprise Manager on a computer running Windows connected to the Repository server.

2.1.1 Installing eGate

The eGate installation process includes the following operations:

- Installing the eGate Repository
- Uploading products to the Repository
- Downloading components (such as eGate Enterprise Designer and Logical Host)
- Viewing product information home pages

2.1.2. Installing the TCP/IP eWay on an eGate-supported System

The TCP/IP eWay is installed during the installation of eGate Integrator. Follow the instructions in the *SeeBeyond ICAN Suite Installation Guide* for installing eGate and include the following steps:

1. During the procedures for uploading files to the eGate Repository using the Enterprise Manager, after uploading the **eGate.sar** file, select and upload the following files:
   - **TCPIPeWay.sar** (to install the TCP/IP eWay)
   - **FileeWay.sar** (to install the File eWay, used in the sample Project)

2 Continue installing the eGate Integrator as instructed in the *SeeBeyond ICAN Suite Installation Guide*.

2.1.3. After Installation

Once you have installed and set properties for the eWay, you must then incorporate it into a Project before it can perform its intended functions. See the *eGate Integrator User's Guide* for more information on incorporating the eWay into an eGate Project.

Chapter 3

Setting TCP/IP eWay Properties

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

Chapter Topics

3.1 TCP/IP eWay Properties Sheet

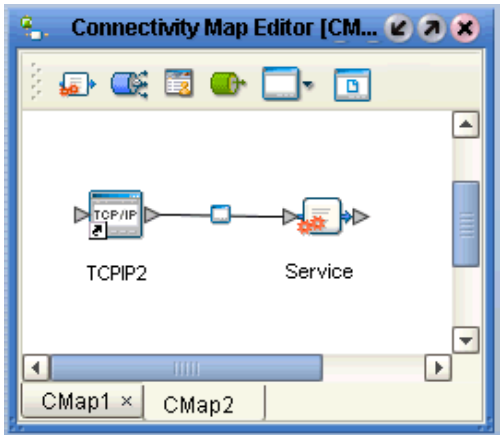
When you install the TCP/IP eWay, a default properties template for the eWay is also installed. The template's default properties are accessible via the eGate Enterprise Designer. These default settings apply to all TCP/IP eWays you use within your current Project or Business Process.

You can set properties for each individual eWay using the Enterprise Designer's eWay **Properties** sheet. This section describes general procedures on how to change these default properties for the eWay. For details on these steps, see the *eGate Integrator User's Guide*.

To set properties for the TCP/IP eWay on the Connectivity Map

1. From the eGate Enterprise Designer's **Project Explorer** create at least one Connectivity Map.
2. Create the desired external systems for your one or more Connectivity Maps.
3. Select the external application whose default eWay properties you want to change by clicking the **eWay** icon. This icon is located on the link between an **External Application** icon and a **Service** icon on the Connectivity Map canvas. See **Figure 1 on page 11**.

Figure 1 eWay Icon



The eWay **Properties** sheet appears. Figure 2 on page 13, and Figure 3 on page 22 show the eWay's default properties available from the **Project Explorer** and Connectivity Map. You can use this window to modify the current eWay's properties settings.

To set properties for the TCP/IP eWay using on the Project Environment

1. From the Enterprise Designer, create your external systems.
2. Click the **Environment Explorer** tab (at the bottom of the left pane).
3. Create an environment for your project, then create external systems on the Environment canvas to correspond to the systems you created using the **Project Explorer**.
4. Select the external system whose default eWay properties you want to change by right-clicking the desired system's icon in the **Environment Explorer**.

   The eWay **Properties** sheet appears. Figure 4 on page 26 shows the eWay's default properties available from the **Environment Explorer**. You can use this dialog box to modify the eWay properties associated with the current external system.
5. Click **OK** then **Save All** to save your changes.

To use the eWay Properties sheet

- The TCP/IP eWay's properties are set using the eGate Enterprise Designer's eWay **Properties** sheet. The default properties are automatically provided.
- Clicking the **Configuration** (Connectivity Map) or **Environment Configuration** folder in the left pane displays properties group subfolders in the right pane. Click any subfolder to display the eWay's editable properties.
- Many of the entries allow you to enter text. Click the desired text box, then click the ellipsis (**...**) that appears, to open a dialog box for this purpose.

Note: *Even if you do not change the eWay's properties, you must open each **Properties** sheet for every eWay and click **OK** to activate the eWay.*

The rest of this chapter explains all of the eWay's properties in detail, under the following sections:

3.2 Settings Properties on the Connectivity Map

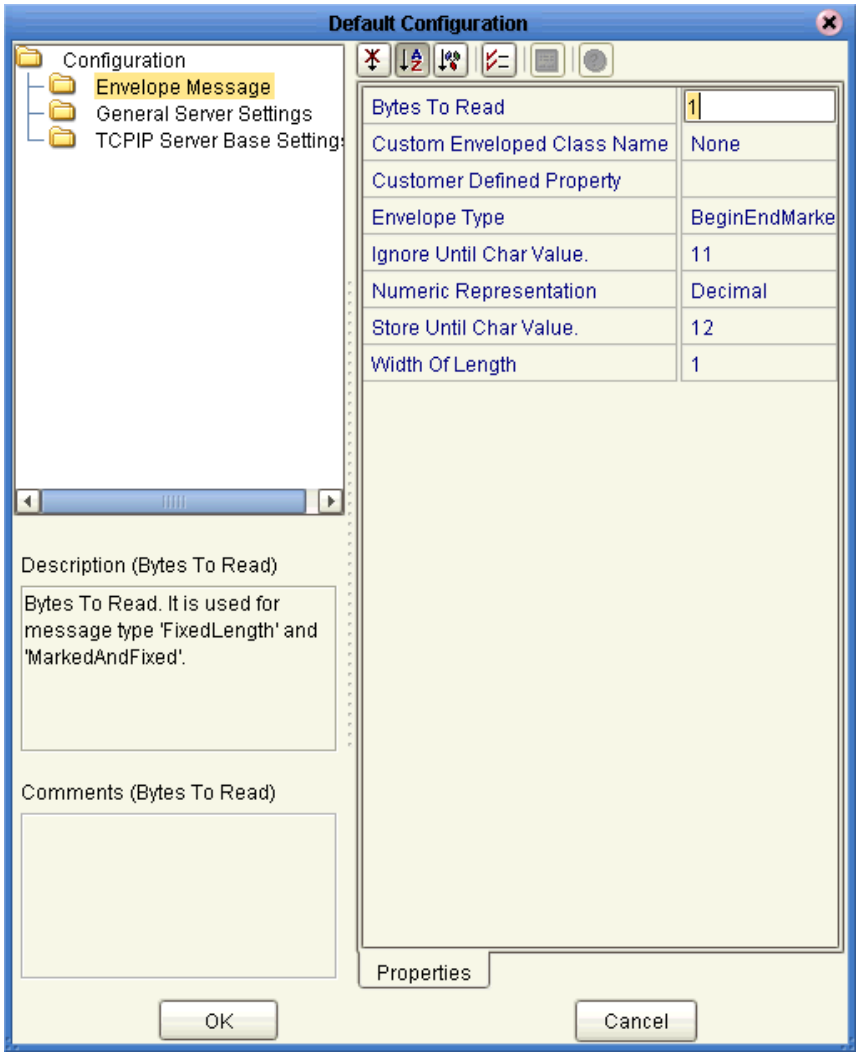
This section explains in detail the eWay's editable properties accessible via the eGate Enterprise Designer's **Project Explorer** and Connectivity Map. You can set these properties using the eWay **Properties** sheet.

The TCP/IP eWay operates in two modes, client (outbound) and server (inbound). The properties relating to each of these modes are explained under the following sections:

3.2.1 Setting eWay Properties in Server Mode

The server (inbound) properties settings determine the eWay's behavior for input operations. See Figure 2.

Figure 2 eWay Properties Sheet: Server (Inbound) Connectivity Map



These eWay server mode properties are organized under the following subfolders:

- **"Envelope Message (Server)" on page 14**
- **"General Server Settings" on page 18**
- **"TCP/IP Server Base Settings" on page 19**

3.2.2 Envelope Message (Server)

This section explains the envelope message format properties for the server. These properties are all associated with TCP/IP enveloping. The properties in this section are:

Bytes to Read

Description

Specifies the number of bytes to read.

Required Values

An integer; the range is 1 to 2 billion, and the default is 1.

Custom Enveloped Class Name

Description

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 Java Collaboration Editor's file import feature. For complete information on how to use this feature, see the *eGate Integrator User's Guide*.

For more details, see **"Customized Enveloping" on page 30**.

Required Values

A valid full Java class name.

Customer Defined Property

Description

Allows you to enter any set of user-defined properties for a custom envelope, to be used when the **Envelope Type** property is set to **Custom**. You can parse this information, such as delimiters, into your customized envelope message implementation.

Required Values

Any text string.

Envelope Type

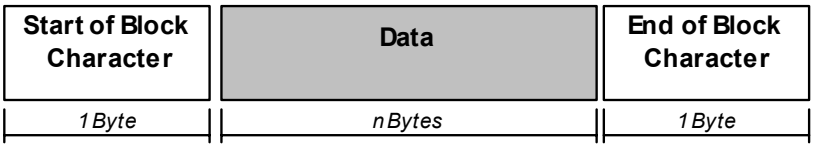
Description

Specifies the envelope type. The envelope type defines where a message starts and stops. The rest of this section explains 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 16** *for an explanation of how the data is handled by that envelope type.*

BeginEndMarked

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

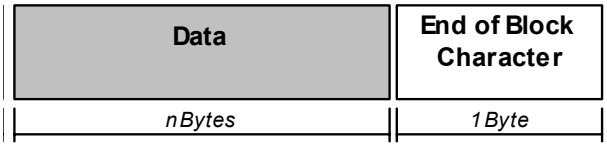


The **Start of Block Character** component of the Begin-end Marked envelope is the same as the editable **Ignore Until Character**. The **End of Block Character** component is the same as the editable **Store Until Character**.

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

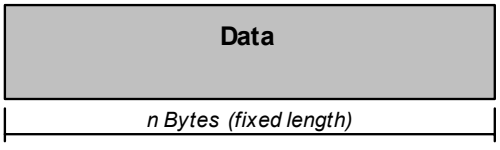
The **EndMarked** envelope has the following structure:



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

FixedLength

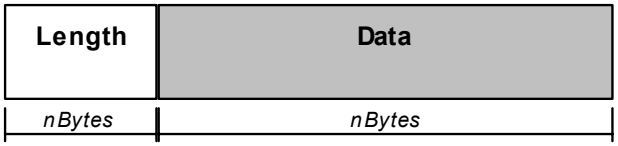
The **FixedLength** envelope has the following structure:



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

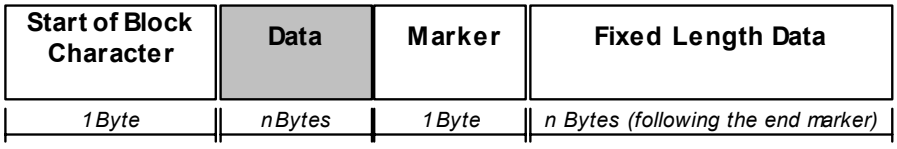
Table 1 displays the available values for each **Numeric Representation** and corresponding **Allowed Width**.

Table 1 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).

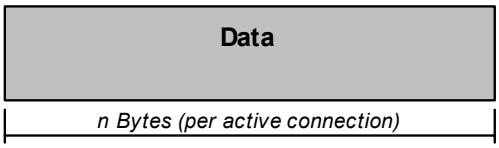
You can express this data structure in the following formula:

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 14**), 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.*

Required Values

For the **Envelope Type**, enter one of the following properties denoting the envelope type:

- **BeginEndMarked**
- **EndMarked**
- **FixedLength**
- **LengthPrefixed**
- **MarkedAndFixed**
- **PerActiveConnection**
- **Custom**

The default is **BeginEndMarked**.

Ignore Until Char Value

Description

Specifies the value for the ignore-until (same as begin block) character. All incoming characters are ignored until this character is encountered.

Required Values

A byte; the default is **11**.

Numeric Representation

Description

Specifies how the counter is represented numerically. This value is expressed in one of the following formats: decimal, hexadecimal, octal, network short, or network long.

Required Values

Enter **Decimal**, **Hexadecimal**, **Octal**, **Network short**, or **Network long**. **Decimal** representation is the default.

Store Until Char Value

Description

Specifies the store-until (same as end block) character value. All incoming characters are stored until this character is encountered.

Required Values

A byte; the default is **12**.

Width of Length

Description

Specifies the length component in the envelope. This value refers to the number of bytes used to represent the number in the length field of an envelope.

Required Values

An integer; the range is 1 to 10, and the default is **1**. This property must be set to **2** for **Network short** and **4** for **Network long**.

3.2.3 General Server Settings

These properties provide the Dedicated Session Mode and maximum data size message settings for the server. The properties in this section are:

- **"Dedicated Session Mode" on page 19**
- **"Max Data Size" on page 19**

Dedicated Session Mode

Description

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.

Required Values

True or **False**. **True** enables the option; the default is **False**.

Max Data Size

Description

Allows you to define the maximum size of the data that the programs can hold internally.

Required Values

The valid range is from 1 to 2 GB (the maximum value of the Java integer).

3.2.4 TCP/IP Server Base Settings

These properties allow you to set the basic TCP/IP values for the server. The properties in this section are:

- **"Keep Alive" on page 20**
- **"Receive Buffer Size" on page 20**
- **"Send Buffer Size" on page 20**
- **"Server Socket Factory Implementation Class" on page 20**
- **"ServerPort" on page 26**
- **"ServerSoTimeout" on page 21**
- **"SoLinger" on page 21**
- **"SoLinger Timeout" on page 21**
- **"SoTimeout" on page 21**
- **"TcpNoDelay" on page 21**

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

Keep Alive

Description

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.*

Required Values

True or **False**. **True** enables the option.

Receive Buffer Size

Description

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.

Required Values

The receive buffer size; the default is **8192**.

Send Buffer Size

Description

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.

Required Values

The send buffer size; the default is **8192**.

Server Socket Factory Implementation Class

Description

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

Required Values

A valid Java class name; the default is:

com.stc.connector.tcpip.model.factory.TCPIPSocketFactoryImpl

ServerSoTimeout

Description

Allows you to set or get the server **SO_TIMEOUT** value, in milliseconds.

Required Values

The server's **SO_TIMEOUT** value in milliseconds; the default is **10,000** milliseconds (10 seconds).

SoLinger

Description

Specifies whether the server's **SO_LINGER** option is enabled or disabled; used for the accepted client socket.

Required Values

True or **False**. **True** enables the option.

SoLinger Timeout

Description

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.

Required Values

The linger time-out in seconds; the default is **-1** seconds, meaning that the **SO_LINGER** option has been disabled.

SoTimeout

Description

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.

Required Values

The **SO_TIMEOUT** value in milliseconds; the default is **10,000** milliseconds (10 seconds).

TcpNoDelay

Description

Specifies whether the server's **TCP_NODELAY** option (that is, Nagle's algorithm) is enabled or disabled; used for the accepted client socket.

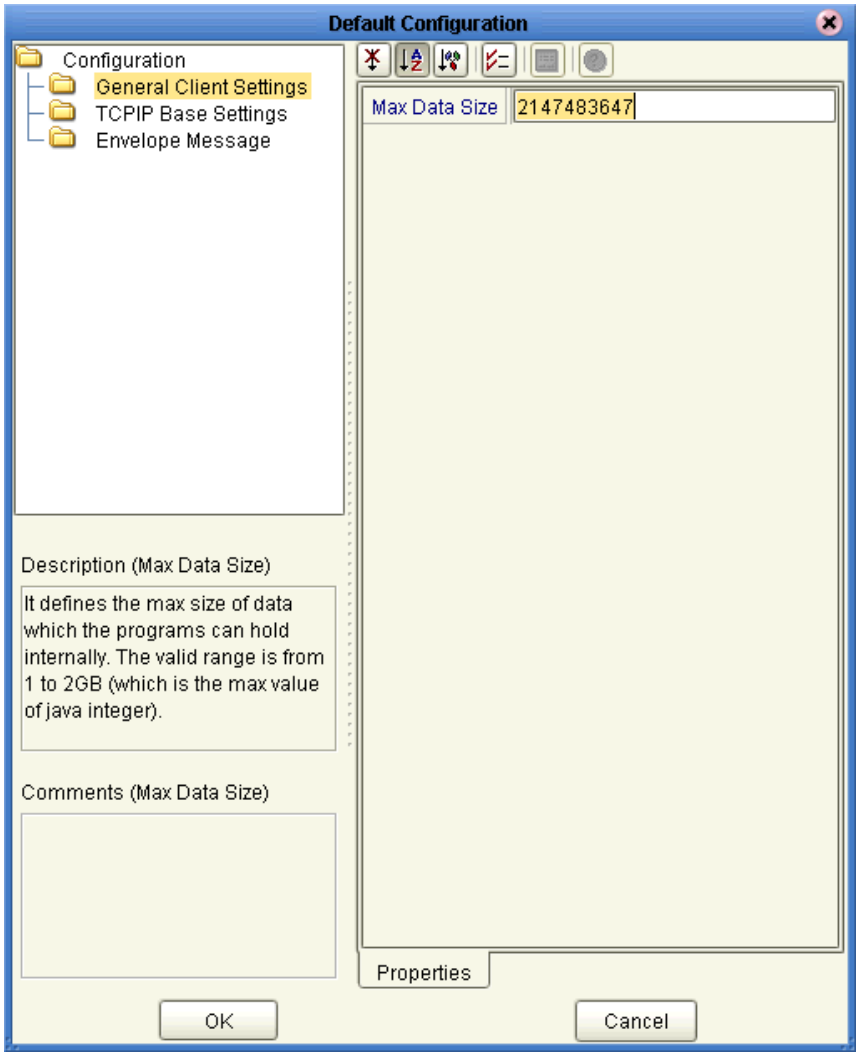
Required Values

True enables the option, and **False** disables it.

3.2.5 Setting eWay Properties in Client Mode

The client (outbound) properties settings determine the eWay's behavior for output operations. See Figure 3.

Figure 3 eWay Properties Sheet: Client (Outbound) Settings/Connectivity Map



These eWay client mode properties are organized under the following folders:

- **"General Client Settings" on page 23**
- **"TCP/IP Base Settings" on page 23**
- **"Envelope Message (Client)" on page 25**

3.2.6 General Client Settings

Specifies the maximum data size for the client. The property in this section is:

- **"Max Data Size" on page 23**

Max Data Size

Description

Allows you to define the maximum size of the data that the programs can hold internally.

Required Values

The valid range is from 1 to 2 GB (the maximum value of the Java integer).

3.2.7 TCP/IP Base Settings

Specifies the basic TCP/IP values for the client. The properties in this section are:

- **"Keep Alive" on page 23**
- **"Receive Buffer Size" on page 23**
- **"Send Buffer Size" on page 24**
- **"Socket Factory Implementation Class" on page 24**
- **"SoLinger" on page 24**
- **"SoLinger Timeout" on page 24**
- **"SoTimeout" on page 25**
- **"TcpNoDelay" on page 25**

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

Keep Alive

Description

Specifies whether the client's **SO_KEEPALIVE** option is enabled or disabled.

Required Values

True or **False**. **True** enables the option.

Receive Buffer Size

Description

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.

Required Values

The receive buffer size; the default is **8192**.

Send Buffer Size

Description

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.

Required Values

The send buffer size; the default is **8192**.

Socket Factory Implementation Class

Description

Enter the name of the Java class that implements the client socket factory. This class is used for creating a client socket.

If you have provided your own client 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

Required Values

A valid Java class name; the default is:

com.stc.connector.tcpip.model.factory.TCPIPSocketFactoryImpl

SoLinger

Description

Specifies whether the client's **SO_LINGER** option is enabled or disabled.

Required Values

True or **False**. **True** enables the option.

SoLinger Timeout

Description

Specifies the client's linger time-out in seconds. The maximum time-out value is platform specific. The setting only affects the socket close.

Required Values

The linger time-out in seconds; the default is **-1** seconds, meaning that the **SO_LINGER** option has been disabled.

SoTimeout

Description

Allows you to set or get the client's **SO_TIMEOUT** value, in milliseconds.

Required Values

The **SO_TIMEOUT** value in milliseconds; the default is **10,000** milliseconds (10 seconds).

TcpNoDelay

Description

Specifies whether the client's **TCP_NODELAY** option (that is, Nagle's algorithm) is enabled or disabled.

Required Values

True enables the option, and **False** disables it.

3.2.8 Envelope Message (Client)

These properties are the envelope message format settings for the client. These properties operate in the same way as those for the server (inbound) eWay properties. See **"Envelope Message (Server)" on page 14** for details.

3.3 Setting Properties in the Project Environment

This section explains in detail the eWay's editable properties accessible via the eGate Enterprise Designer's **Environment Explorer**. You can set these properties using the eWay **Properties** sheet.

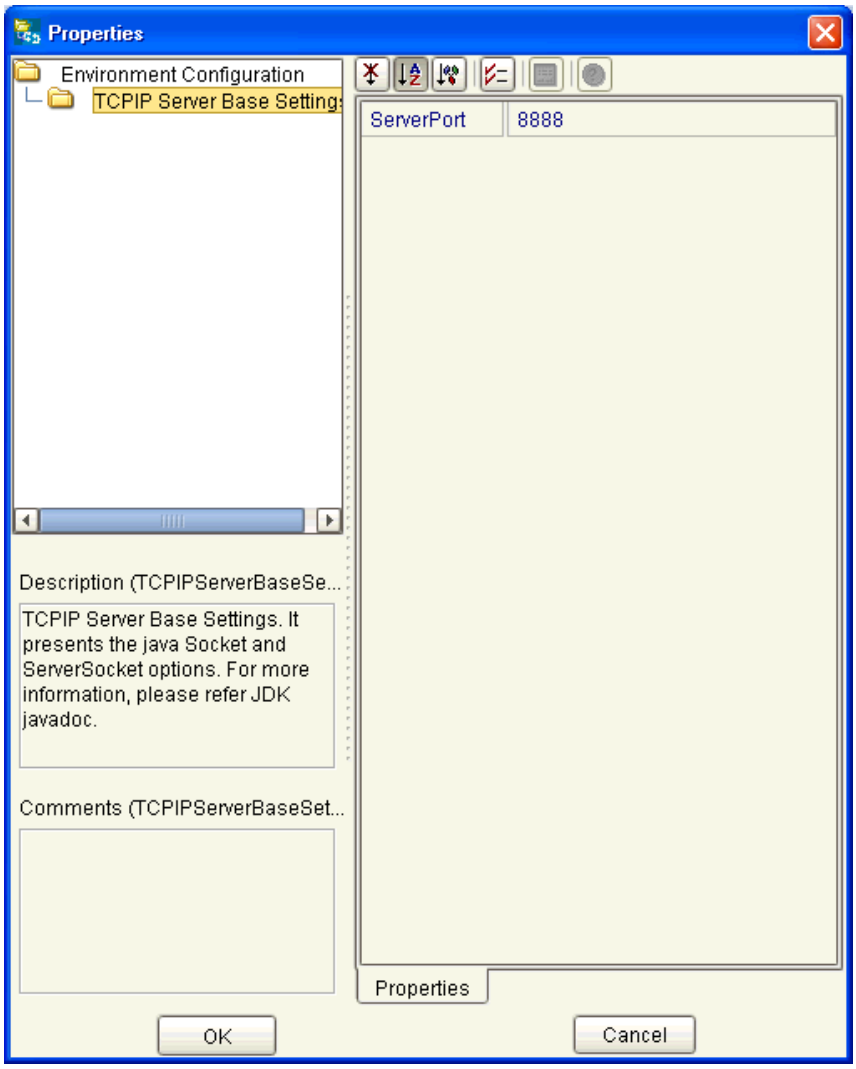
The TCP/IP eWay operates in two modes, client (outbound) and server (inbound). The properties relating to each of these modes are explained under the following sections:

- **"Setting eWay Properties in Server Mode" on page 25**
- **"Setting eWay Properties in Client Mode" on page 27**

3.3.1 Setting eWay Properties in Server Mode

The server (inbound) properties settings determine the eWay's behavior for input operations. See **Figure 4 on page 26**.

Figure 4 eWay Properties Sheet: Server (Inbound) Settings/Environment



The eWay server property in the **Environment Explorer** is:

- **"ServerPort" on page 26**

ServerPort

Description

Specifies the TCP/IP port number of the server.

Required Values

A valid port number; the default is **8888**.

3.3.2 Setting eWay Properties in Client Mode

The client (outbound) properties settings determine the eWay's behavior for output operations. See Figure 5.

Figure 5 eWay Properties Sheet: Client (Outbound) Settings/Environment

These eWay client properties in the **Environment Explorer** are:

- **"Host" on page 27**
- **"Port" on page 28**

Host

Description

Specifies the default server host used (in the client mode) to establish a TCP/IP connection. You can use either the host name or the IP address.

Note: *In the server mode, this property is not necessary because the server gets the client's host name automatically.*

Required Values

A valid TCP/IP host name or IP address; the default is **localhost**.

Port

Description

Specifies the TCP/IP port number of the host. This number represents the port the client connects to. In the server mode, this is the port number the server listens to for inbound connections.

Required Values

A valid port number; the default is **7777**.

Chapter 4

Building a TCP/IP eWay Project

This chapter describes how to create an eGate Project. These procedures are the same as those used when implementing a TCP/IP eWay in a production environment.

See the *eGate Tutorial* for an explanation of how to use the eGate Enterprise Designer to create and set up the components of an eGate Project.

Chapter Topics

4.1 Project Canvas

Each eGate Project is created using the Enterprise Designer's Project canvas. The Project canvas contains windows that represent the various stages of your Project. The types of windows in your Project canvas area include:

- **OTD Editor**: Contains the source files used to create Object Type Definitions (OTDs) to use with a Project. OTDs carry information and instructions through the eGate system. The eWay employs a two TCP/IP OTDs, client (outbound) and server (inbound), enabling the eWay's client and server modes.
- **Java Collaboration Definition Editor**: Allows you to create a Java-based Collaboration Definition for each eGate Collaboration in your Project. This feature makes it possible to develop external Collaboration definitions.
- **Connectivity Map**: Contains the business logic components, such as Collaborations, Topics, Queues, and eWays, that you can use to create the structure of the Project.

4.2 Setting Up the eWay

In general, you set up the TCP/IP eWay as follows:

- **Creating a Connectivity Map**: When you create a Connectivity Map using the TCP/IP eWay, the Enterprise Designer places the TCP/IP external application object on a toolbar where it can be dragged onto the canvas of the map, as needed.
- **Creating a Collaboration**: A wizard allows you to create a Java Collaboration using one or both TCP/IP OTDs. A Collaboration Definition Editor allows you to create the Collaboration's business rules. You can create the desired business rules by dragging and dropping values from a source OTD onto the nodes of a destination TCP/IP OTD and other OTDs. These nodes represent TCP/IP functions, which are in turn able to call TCP/IP methods. Another dialog box also allows you to create Collaboration Bindings that control the Collaboration's inputs and outputs.
- **Creating the Client/Server Interface**: You create a TCP/IP client interface to the server or implement a server in eGate using the drag-and-drop features of the TCP/IP OTD in the Collaboration Definition Editor. The logic of the Collaboration employs high-level application programming interfaces (APIs) that use either the eWay's predefined properties or run-time supplied properties that make calls directly into the TCP/IP Resource Adapter (RA).
- **Setting Message Enveloping**: Via the TCP/IP OTDs (client and server), the eWay can use enveloping, create responses, and employ other higher-level functions that hide the complexity of multiple TCP/IP calls. In both client and server modes, you can choose an enveloping standard from the eWay's predefined set or create a custom message envelope that implements a TCP/IP enveloping interface available in the RA. This interface can include envelope and de-envelope method calls.
- **Setting eWay Properties**: Once a data flow is created between the current Collaboration and the external application, an eWay object is created. You can set the properties of this eWay object as desired for either the client or server mode, using the eWay **Properties** sheet in the Enterprise Designer. For example, you can control the desired behavior of the RA. You can set whether the connection is poolable or non-poolable. You can also set the IP and port to connect to.

See the *eGate Integrator User's Guide* for details on how to use these features.

4.2.1 Customized Enveloping

The eWay properties settings 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.

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

For more information on the eWay's enveloping feature, see **"Envelope Type" on page 15**.

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

- "Custom Enveloped Class Name" on page 14
- "Customer Defined Property" on page 14

To create 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. This new class must implement the following Java interfaces provided in the eWay:
   - **com.stc.connector.tcpip.ext.msg.EnvelopedMsgReceiver**
   - **com.stc.connector.tcpip.ext.msg.EnvelopedMsgSender**
2. For your implemented Java interface (see the previous list), 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:

   edesigner\usrdir\modules\ext\tcpipextadapter

   For example:

   C:\ican50\edesigner\usrdir\ ...

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 for compiling 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 window 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.

Note: *For details on the eWay Properties window, see* **Chapter 3**.

Next, you must include your custom envelope in the current eGate Project. To do this operation, you must first create a Repository object that contains your custom envelope's .**jar** file, then add that file to the CLASSPATH of any Collaboration that uses this envelope.

To use a custom envelope in a Project

1. From the Enterprise Designer's **Project Explorer** pane, right-click the desired Project and choose **File** from the pop-up menu.
2. From the resulting dialog box, choose your custom envelope's .**jar** file (see the **procedure on page 31**) and click **Import**.

   This action creates a Repository object containing your custom envelope's .**jar** file and makes the file available in the **Project Explorer**.
3. From the Java Collaboration Editor for any Collaboration using your custom envelope, click the **File Import** button on the toolbar.
4. Again, from the resulting dialog box, choose your custom envelope's .**jar** file and click **Add**.

   This action adds your customized envelope's class to the current Collaboration's CLASSPATH.

For details on how to use the Enterprise Designer's file import features, see the *eGate Integrator User's Guide*.

Custom envelope sample

The following text provides a sample of a custom envelope:

```
package com.stc.customenvelope;
import java.io.IOException;
import java.util.StringTokenizer;
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.EnvelopedMsgReceiver;
import com.stc.connector.tcpip.ext.msg.EnvelopedMsgSender;

/**
 * Custom envelope message sample:
 * This sample demonstrates the use of custom message
 * enveloping. The sample uses the following envelope parameters:
 * offset, len, and delimeter.
 * They are specified in the eWay configuration parameter
 * as the string "offsetdelimeterlen",
 * for example, "4&10",
 * where:
 * offset = 4   len = 10   delimeter = &.
 * You can assign values to offset and len, but delimeter must be "&".
 * The eWay parses the data and assigns
 * the given values to
 * offset and len, then reads data on the socket from the "offset" position
 * until it reaches the end-of-length value "len".
 */
```

```
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 CustomEnvelopedSample.
     */
    public CustomEnvelopedSample() {

        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.


     /* The following code parses the eWay custom property given as 4&10,
      * where 4 = offset, 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 =
                "CustomEnvelopedSample.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.
          */
        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;
            }
        }
    }
```

This custom envelope is used in the eWay's sample Project. For more information, see **"Custom Envelope" on page 38**.

4.3 Overview of TCP/IP eWay Sample Project

This section explains how to implement the TCP/IP eWay using the eGate Project sample included on your installation CD-ROM. The sample is named **TCPIP_Project**. It allows you to observe an end-to-end data-exchange scenario involving eGate and the TCP/IP eWay in both client and server modes.

This section also explains how to implement this sample Project, including the TCP/IP eWay. You can also use the procedures given in this chapter to create your own Projects based on the sample provided.

To import the sample Project

1. From the eGate Enterprise Designer, right-click the desired Repository in the **Project Explorer**.
2. From the pop-up menu, choose **Import Project**.

   The **Select File to Import** dialog box appears.
3. Browse to the directory where you downloaded the sample .**zip** file while you were installing the eWay. For details on how to download this file, see the *SeeBeyond ICAN Suite Installation Guide*.

   You are looking for the container file **TCPIP_eWay_Sample.zip**. This file also contains the sample Project.

4 From the **File Destination** dialog box, select **Import to a New Project** and enter the following name for the Project:

TCPIP_Project

The Enterprise Designer imports the selected Project, and its name appears in the **Project Explorer**.

5 Before opening the new Project, click **Save All** then **Refresh All From the Repository** to refresh the Enterprise Designer.

Important: *An imported Project does not contain an environment or a deployment profile. After importing a Project, you must use the Enterprise Designer to create these functions for the Project. See* **"Creating the Project's Environment" on page 92** *and* **"Deploying a Project" on page 96***. For additional information, see the* ***eGate Integrator User's Guide*** *and* ***SeeBeyond ICAN Suite Deployment Guide****.*

You must *check out* the major eGate components before you can change them. For details, see the *eGate Tutorial*.

4.3.1 Sample Scenarios

Figure 6 shows a general diagram of how the TCP/IP eWay operates in the scenario used by the Project sample, in the client mode. This sample allows eGate to communicate with both a TCP/IP server.

Figure 6 TCP/IP Client Mode Scenario

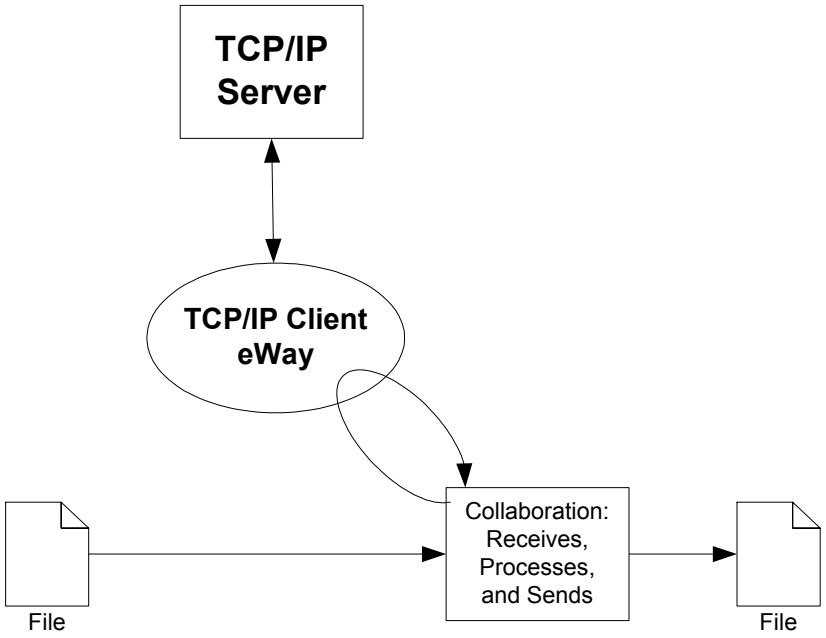
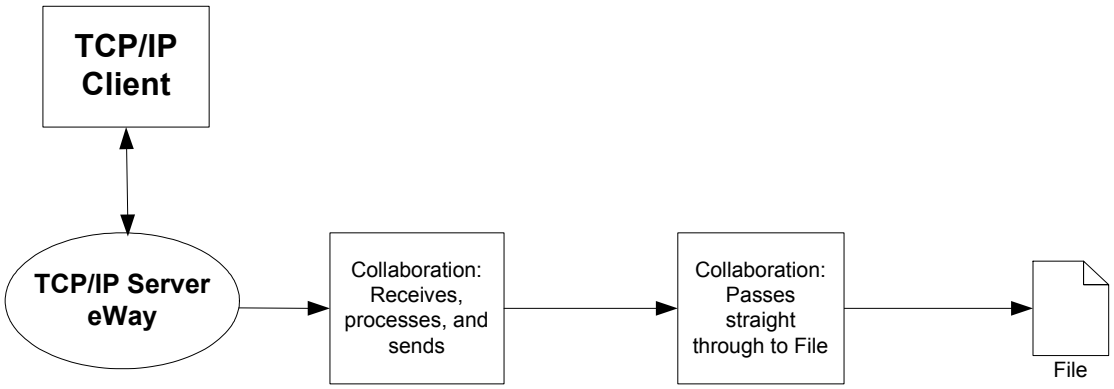


Figure 7 shows a general diagram of how the TCP/IP eWay operates in the scenario used by the Project sample, in the server mode. This sample allows eGate to communicate with both a TCP/IP client.

Figure 7 TCP/IP Server Mode Scenario



Note: *To be sure the eWay can receive multiple messages in the server mode, you must set up your server Java Collaboration Business Rules with additional looping logic to make the Collaboration longer-lived. For example, you can add a* ***while*** *loop as an initial Business Rule, which is set to make multiple tries to receive any incoming message until a quit condition is met. This condition could be, for example, a message counter that confirms the message has been received.*

4.3.2 Basic eWay Components

To use the TCP/IP eWay, you must:

- Set the desired properties settings.
- Use the TCP/IP client and server OTDs.

The sections that describe how to build the Project explain more about how to do these operations.

TCP/IP eWay Properties

The properties for the TCP/IP eWay allow you to edit values used to connect with a specific external system. These properties are set using the eGate eWay **Properties** sheet. For more information about setting TCP/IP eWay properties and the eWay **Properties** sheet, see **Chapter 3**.

TCP/IP OTDs

The TCP/IP client and server OTDs are part of the eWay and ready-made for your use in a Project. See **"Using the TCP/IP OTDs" on page 99** for an explanation of the features available to you with these OTDs.

4.3.3 Sample Project

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.

Project Components

The Project has the following components:

- Client scenario
  - Inbound file external application: **File1**
  - Inbound File eWay
  - Java Collaboration for processing data: **file2TCPIPClientCollab**
  - TCP/IP eWay
  - TCP/IP external application: **TCPIP1**
  - Outbound File eWay
  - Outbound file external application: **File2**
- Server scenario
  - Inbound TCP/IP external application: **TCPIP2**
  - TCP/IP eWay
  - Java Collaboration for processing data: **TCPIPServer2FileCollab**
  - Outbound File eWay
  - External outbound file system: **File3**

Project Operation

The client part of the Project operates as follows:

- Data enters eGate from a file system via a File eWay; information needs to be communicated to a TCP/IP server.
- The data becomes an eGate message and is sent to a Java Collaboration for processing
- The message is sent to the TCP/IP server through a TCP/IP eWay.
- The desired information is sent and received; information returns to eGate through the eWay.
- The information is processed again by the same Collaboration then written to an external file system, again via a File eWay.

The server part of the Project operates as follows:

- Data enters eGate from a TCP/IP client via the TCP/IP eWay.
- The data becomes an eGate message and is passed to a Java Collaboration where it is processed.
- The information is written to an external file system via a File eWay.

Input and Output Data

You can input any file, preferably a text file. For input purposes, you can create a text file containing any set of characters. The output data is the same file as the one input.

Handshake Protocol

The TCP/IP client checks the connection state, and if the state is null or if the state is **not connected**, the client sends the user name and password. The TCP/IP server checks the user name and password. If they are correct, the server sends a positive acknowledgement and changes the state to **connected**. Otherwise, the server sends a negative acknowledgement.

The client checks the message from the server to check whether the message is a positive or negative acknowledgement. If the state is **connected**, the client sends the data to TCP/IP server, and the server receives the data.

Custom Envelope

The sample Project uses a custom envelope. For details on the eWay's custom enveloping feature and the custom envelope used in this sample, see **"Customized Enveloping" on page 30**.

This custom envelope has two properties, offset and length. The custom envelope parses the incoming data, starting from the offset position and continuing until the stated length has been reached.

4.4 Building the Sample Project

This section explains the general steps and operations you need to follow to build the sample Project for the TCP/IP eWay.

4.4.1 Creating a New Project

To create and name a new Project

1. Start the eGate Enterprise Designer.
2. Select the Enterprise Explorer's **Project Explorer** tab, to show the **Project Explorer** pane (left pane).
3. Select the **Repository** icon on the Project Explorer tree.

4 Right-click the **Repository** and select **New** then **Project** on the pop-up menus.

A new Project appears on the **Project Explorer** tree, named **Project1**.

5 Double-click the Project name and rename the Project, for this sample, **TCPIP_Project**.

4.4.2 Creating the Connectivity Map

A Connectivity Map provides a canvas for assembling and setting up a Project's components.

To create the Connectivity Maps

1 In the eGate Enterprise Designer's **Project Explorer** pane, right-click the new Project's name and select **New** then **Connectivity Map** on the pop-up menus.

2 The new Connectivity Map appears and adds a node for the Connectivity Map under the Project on the Project Explorer tree labeled **CMap1**.

3 You can rename the new Connectivity Map **CMap1**.

The icons in the toolbar represent the components used to populate the Connectivity Map workspace. When linked together, these components define the Connectivity Map for your Project.

4.4.3 Selecting External Applications

When creating a Connectivity Map, you must associate the inbound and outbound Collaborations with an External Application. For example, to establish a connection to TCP/IP, you must first select TCP/IP as the External Application to use in your Connectivity Map.

To select external applications

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

2 Select the external applications necessary for your Project. For this sample, select the File and TCP/IP external applications. Icons representing the File and TCP/IP external applications are then added to the Connectivity Map toolbar.

4.4.4 Populating the Connectivity Map

Add the Project components to the Connectivity Map by dragging the icons from the toolbar to the canvas. This operation creates the components for you. For this sample Project, there is one Connectivity Map, **CMap1**, for both scenarios, TCP/IP client and server.

Drag and drop the following client components onto the Connectivity Map canvas:

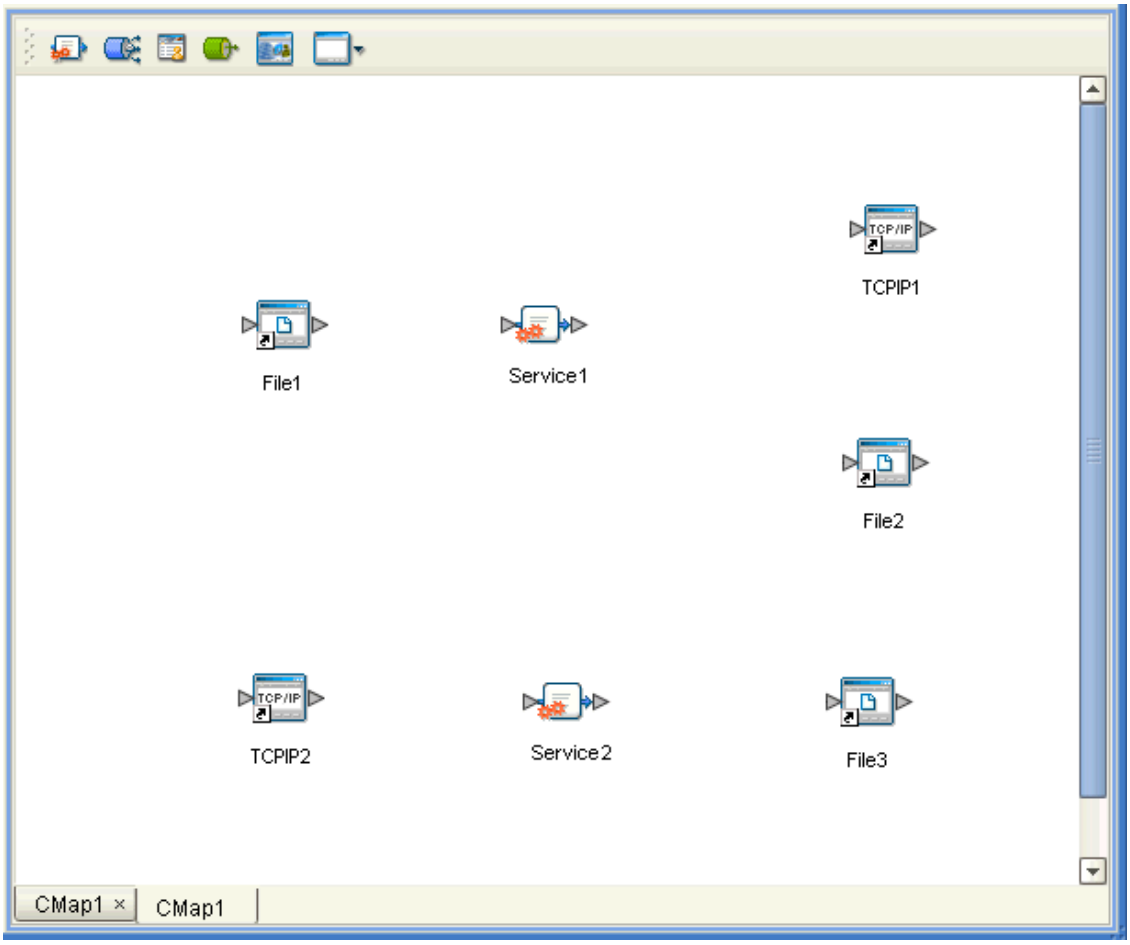
- Two File eWays/external applications named:
  - **File1**
  - **File2**
- One Service named **file2TCPIPClientCollab**
- TCP/IP eWay/external application named **TCPIP1**

Drag and drop the following server components onto the canvas:

- One File eWay/external application named **File3**
- One Service named **TCPIPServer2FileCollab**
- TCP/IP eWay/external application named **TCPIP2**

Figure 8 on page 40 shows the components on the Connectivity Map.

Figure 8 Basic Connectivity Map With Components: Client and Server



Rename the **Service1** component to **file2TCPIPClientCollab** and **Service2** to **TCPIPServer2FileCollab**. Name the other components as shown in the previous figure. Be sure to save the new Connectivity Map before you proceed. You can click the **Save** on the Enterprise Designer toolbar for this purpose.

4.4.5 Creating Java Collaboration Definitions

The next step in the sample Project is to create a Java Collaboration Definition. This component contains a set of rules that defines the processing and transport of data as it travels between eGate components.

You create Java Collaboration Definitions to assign to the Collaborations in your Project. This operation enables eGate to recognize the source and destination data relationships in your Project.

You must use the Java Collaboration Definition wizard to create a Java Collaboration Definition. After a Collaboration Definition is created, you can use the Java Collaboration Editor to create the Business Rules logic of the Collaboration.

Note: *See the **eGate Integrator User's Guide** for complete information on editing Collaborations.*

Using OTDs

An OTD contains a set of rules that define an object, which encodes data as it travels through eGate. OTDs are used as the basis for creating Collaboration Definitions for a Project. The TCP/IP eWay allows you to use its own predefined OTDs for the client and server modes (see **"Basic OTD Operation" on page 99** for details).

User-defined OTD

You can use the OTD wizard to create an eGate User-defined OTD. See the *eGate Integrator User's Guide* for a complete explanation of how to create a User-defined OTD.

To create the Java Collaboration Definitions

1. From the **Project Explorer** pane, right-click the sample Project and select **New** from the shortcut menu, and **Java Collaboration Definition** from the submenu.

   The **Java Collaboration Definition** wizard appears.

2. Enter the Collaboration Definition name, **File2TCPIPClient**, and click **Next**.

3. For the wizard's **Step 2**, you must select a Web Services Interface. From the main selection pane, click **Existing Web Service** then double-click in succession the icons for **SeeBeyond** > **eWays** > **File** > **FileClient** > **receive**.

   The **Name** text box now displays **receive**.

4. Click **Next**.

5. For the wizard's **Step 3**, you must next select the OTDs. From the main selection pane, again double-click in succession the icons for **SeeBeyond > eWays > File > FileClient**.

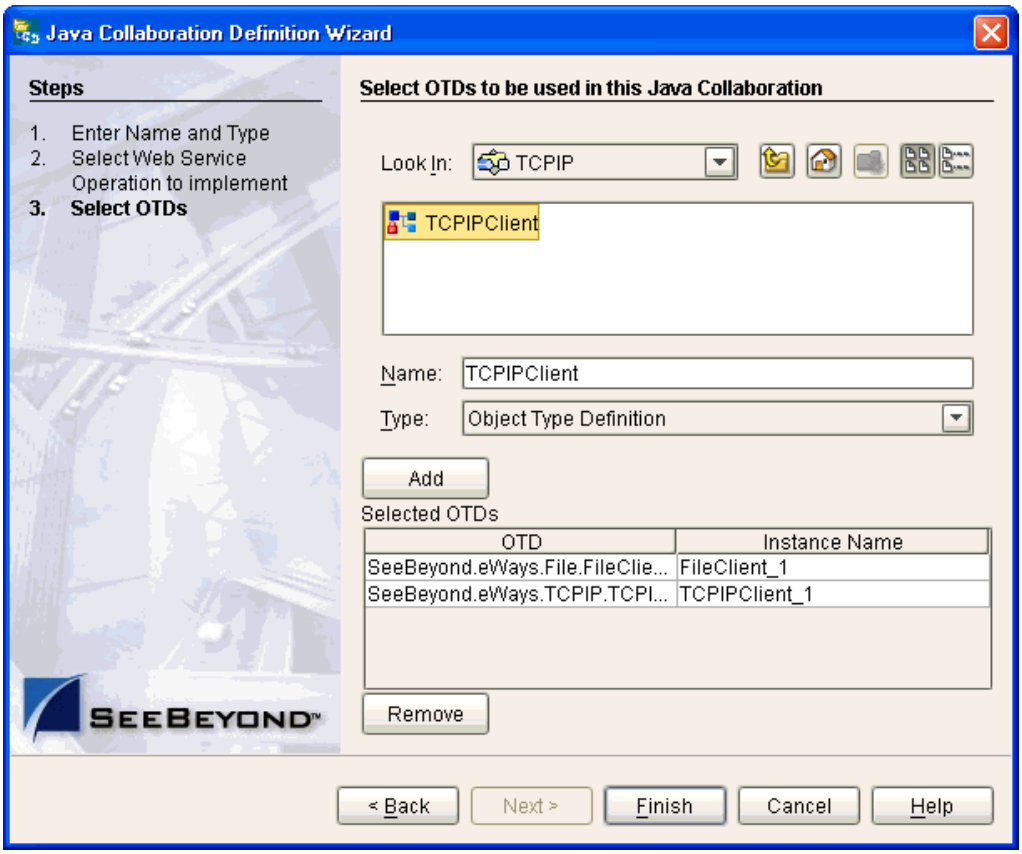
   The **Selected OTDs** pane now lists **FileClient**.

6 At the main selection pane, return to the top of the selection tree.

7 Double-click in succession the icons for **SeeBeyond > eWays > TCPIP > TCPIPClient**.

See **Figure 8 on page 40**.

Figure 9 Java Collaboration Definition Wizard With OTDs



8 Click **Finish** to create the Java Collaboration Definition **File2TCPIPClientCollab**.

The Java Collaboration Editor with the new **File2TCPIPClient** Collaboration Definition icon appears in the **Project Explorer** pane.

9 From the **Project Explorer** pane, for the next Java Collaboration Definition, display the **Java Collaboration Definition** wizard again.

10 Enter the Collaboration Definition name, **TCPIPServer2File**, and click **Next**.

11 For the wizard's **Step 2**, click **Existing Web Service** then double-click in succession the icons for **SeeBeyond** > **eWays** > **TCPIP** > **TCPIPClient** > **receive**.

The **Name** text box now displays **receive**.

12 Click **Next**.

13 For the wizard's **Step 3**, to select the OTD, double-click in succession the icons for **SeeBeyond > eWays > File > FileClient**.

The **Selected OTDs** pane now lists **FileClient**.

14 Click **Finish** to create the Java Collaboration Definition **TCPIPServer2File**.

The Java Collaboration Editor with the new **TCPIPServer2File** Collaboration Definition icon appears in the **Project Explorer** pane.

4.4.6 Creating Business Rules Within Collaboration Definitions

You can customize Collaboration Definitions by mapping Business Rules using the Java Collaboration Editor window in the Enterprise Designer. By clicking and dragging, you map items in the Transformation Designer areas with other desired items via their icons.

Using the Java Collaboration Editor

This section explains generally how to create the Business Rules used in the sample Project.

The Java Collaboration Editor window displays in the Enterprise Designer after you create a new Java Collaboration Definition. You can also open this Editor by right-clicking on the name of the desired Collaboration Definition in the **Project Explorer** and choosing **Open** from the pop-up menu.

See the *eGate Integrator User's Guide* for complete information on how to use the Java Collaboration Editor. To complete a Collaboration Definition, you can use the Java Collaboration Editor to create Business Rules.

To create Business Rules within File2TCPIPClient

1 From the **Project Explorer** tree, double-click the **File2TCPIPClient** Collaboration Definition.

The Java Collaboration Editor displays **File2TCPIPClient**.

2 Expand all the nodes in the left pane of the Transformation Designer. Also, expand all the nodes in the **Business Rules** pane.

There are 12 Business Rules under the **receive()** method in the **Business Rules** pane.

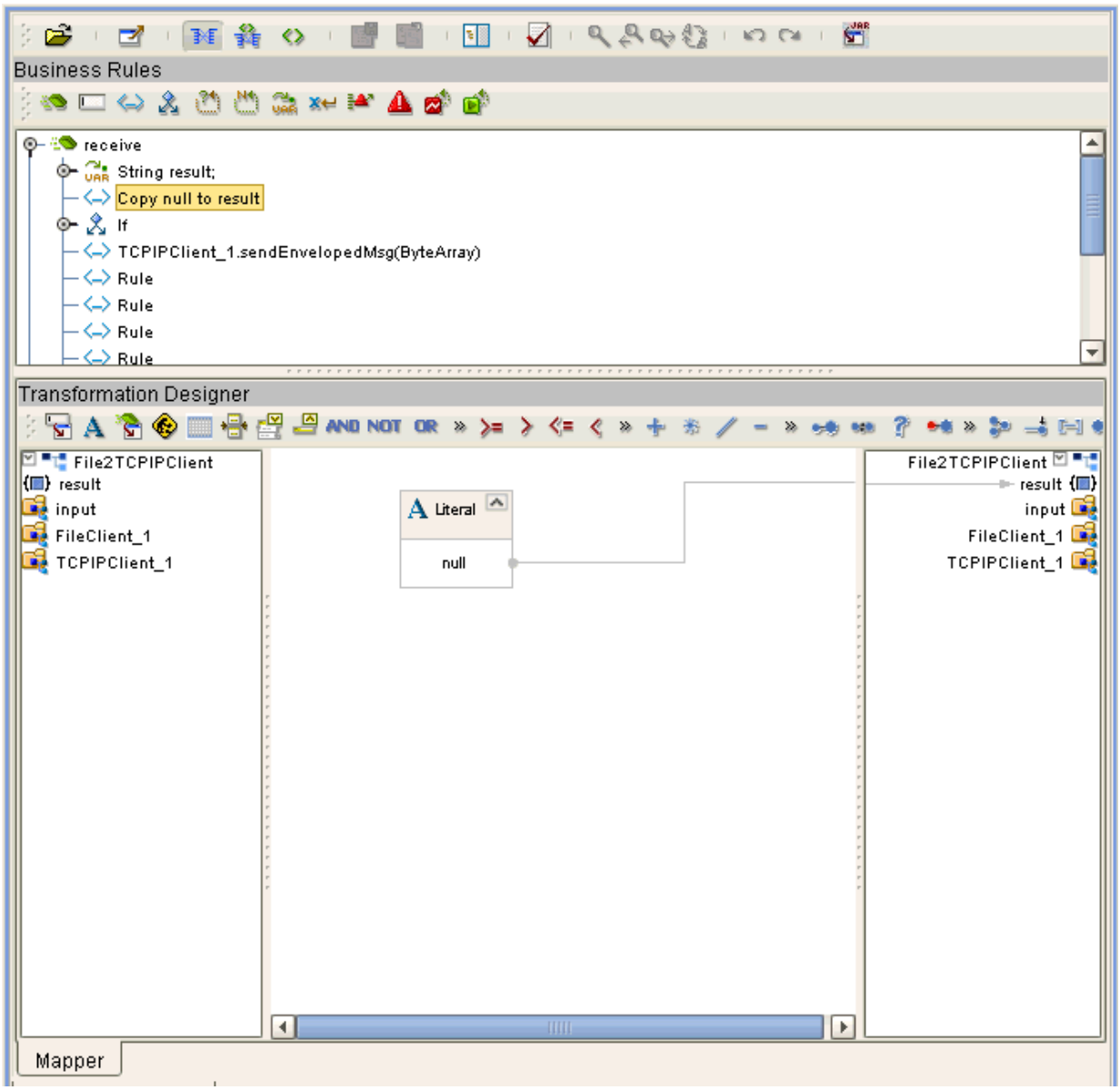
3 For the first Business Rule, create a local variable **String result** in the Business Rules pane.

4 Enter **initializer** in the **Local Variable** dialog box.

5 For the second Business Rule, create a null **Literal** as a **rule**, and drag the **null** to the **result** node in the right pane.

This action creates the **Copy null to result** Business Rule. See Figure 10.

Figure 10 File2TCPIPClient Collaboration Definition: Second Business Rule



6 Click **Save** on the Enterprise Designer toolbar to save your changes.

Note: *Saving your Java Collaboration Definition automatically validates it. You can ignore any error messages until you are finished with the Collaboration.*

7 For the third Business Rule, create an **If rule**.

Note: *Always create new Business Rules in the Business Rules pane.*

8 Under the **If rule**, for the **condition**, create the following boxes in the Transformation Designer:

- **Literal:** string type, string: **Connected**
- **Literal:** null type
- **equal** (comparison operator)
- **not equal** (comparison operator)
- **OR** (Boolean operator)

9 Position the boxes in the Transformation Designer as shown in **Figure 11 on page 46**.

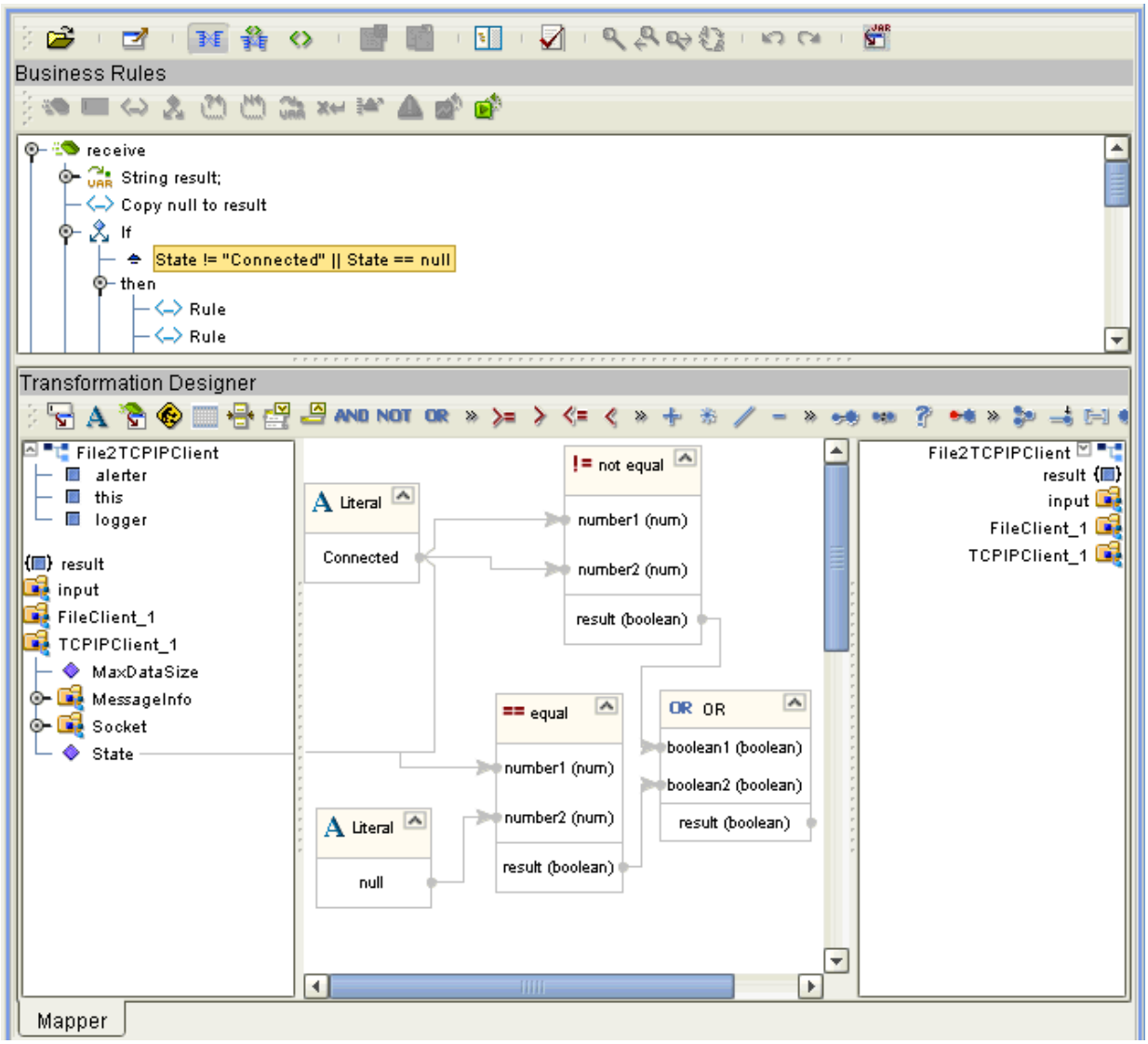
10 Create the following connections between the boxes (shown in the previous list) in the Transformation Designer:

- Drag the **State** node under **Socket** in the left pane to **number1 (num)** under **not equal**, then drag the same node to **number1 (num)** under **equal**.
- Drag **Connected** under its **Literal** to **number2 (num)** under **not equal**.
- Drag **null** under its **Literal** to **number2 (num)** under **equal**.
- Drag **null** under its **Literal** to **number2 (num)** under **equal**.
- Drag **result (boolean)** under **not equal** to **boolean1 (boolean)** under **OR**.
- Drag **result (boolean)** under **equal** to **boolean2 (boolean)** under **OR**.

Make sure the resulting window appears as shown in **Figure 11 on page 46**.

Note: *Instructions from step 7 to step 67 are for this third Business Rule.*

Figure 11 File2TCPIPClient Collaboration Definition: If Rule Condition



11 Click **Save** to save your changes.

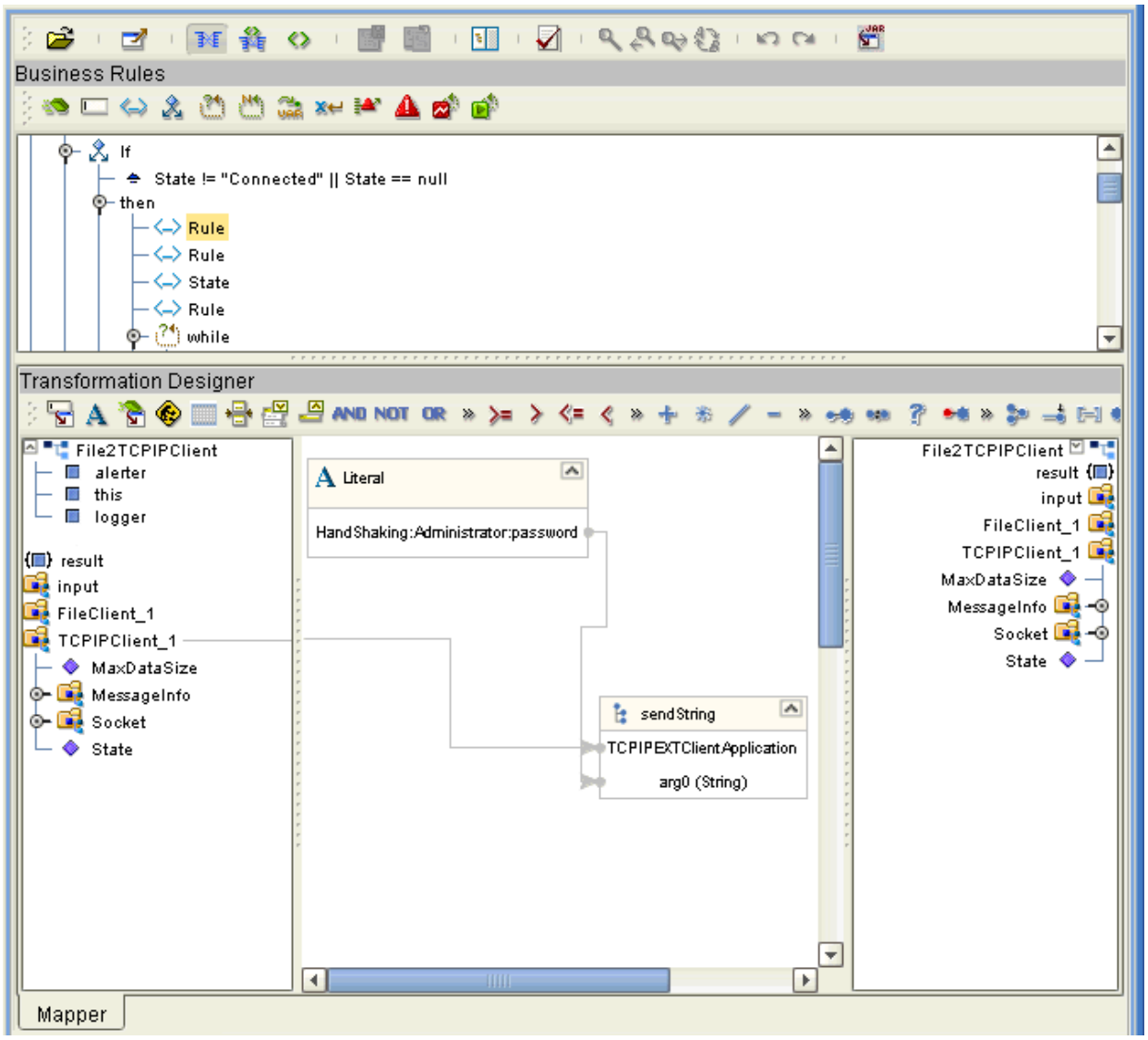
12 Under **then**, create a new **rule**.

13 Create a method, **sendString**, from the **TCPIPClient_1** node in the left pane.

14 Create a string **Literal** with the string **HandShaking.Administrator.password**.

15 From the Literal box in the Transformation Designer, drag the string (**HandShaking.Administrator.password**) to **arg0 (String)** under the method **sendString**. See Figure 12.

Figure 12 File2TCPIPClient Collaboration Definition: then Rule 1

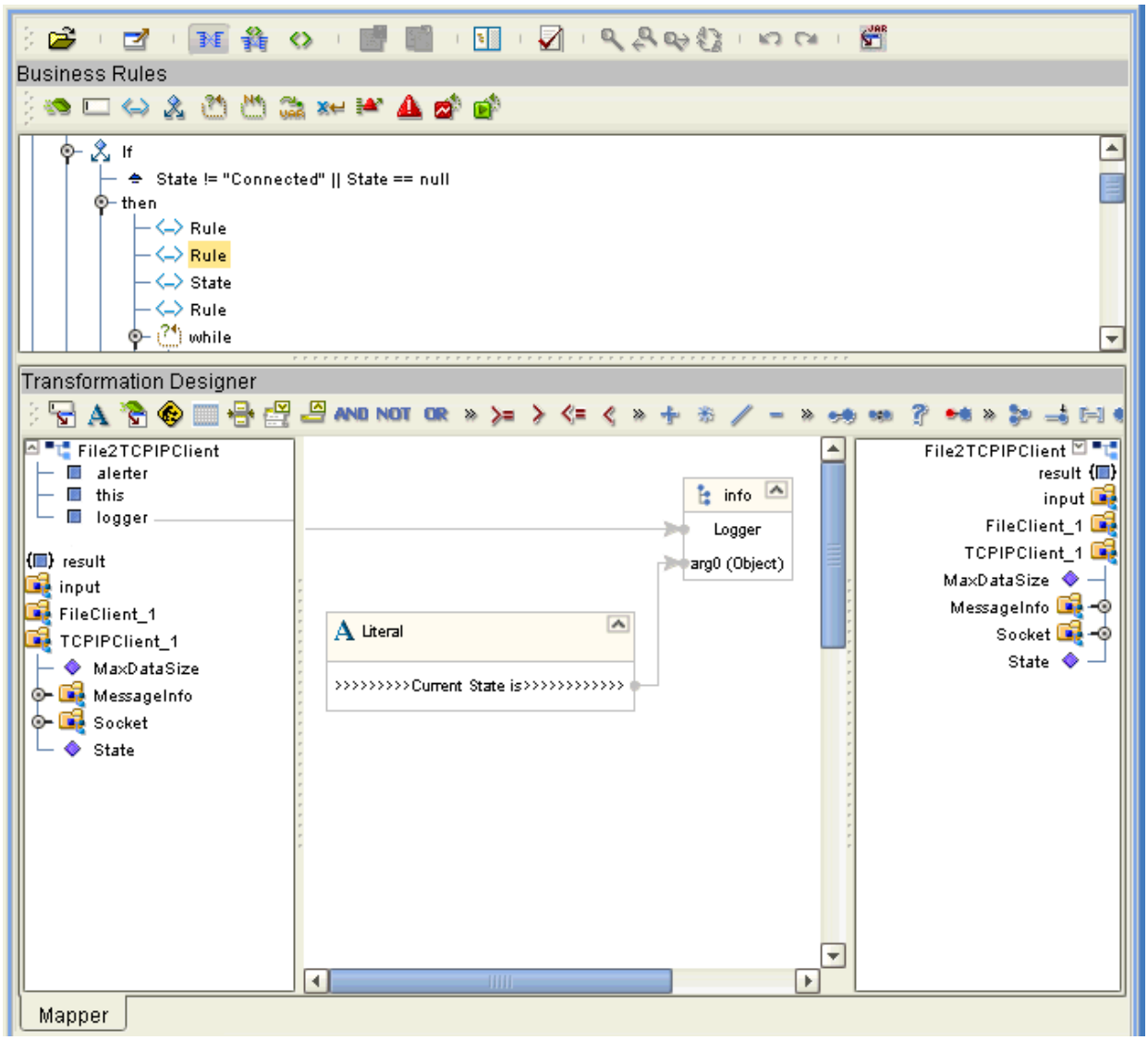


16 From the **logger** node under **File2TCPIPClient**, create the method **info**.

17 Create a string **Literal** with the string **Current State is** including angle brackets, as shown in Figure 13.

18 From the Literal box in the Transformation Designer, drag the string (**Current State is**) to **arg0 (Object)** under **info**. See Figure 13.

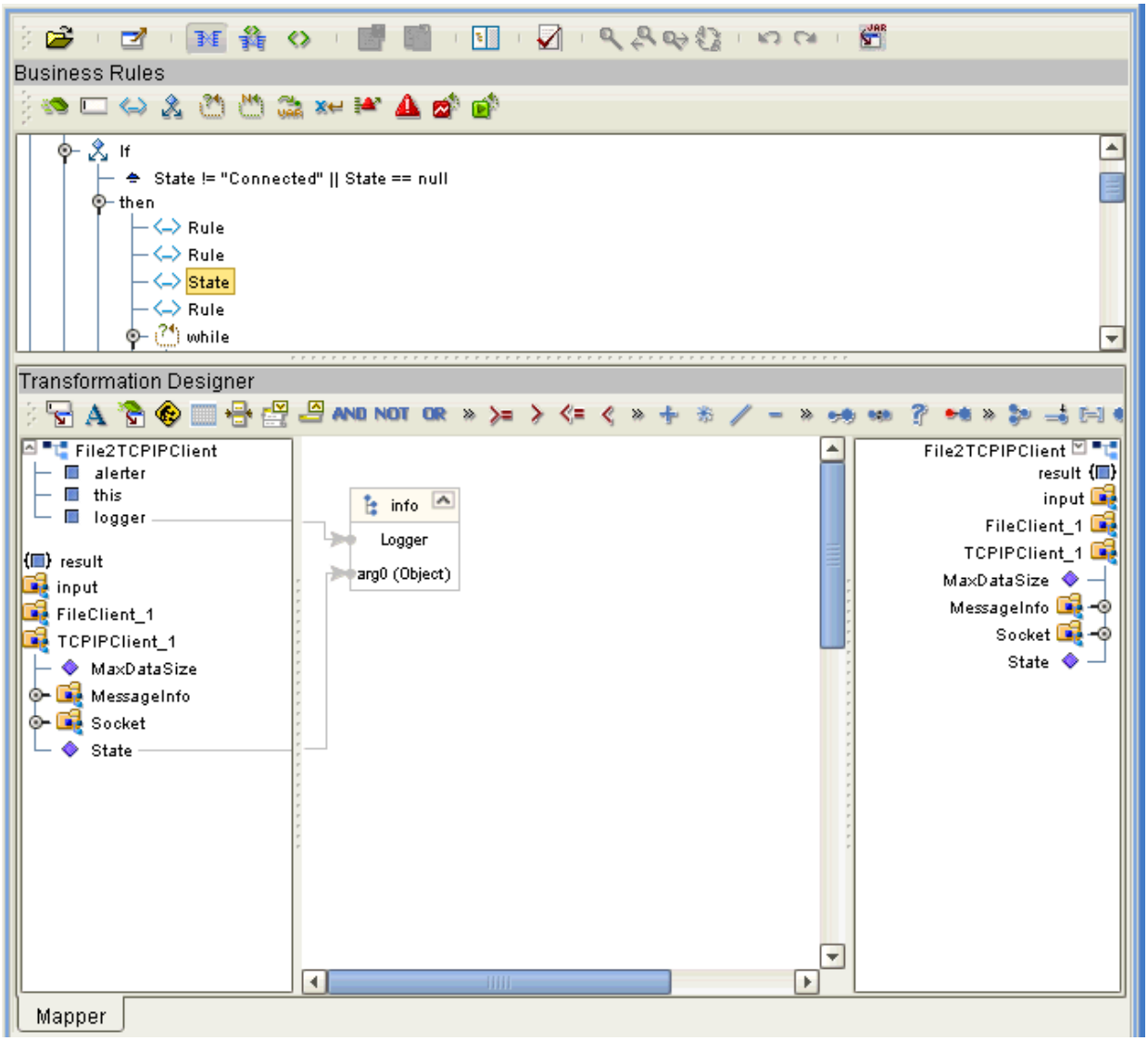
Figure 13 File2TCPIPClient Collaboration Definition: then Rule 2



19 From the **logger** node under **File2TCPIPClient**, create the method **info**.

20 Drag the **State** node under **Socket** in the left pane to **arg0 (Object)** in the **info Method** box. See Figure 14.

Figure 14 File2TCPIPClient Collaboration Definition: then State



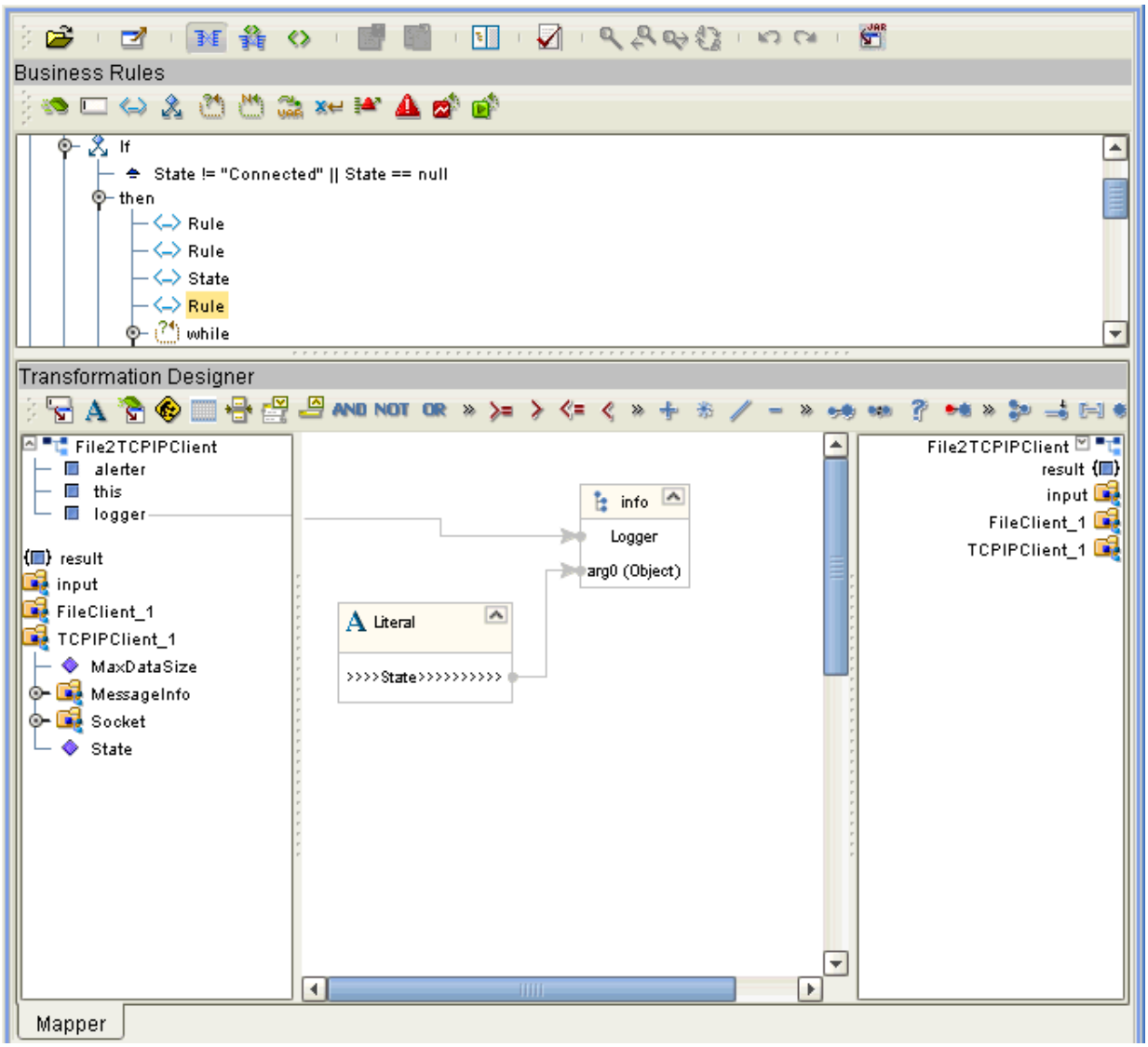
21 Click **Save** to save your changes.

22 From the **logger** node under **File2TCPIPClient**, create the method **info**.

23 Create a string **Literal** with the string **State** including angle brackets, as shown in Figure 15.

24 From the Literal box in the Transformation Designer, drag the string (**State**) to **arg0 (Object)** under **info**. See Figure 15.

Figure 15 File2TCPIPClient Collaboration Definition: then Rule 3



25 Under the current **If rule**, create a **while** rule.

26 Create the following boxes in the Transformation Designer:

- **Literal:** null type
- **Literal: int** type, integer: **0**
- **equal** (comparison operator)
- Another **equal** (comparison operator)
- **OR** (Boolean operator)

27 In addition, from the **result** node under **File2TCPIPClient**, create the method **length**.

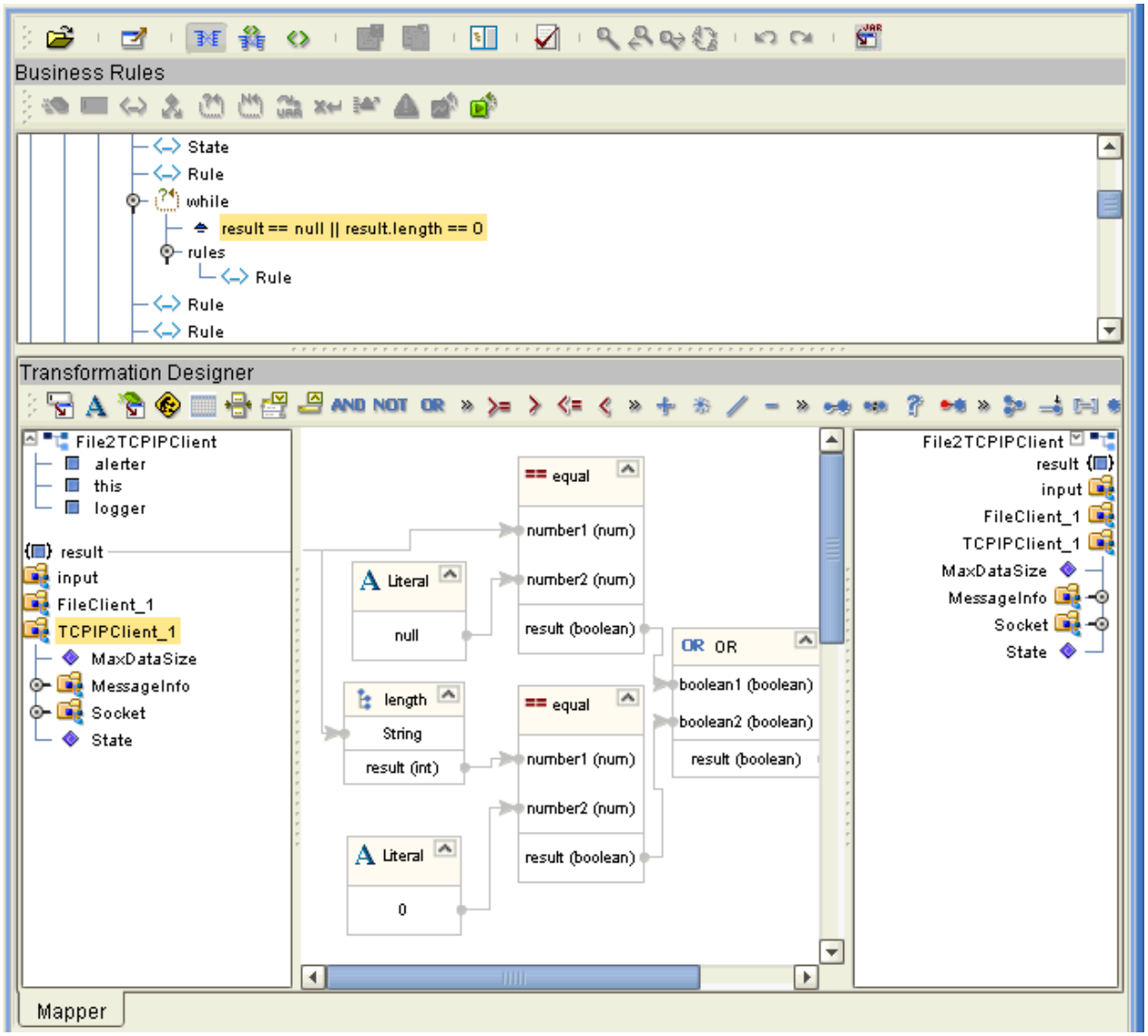
28 Position the boxes in the Transformation Designer as shown in **Figure 16 on page 52**.

29 Create the following connections between the boxes (shown in the previous list), including the **Method** box, in the Transformation Designer:

- Drag the **result** node under **File2TCPIPClient** to **number1 (num)** under **equal** (the upper box).
- Drag **null** under its **Literal** to **number2 (num)** under **equal** (the upper box).
- Drag **0** under its **Literal** to **number2 (num)** under **equal** (the lower box).
- Drag **result(int)** under the **length** method to **number1 (num)** under **equal** (the lower box).
- Drag **result (boolean)** under **equal** (the upper box) to **boolean1 (boolean)** under **OR**.
- Drag **result (boolean)** under **equal** (the lower box) to **boolean2 (boolean)** under **OR**.

Make sure the resulting window appears as shown in **Figure 16 on page 52**.

Figure 16 File2TCPIPClient Collaboration Definition: while Rule

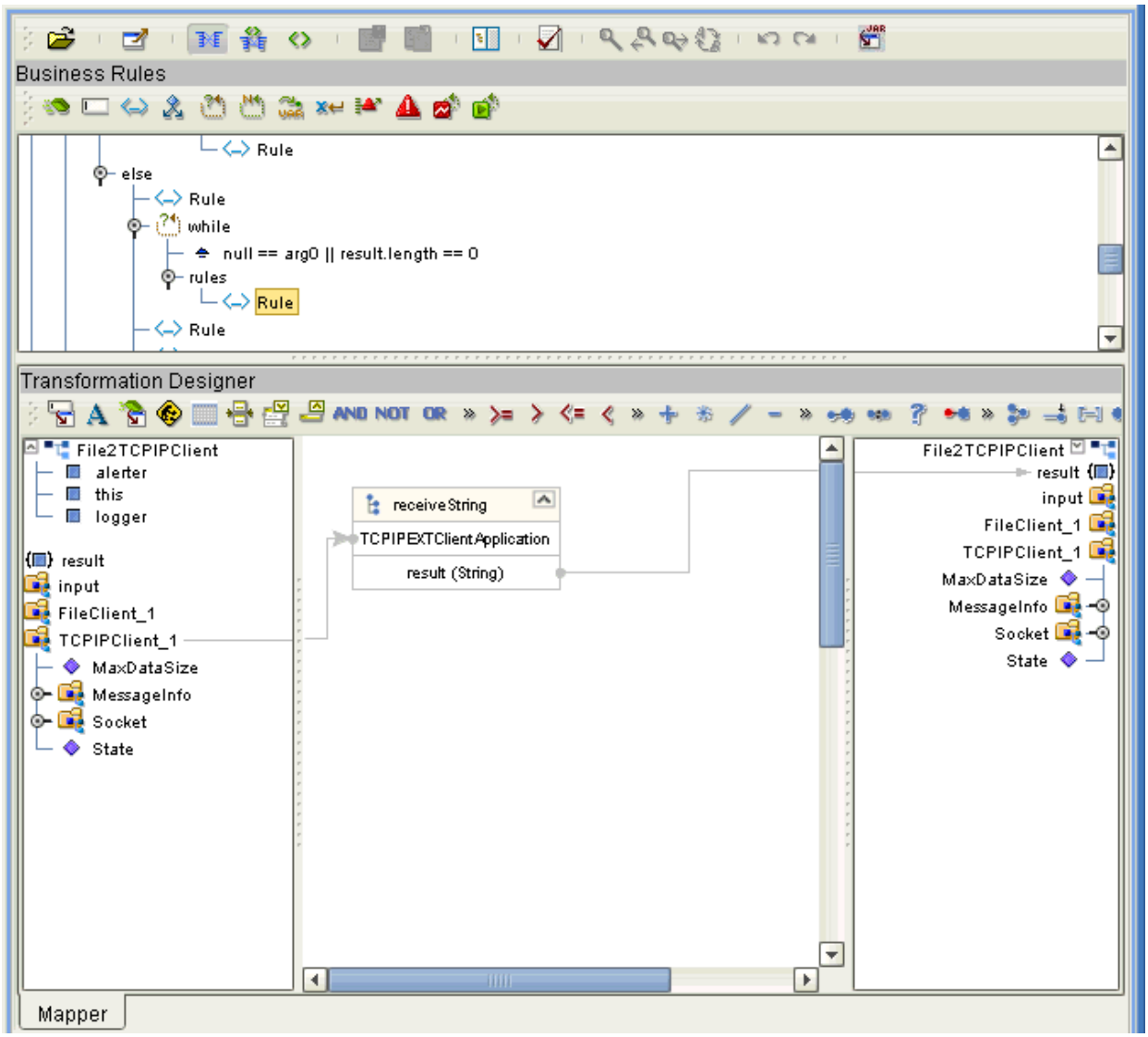


30 Click **Save** to save your changes.

31 From the **TCPIPClient_1** node in the left pane, create the method **receiveString**.

32 Drag the **result (String)** from the **Method** box to the **result** node under **File2TCPIPClient** in the right pane, as shown in Figure 17.

Figure 17 File2TCPIPClient Collaboration Definition: while rules Rule

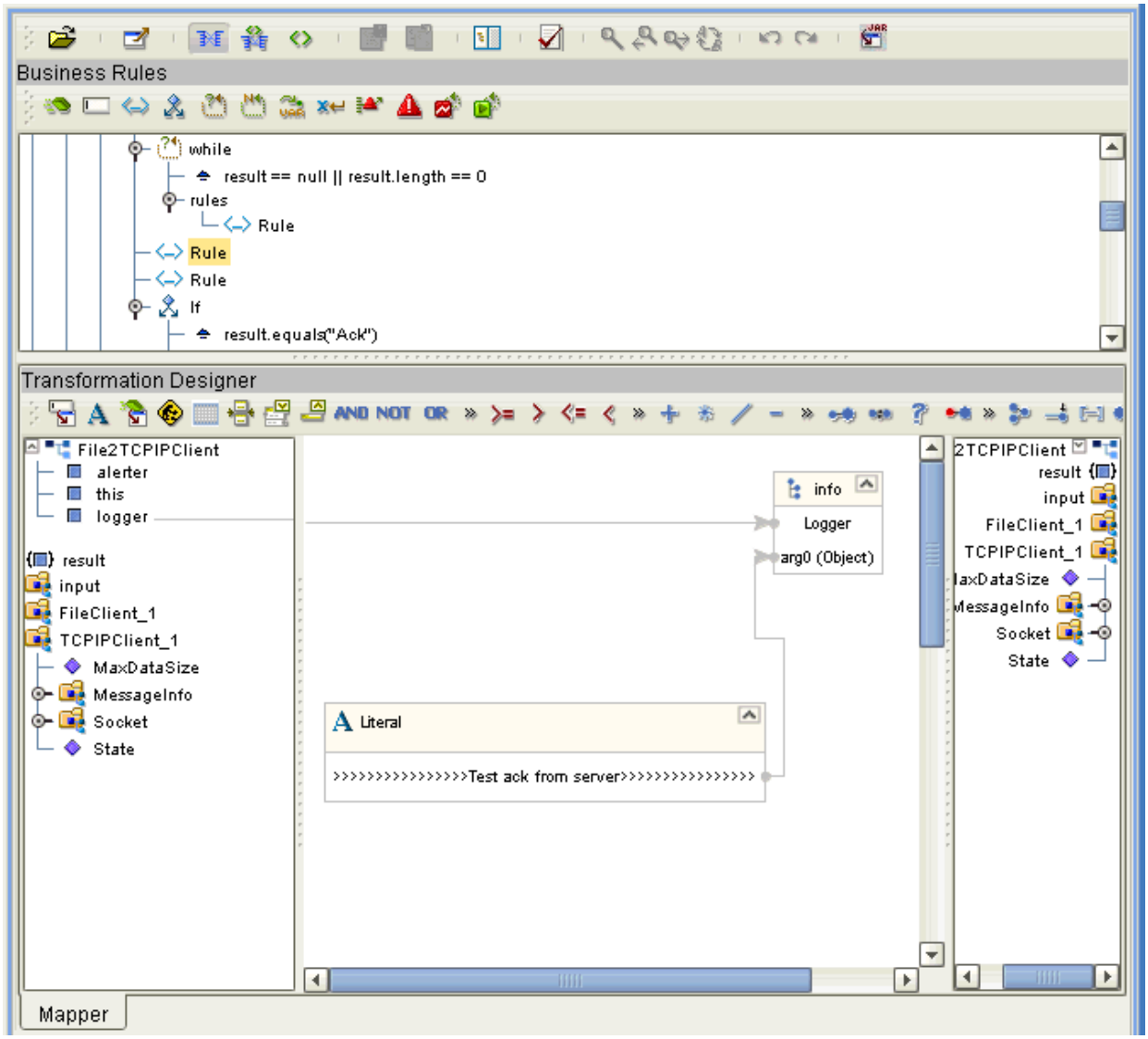


33 From the **logger** node under **File2TCPIPClient** in the left pane, create the method **info**.

34 Create a string **Literal** with the string **Test ack from server** including angle brackets, as shown in Figure 18.

35 From the Literal box in the Transformation Designer, drag the string (**Test ack from server**) to **arg0 (Object)** under **info**. See Figure 18.

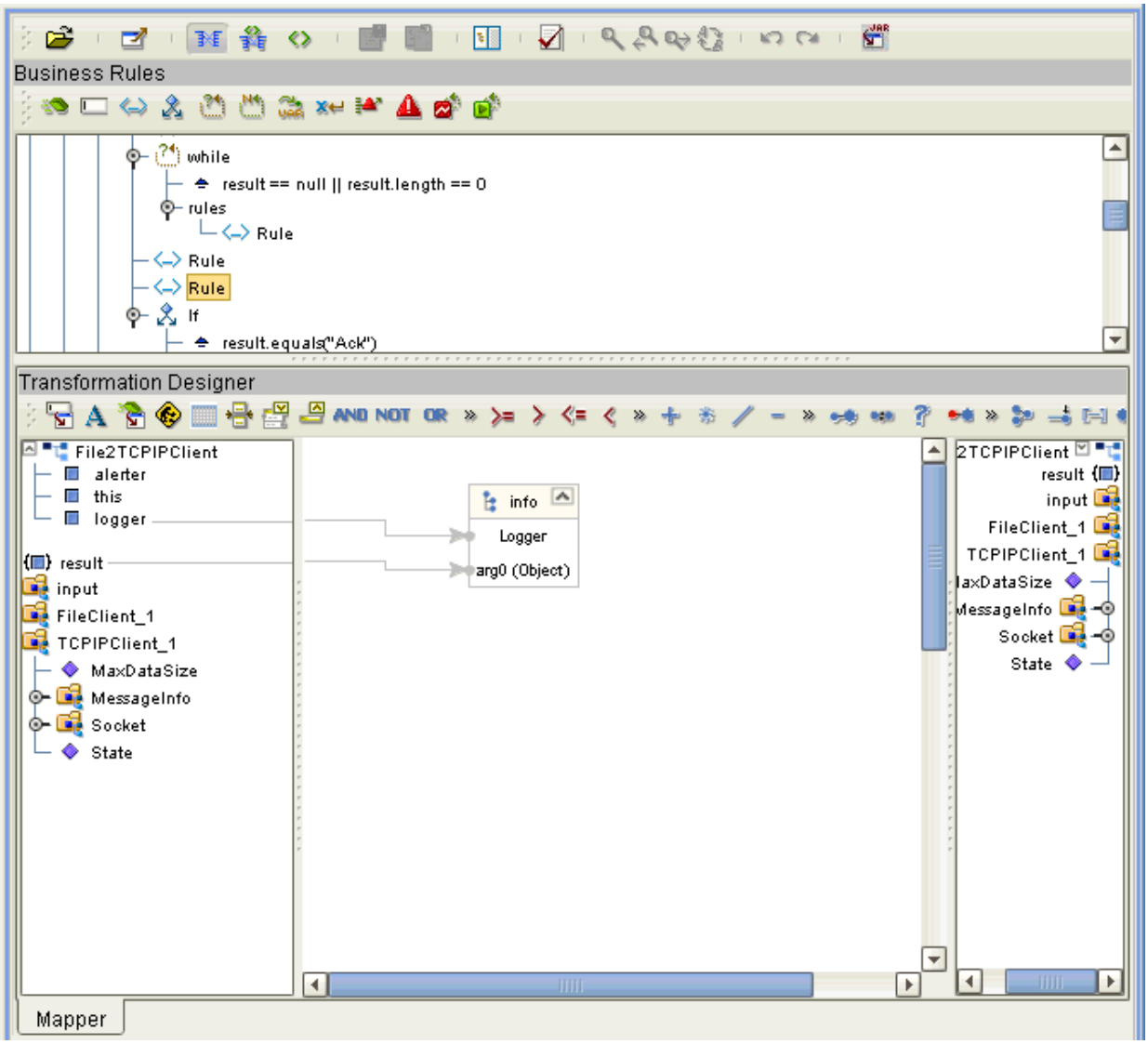
Figure 18 File2TCPIPClient Collaboration Definition: Rule Under while Rule 1



36 From the **logger** node under **File2TCPIPClient** in the left pane, create the method **info**.

37 Drag the **result** node in the left pane to **arg0 (Object)** under the **info** method, as shown in Figure 19.

Figure 19 File2TCPIPClient Collaboration Definition: Rule Under while Rule 2



38 Click **Save** to save your changes.

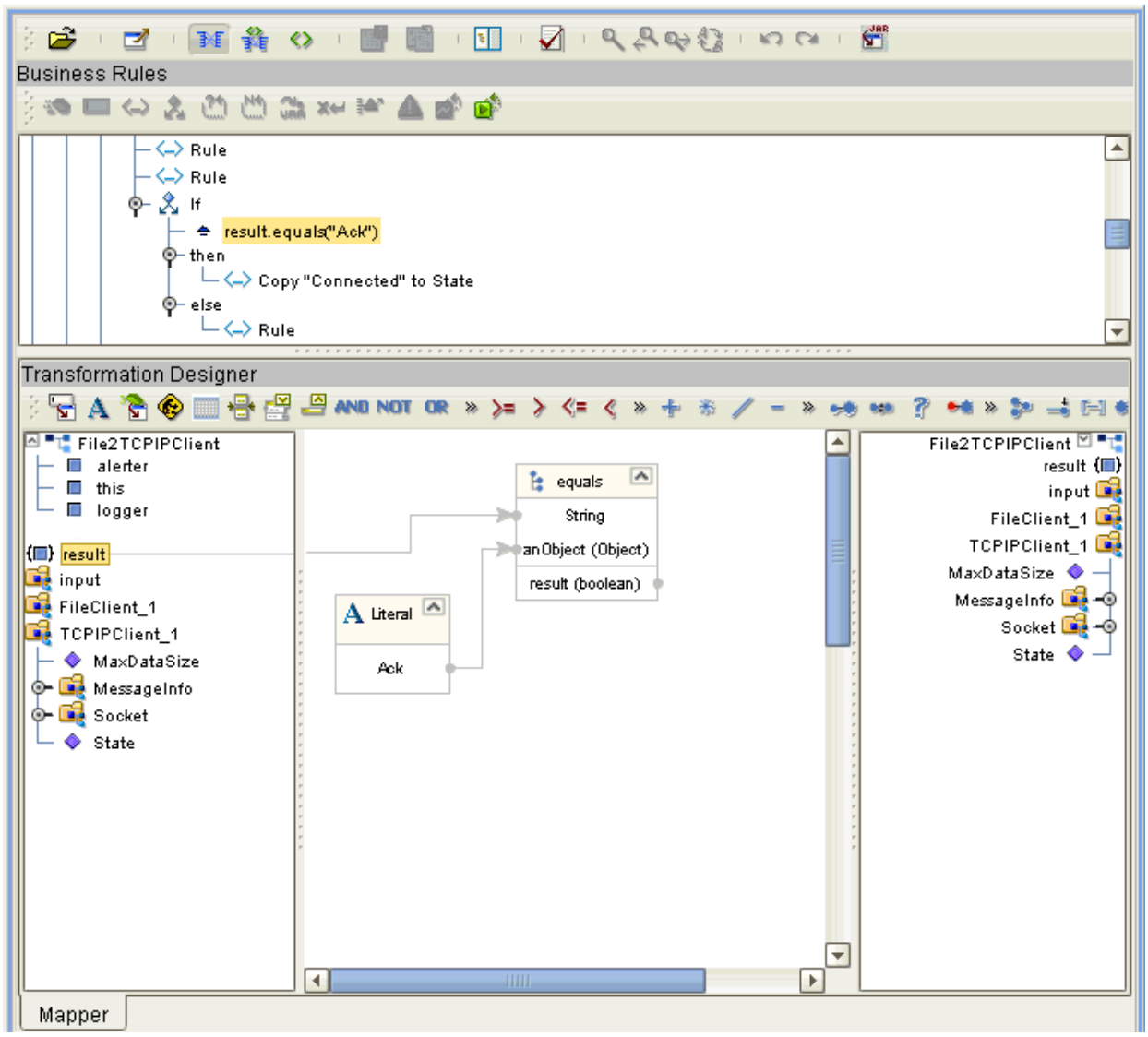
39 For the next Business Rule, create a (nested) **If rule** under the previous **rule**.

40 Create a string **Literal** with the string **Ack**, as shown in Figure 20.

41 From the **result** node in the left pane, create an **equals** method.

42 Drag the string (**Ack**) to **an Object (Object)** under the **equals** method. See Figure 20.

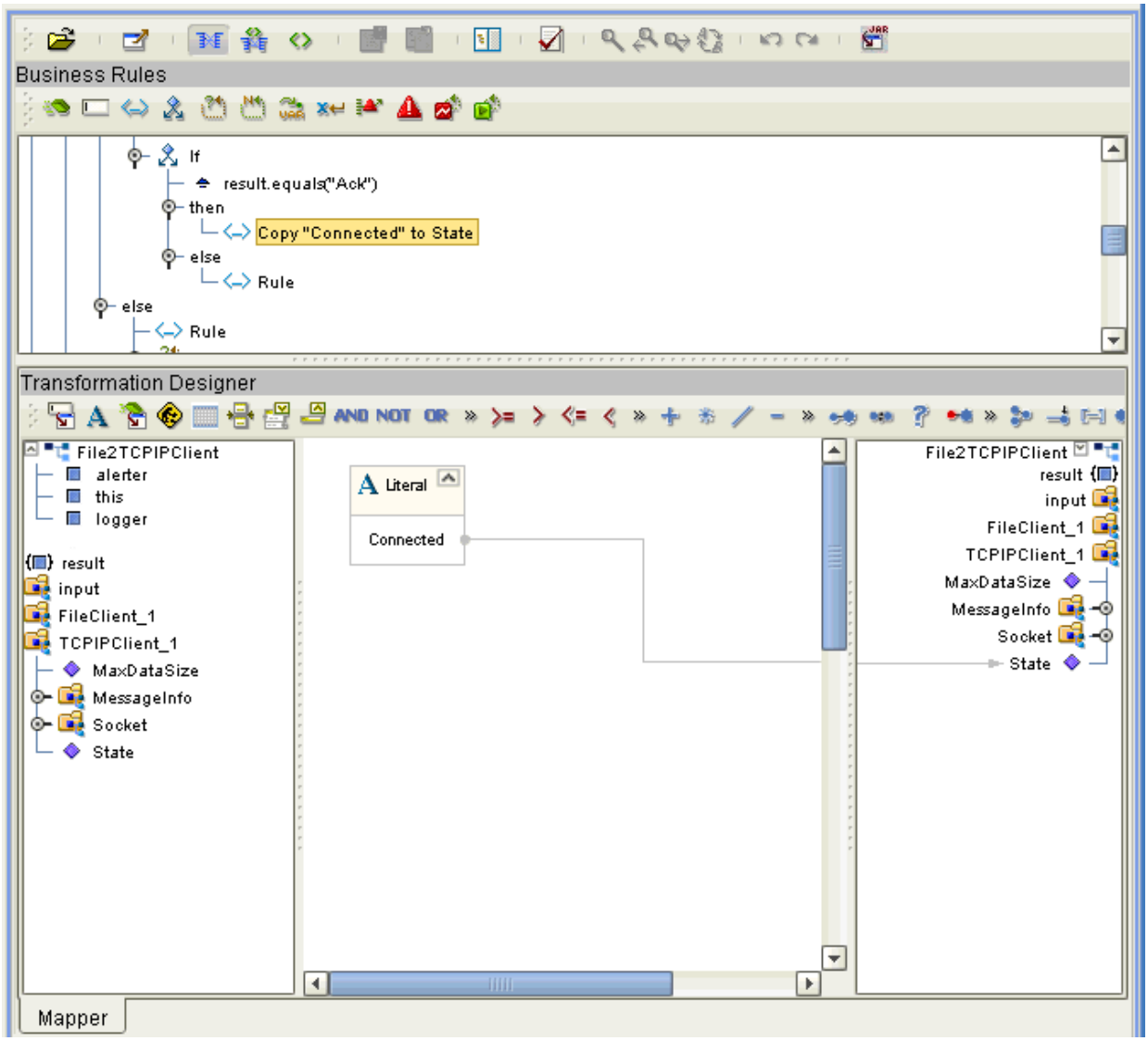
Figure 20 File2TCPIPClient Collaboration Definition:



43 For a **rule** under **then**, create a string **Literal** with the string **Connected**, as shown in Figure 21.

44 Drag the string (**Connected**) to the **State** node under **Socket** in the right pane. See Figure 21.

Figure 21 File2TCPIPClient Collaboration Definition: Rule Under then

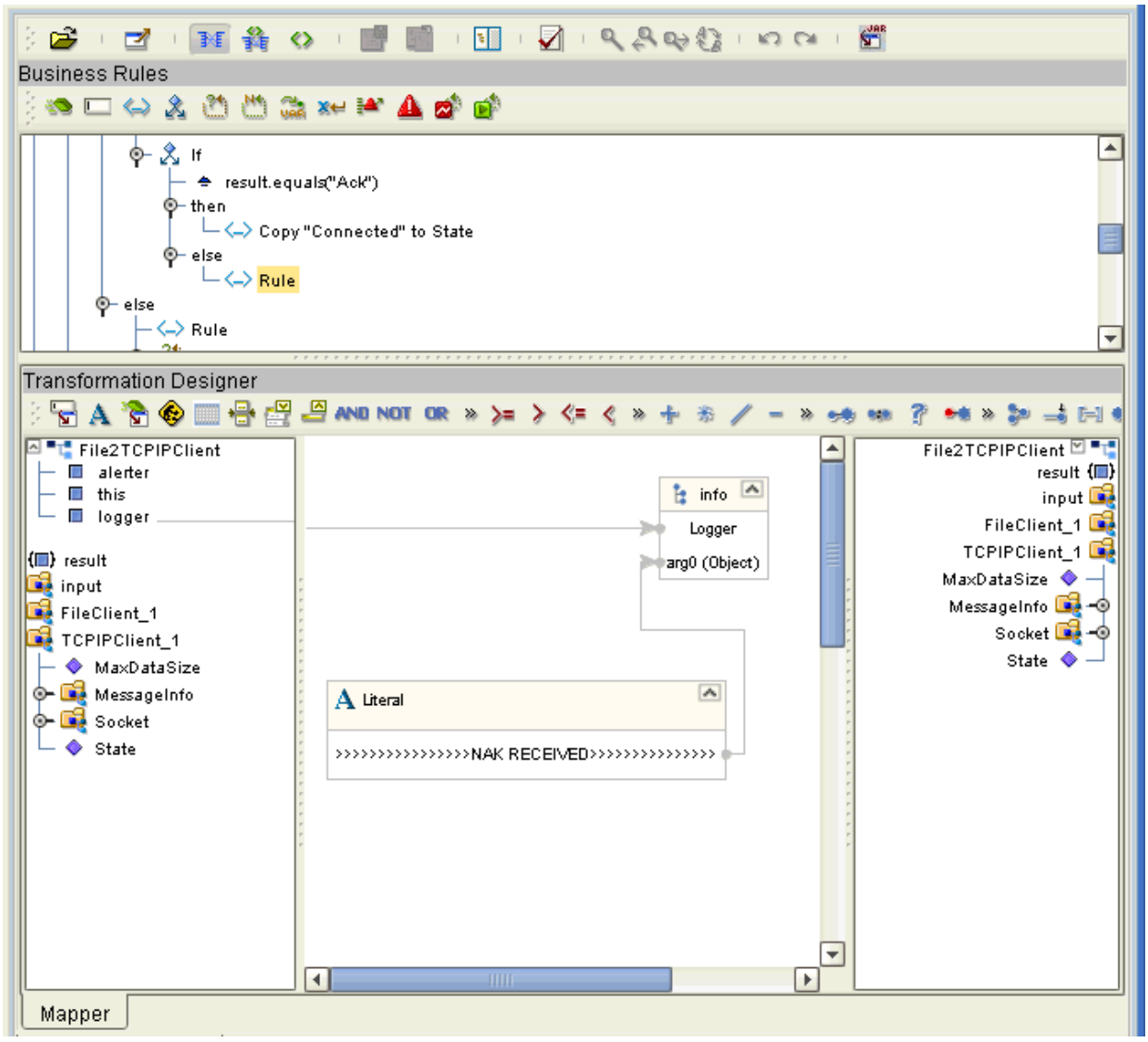


45 For a **rule** under **else**, start from the **logger** node under **File2TCPIPClient** in the left pane and create the method **info**.

46 Create a string **Literal** with the string **NAK RECEIVED** including angle brackets, as shown in Figure 22.

47 Drag the string (**NAK RECEIVED**) to **arg0 (Object)** under the **info** method. See Figure 22.

Figure 22 File2TCPIPClient Collaboration Definition: Rule Under else



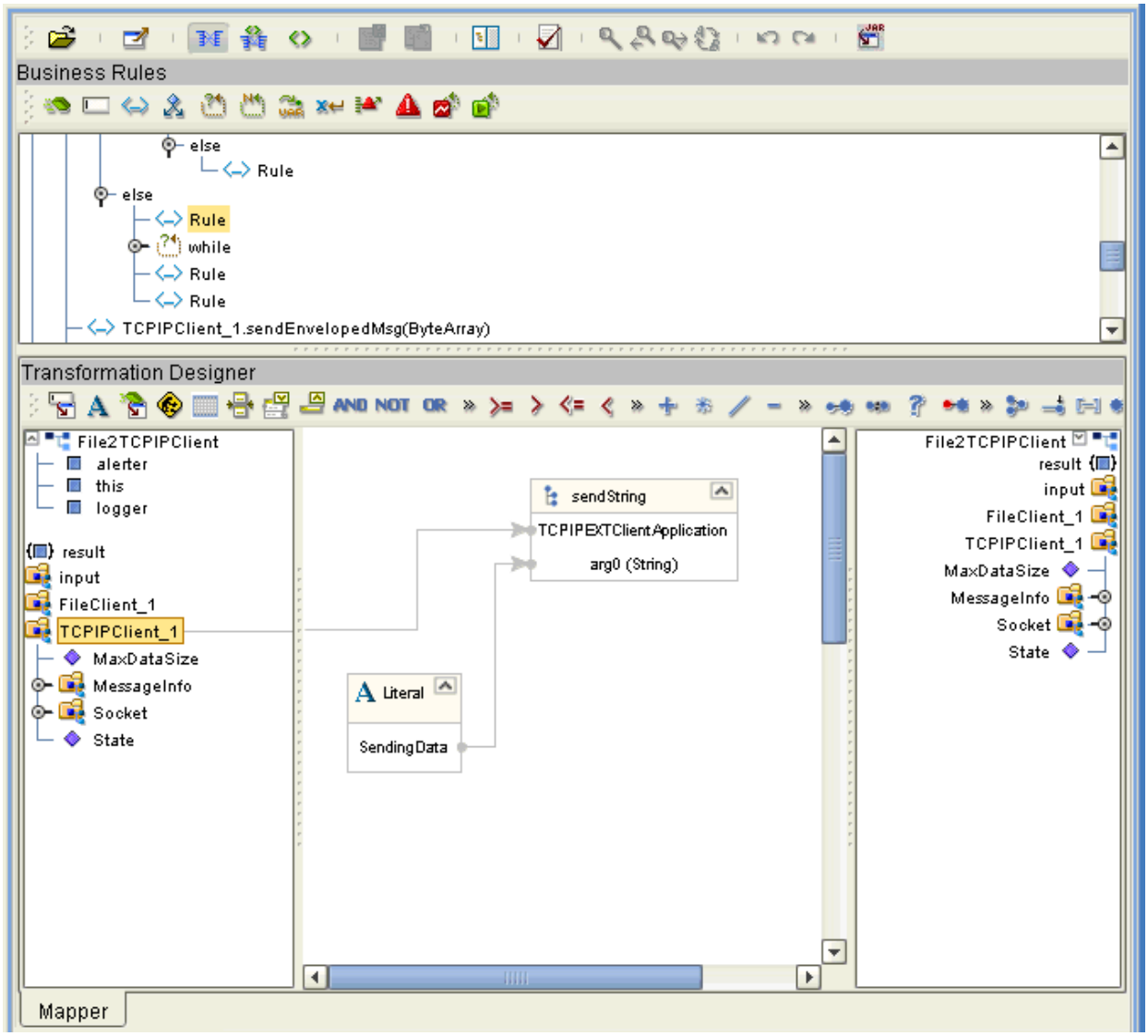
48 Click **Save** to save your changes.

49 Create a method, **sendString**, from the **TCPIPClient_1** node in the left pane.

50 Create a string **Literal** with the string **Sending Data**.

51 From the Literal box in the Transformation Designer, drag the string (**Sending Data**) to **arg0 (String)** under the **sendString** method. See Figure 23.

Figure 23 File2TCPIPClient Collaboration Definition:



52 Under the current **else**, create a (nested) **while** rule.

53 Create the following boxes in the Transformation Designer:

- **Literal:** null type
- **Literal: int** type, integer: **0**
- **equal** (comparison operator)
- Another **equal** (comparison operator)
- **OR** (Boolean operator)

54 In addition, from the **result** node under **File2TCPIPClient**, create the method **length**.

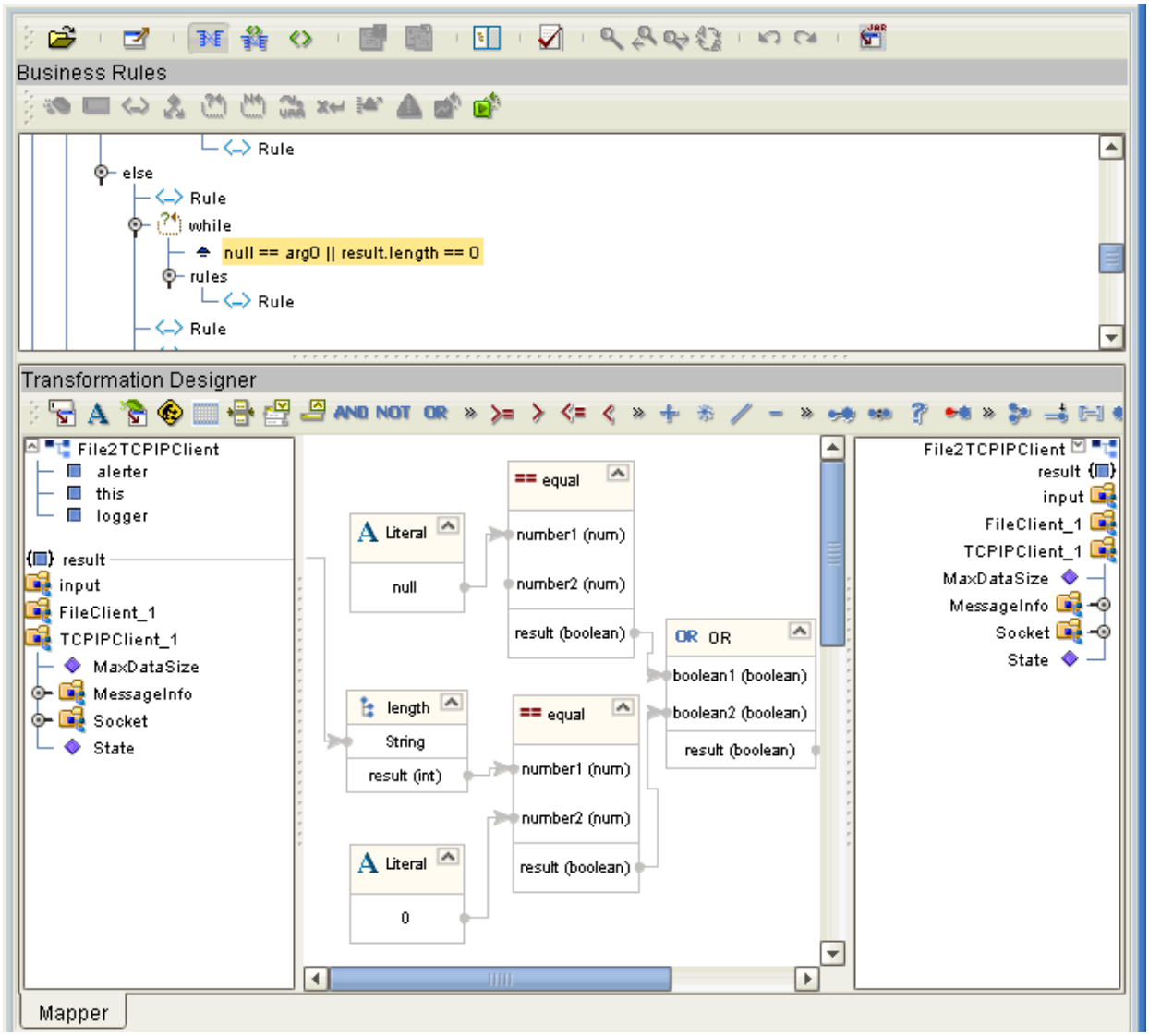
55 Position the boxes in the Transformation Designer as shown in **Figure 24 on page 61**.

56 Create the following connections between the boxes (shown in the previous list), including the **Method** box, in the Transformation Designer:

- Drag **null** under its **Literal** to **number1 (num)** under **equal** (the upper box).
- Drag **0** under its **Literal** to **number2 (num)** under **equal** (the lower box).
- Drag **result(int)** under the **length** method to **number1 (num)** under **equal** (the lower box).
- Drag **result (boolean)** under **equal** (the upper box) to **boolean1 (boolean)** under **OR**.
- Drag **result (boolean)** under **equal** (the lower box) to **boolean2 (boolean)** under **OR**.

Make sure the resulting window appears as shown in **Figure 24 on page 61**.

Figure 24 File2TCPIPClient Collaboration Definition: Nested while rule

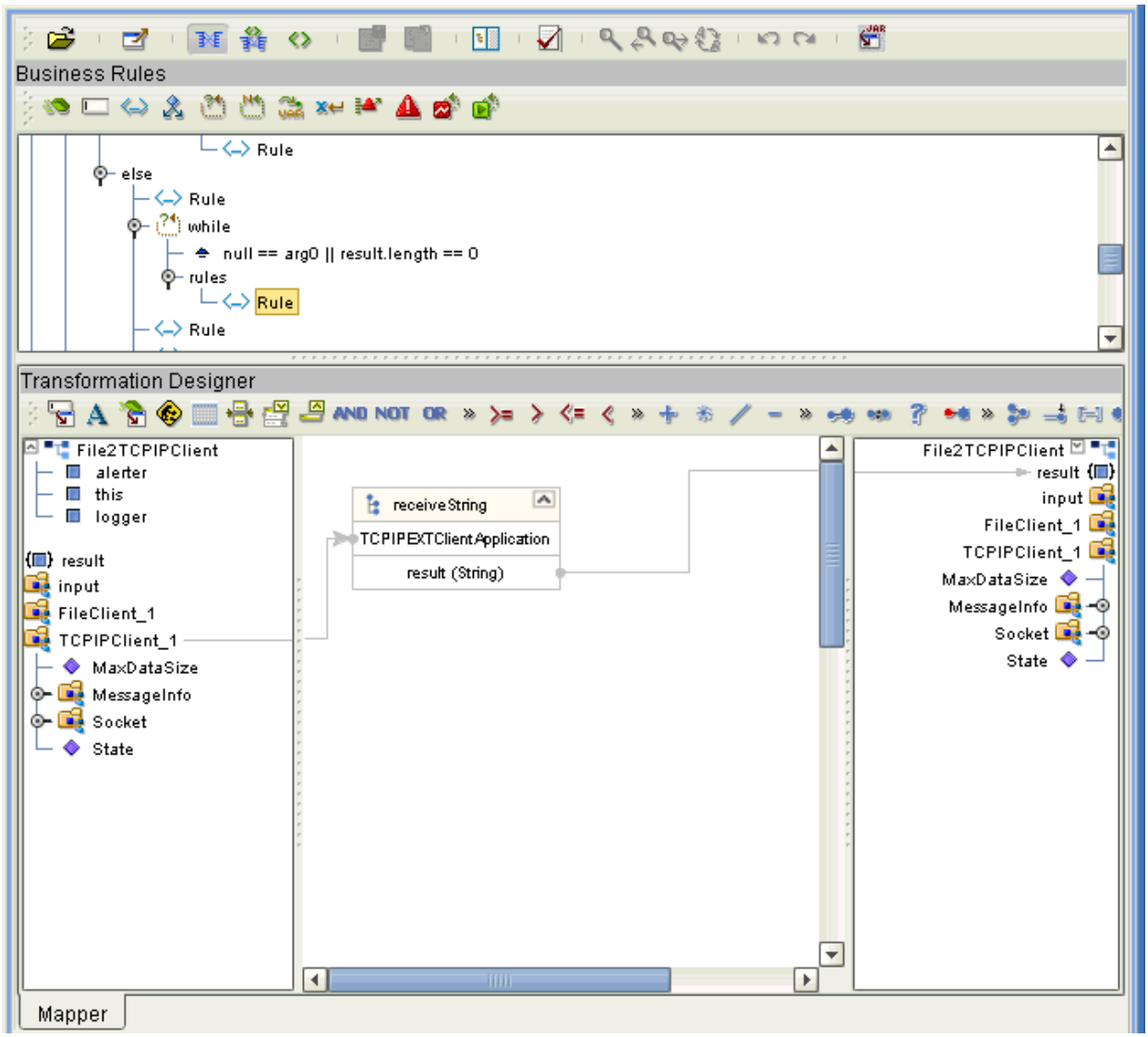


57 Click **Save** to save your changes.

58 Under the **rules** for **while**, create a method, **receiveString**, from the **TCPIPClient_1** node in the left pane.

59 Drag the method's **result String** to the **result** node in the right pane. See Figure 25.

Figure 25 File2TCPIPClient Collaboration Definition: Rule Under while rules

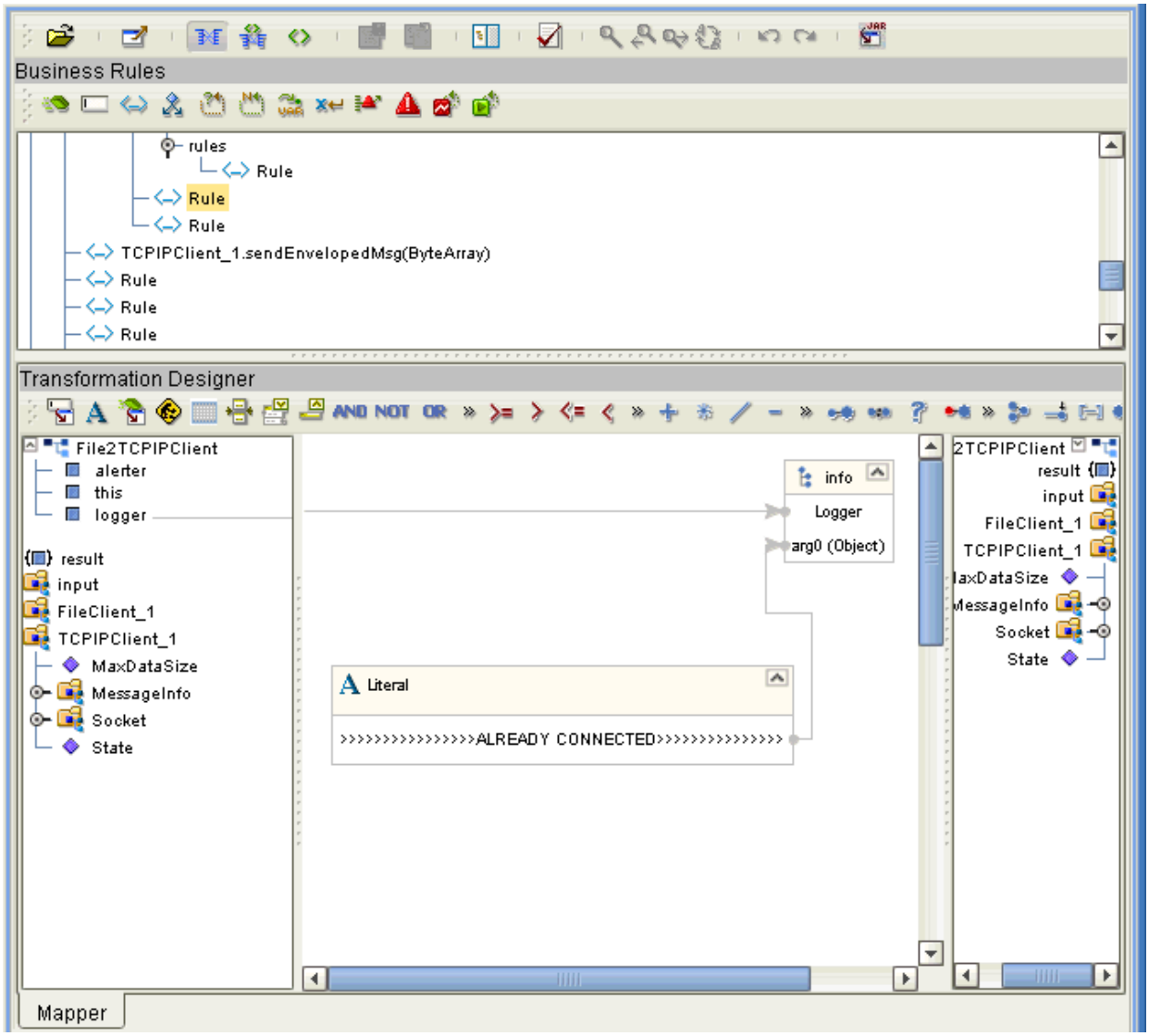


60 There are two more rules under **else/while**. For the first of these rules, start from the **logger** node under **File2TCPIPClient** in the left pane and create the method **info**.

61 Create a string **Literal** with the string **ALREADY CONNECTED** including angle brackets, as shown in Figure 26.

62 Drag the string (**ALREADY CONNECTED**) to **arg0 (Object)** under the **info** method. See Figure 26.

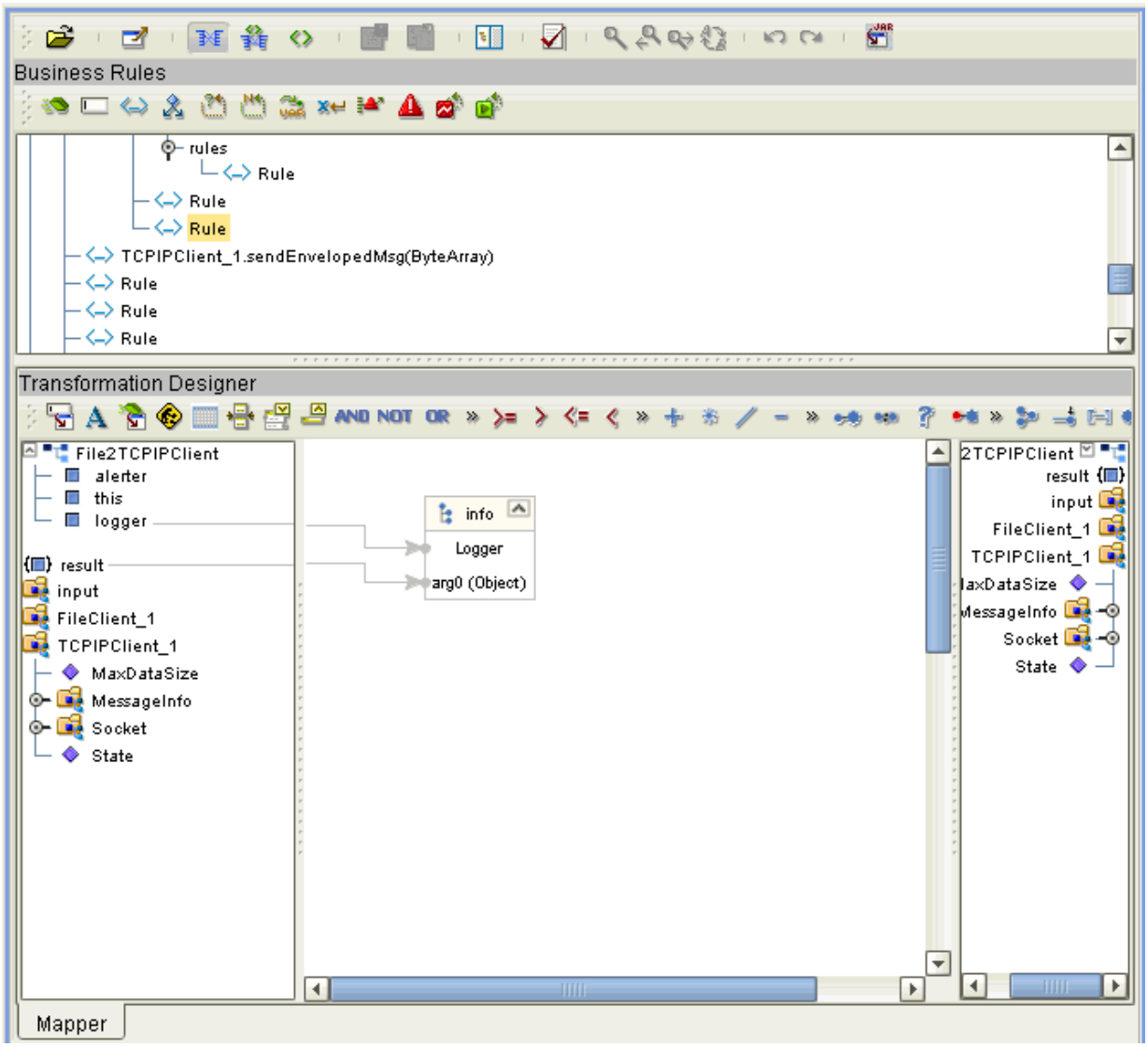
Figure 26 File2TCPIPClient Collaboration Definition: First rule Under else/while



63 From the **logger** node under **File2TCPIPClient**, create the method **info**.

64 Drag the **result** node in the left pane to **arg0 (Object)** in the **info Method** box. See Figure 27.

Figure 27 File2TCPIPClient Collaboration Definition: Second rule Under else/while



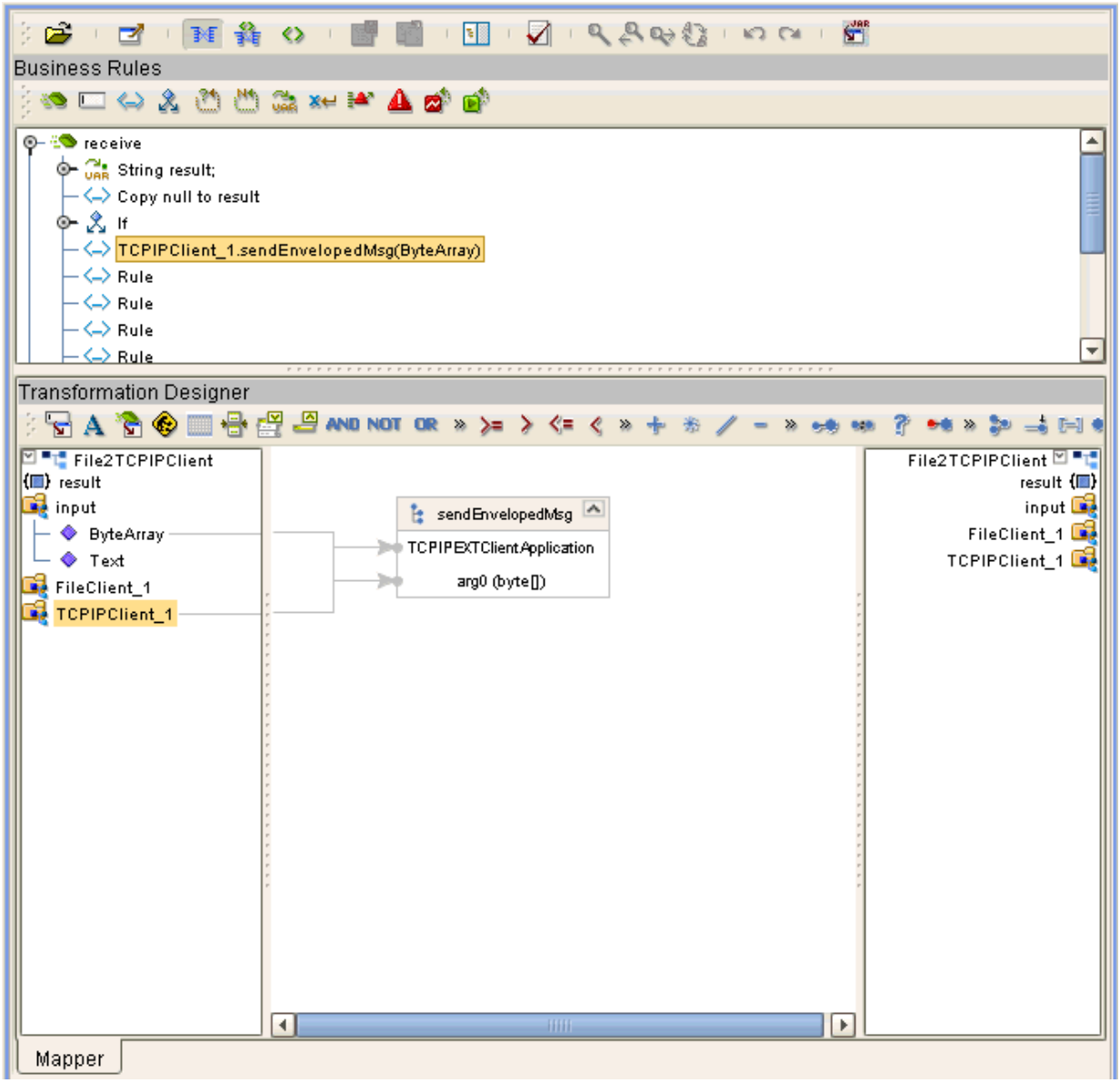
65 Click **Save** to save your changes.

66 You have finished creating the Collaboration Definition's primary **If rule**. You can close it up, if you wish.

67 For the Collaboration Definition's fourth Business Rule, right-click on **TCPIPClient_1** in the Transformation Designer's left pane to select and create the method **sendEnvelopedMessage**.

68 Drag the **ByteArray** node under **input** in the left pane to **arg0 (byte[])** under the **sendEnvelopedMessage** method. See Figure 28.

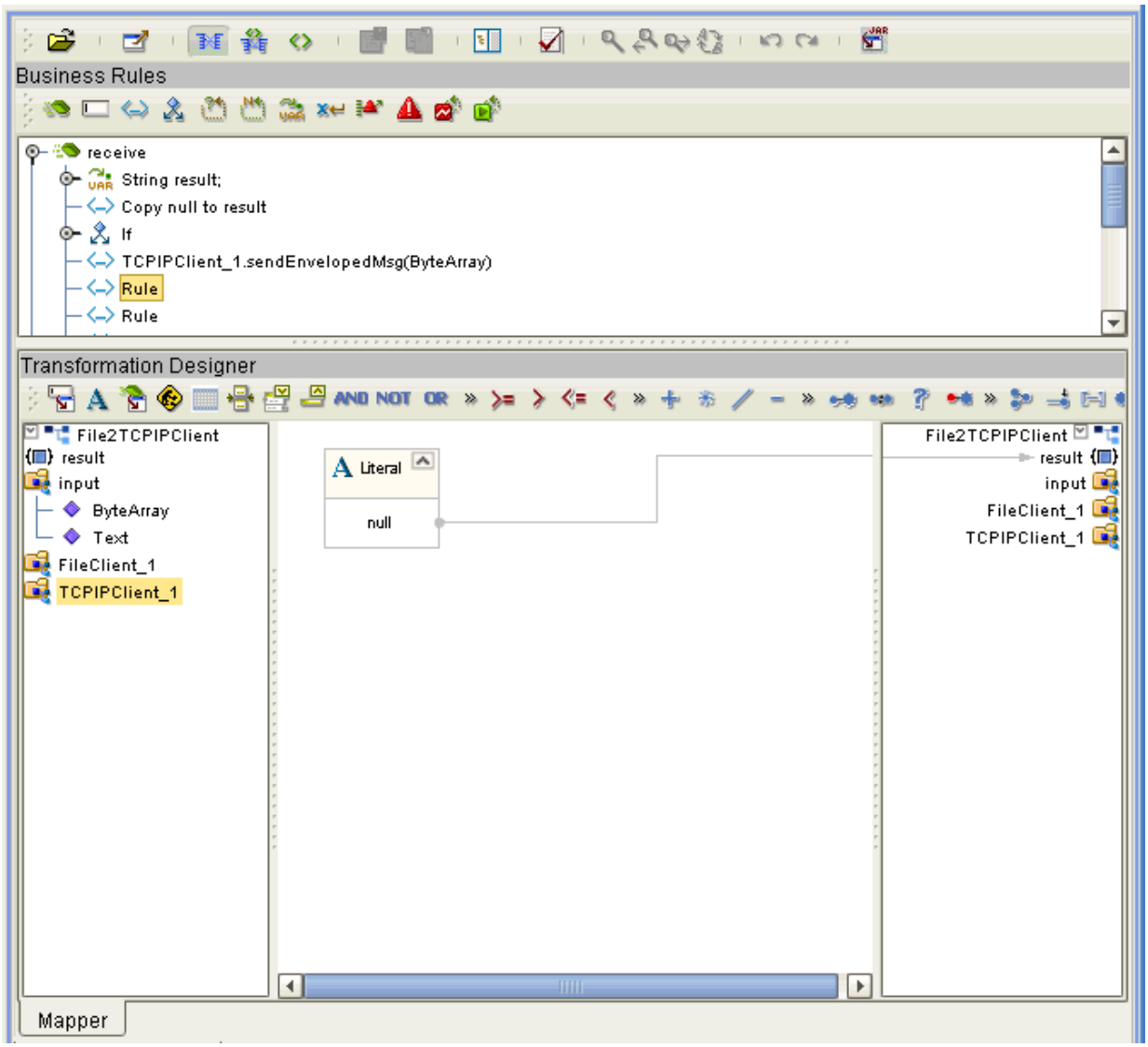
Figure 28 File2TCPIPClient Collaboration Definition: Fourth Business Rule



69 For the fifth Business Rule, create a null **Literal**.

70 Drag the **null** to the **result** node under **File2TCPIPClient**, in the right pane. See Figure 29.

Figure 29 File2TCPIPClient Collaboration Definition: Fifth Business Rule



71 Click **Save** to save your changes.

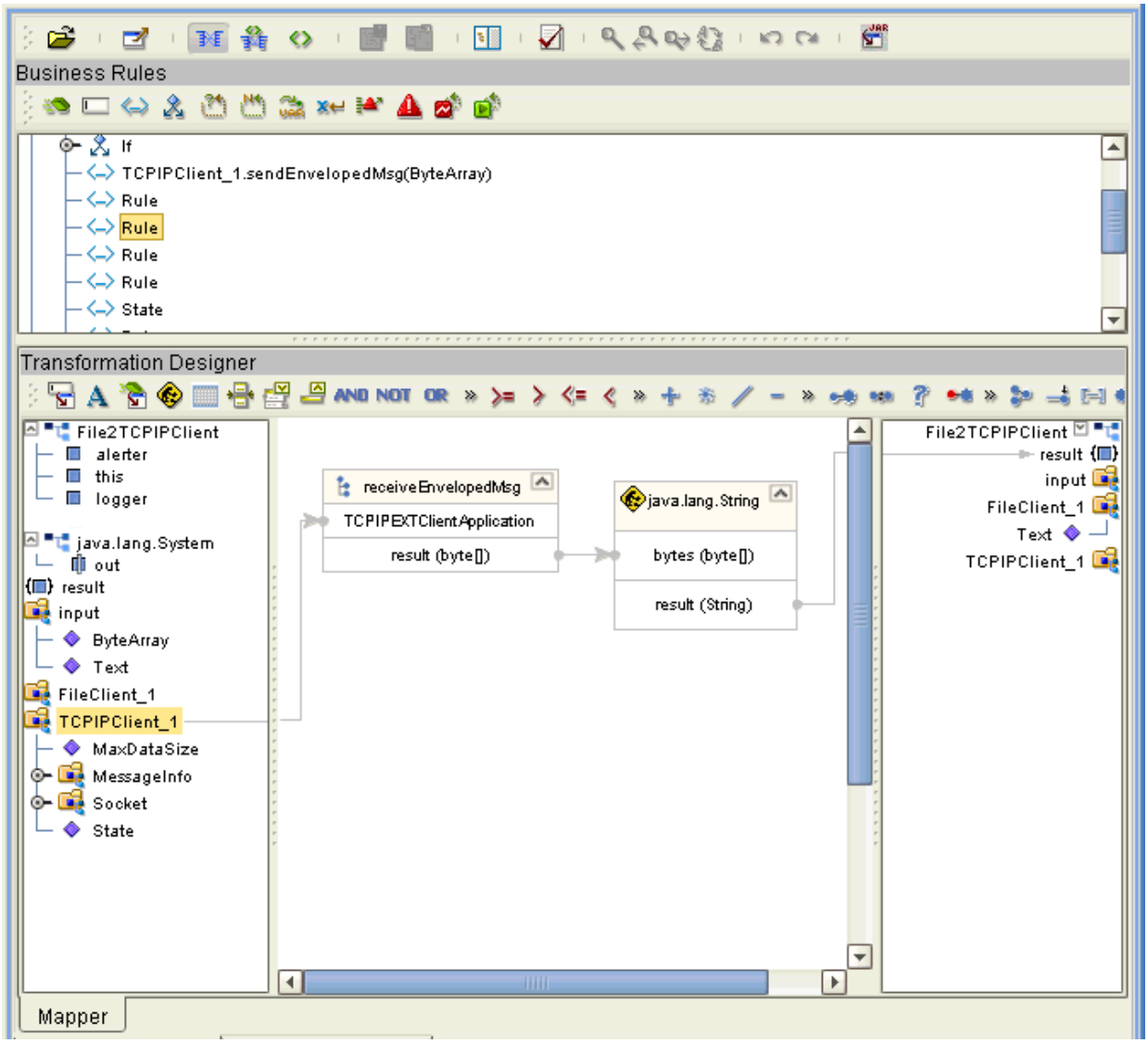
72 For the sixth Business Rule, right-click on **TCPIPClient_1** to create a **receiveEnvelopeMsg** method.

73 Call a **New Constructor** under the class **java.lang.String**.

74 Drag the **result (byte[])** to **bytes (byte[])** under **java.lang.String**.

75 Drag the **result (String)** under the **Constructor** to the **result** node in the right pane. See Figure 30.

Figure 30 File2TCPIPClient Collaboration Definition: Sixth Business Rule



76 For the seventh Business Rule, create a string **Literal** with the string **The value of receive String is**.

77 Create an **addition** operator and drag the string (**The value of receive String is**) to **number1 (num)** in the **addition** box.

78 From **File2TCPIPClient** in the left pane, right-click and select **Create New Variable of This Type** from the pop-up menu.

The **Create a Variable** dialog box appears.

79 Click the ellipsis (**...**) to display a choice list. Select **System** from the list in the dialog box.

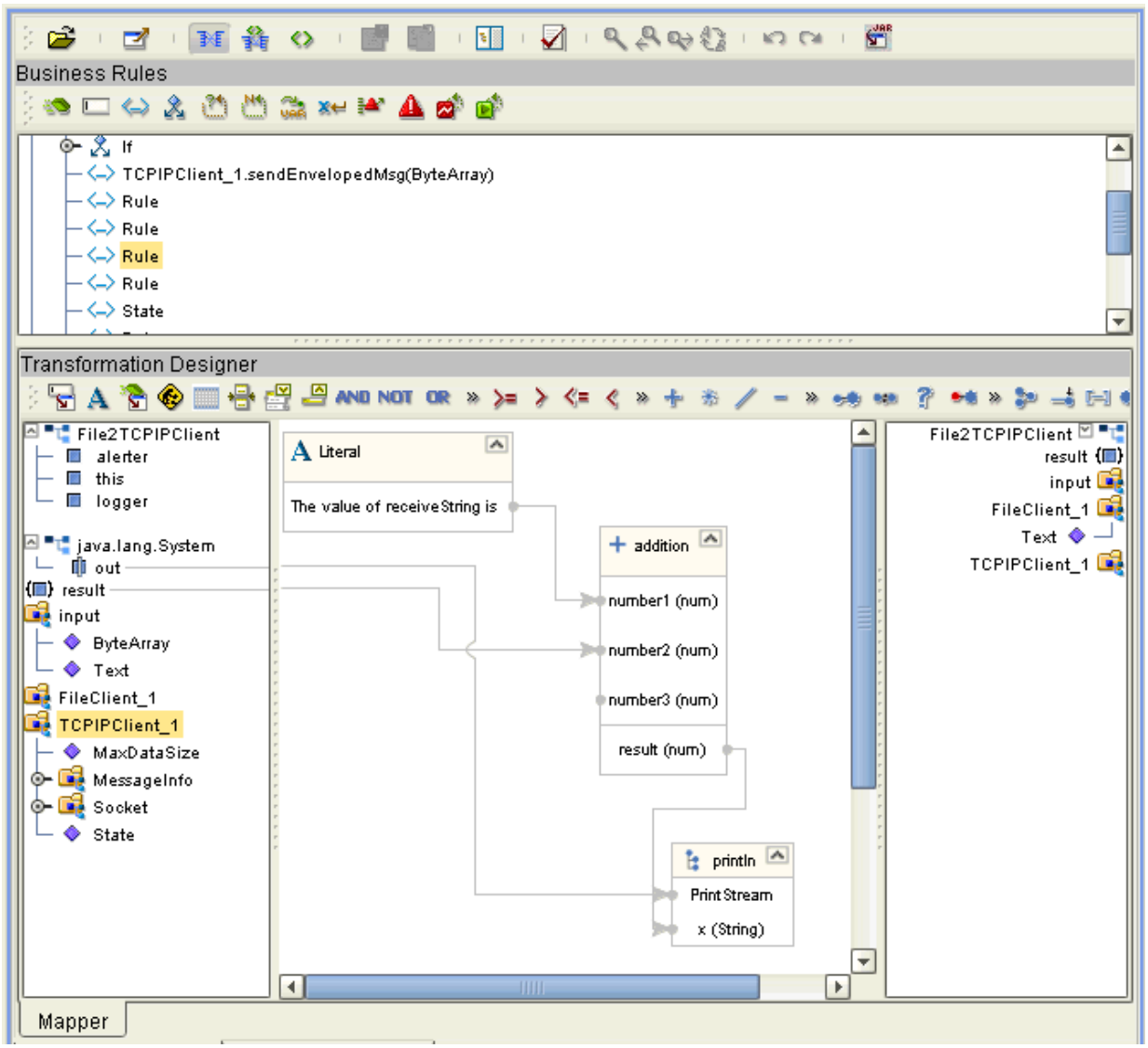
The node **java.lang.System** appears in the left pane. Expand this node to display the **Out** subnode.

80 From the **Out** subnode, create a **printIn** method.

81 Drag the **result** node in the left pane to **number2 (num)** in the **addition** box.

82 Drag the **result (num)** in the **addition** box to **x (String)** under the **printIn** method. See **Figure 32 on page 70**.

Figure 31 File2TCPIPClient Collaboration Definition: Seventh Business Rule



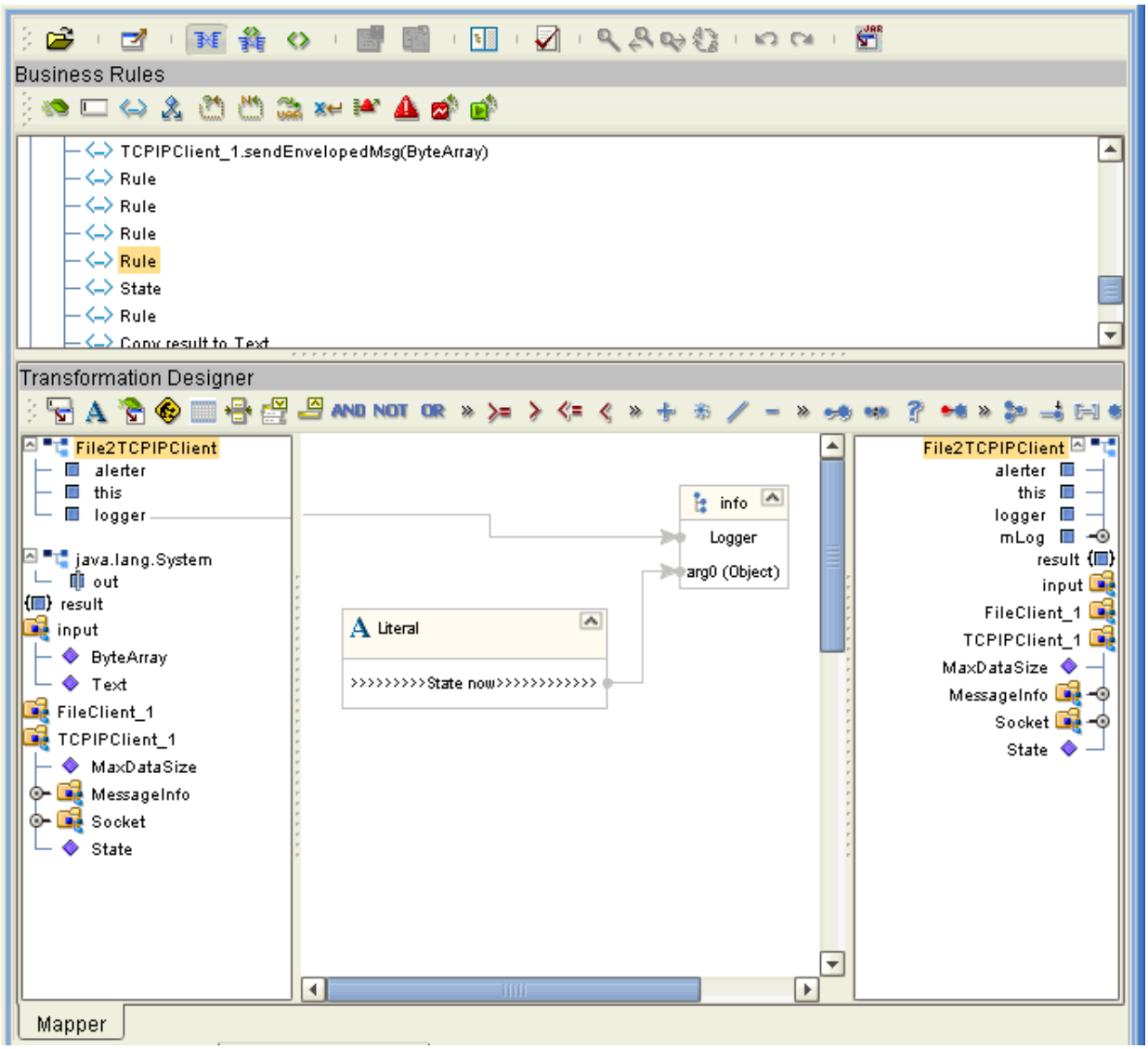
83 Click **Save** to save your changes.

84 For the eighth Business Rule, start from the **logger** node under **File2TCPIPClient** in the left pane and create the method **info**.

85 Create a string **Literal** with the string **State now** including angle brackets, as shown in Figure 32.

86 Drag the string (**State now**) to **arg0 (Object)** under the **info** method. See Figure 32.

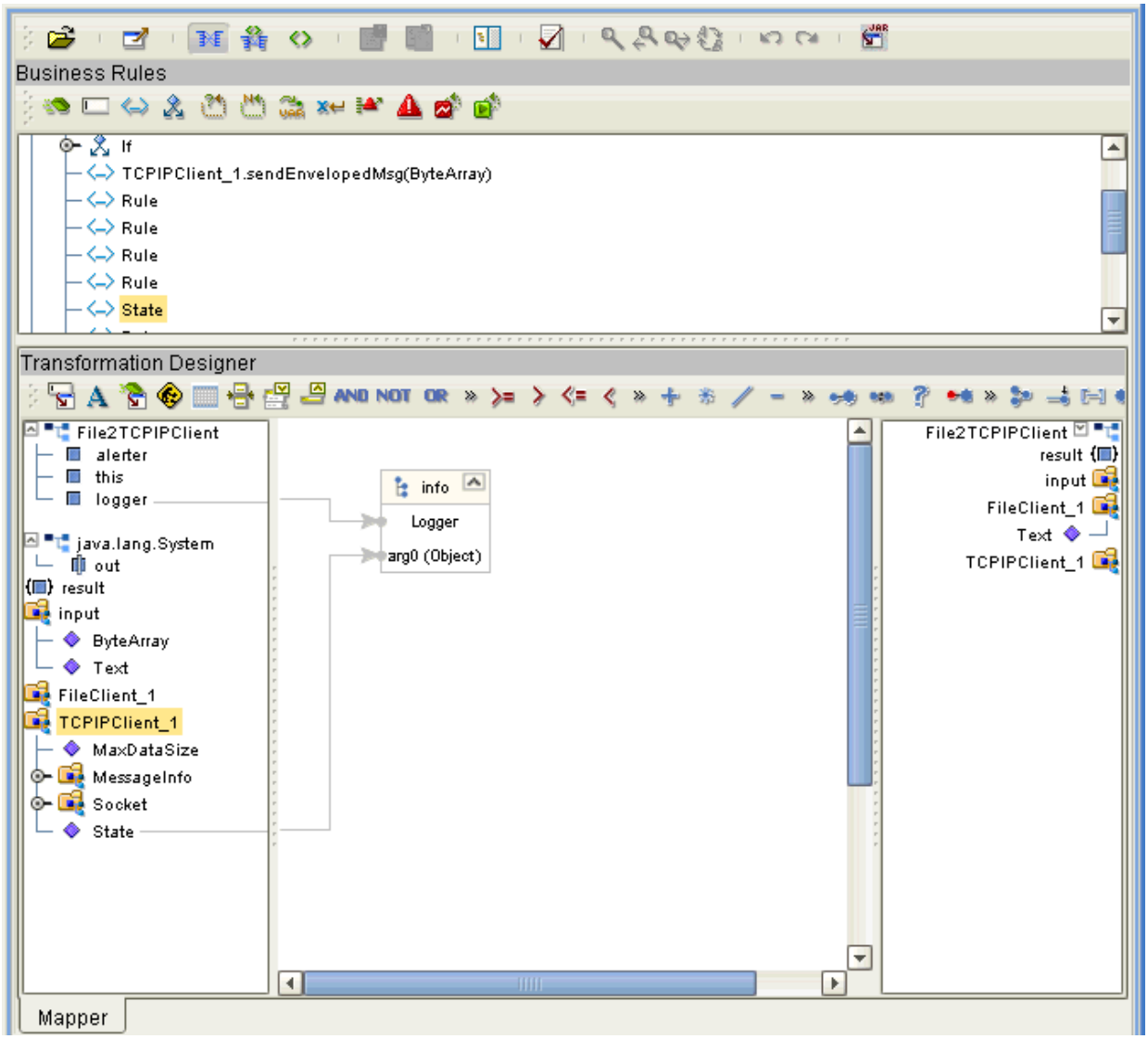
Figure 32 File2TCPIPClient Collaboration Definition: Eighth Business Rule



87 For the ninth Business Rule, start from the **logger** node under **File2TCPIPClient**, and create the method **info**.

88 Drag the **State** node under **Socket** in the left pane to **arg0 (Object)** in the **info Method** box. See Figure 33.

Figure 33 File2TCPIPClient Collaboration Definition: Ninth Business Rule

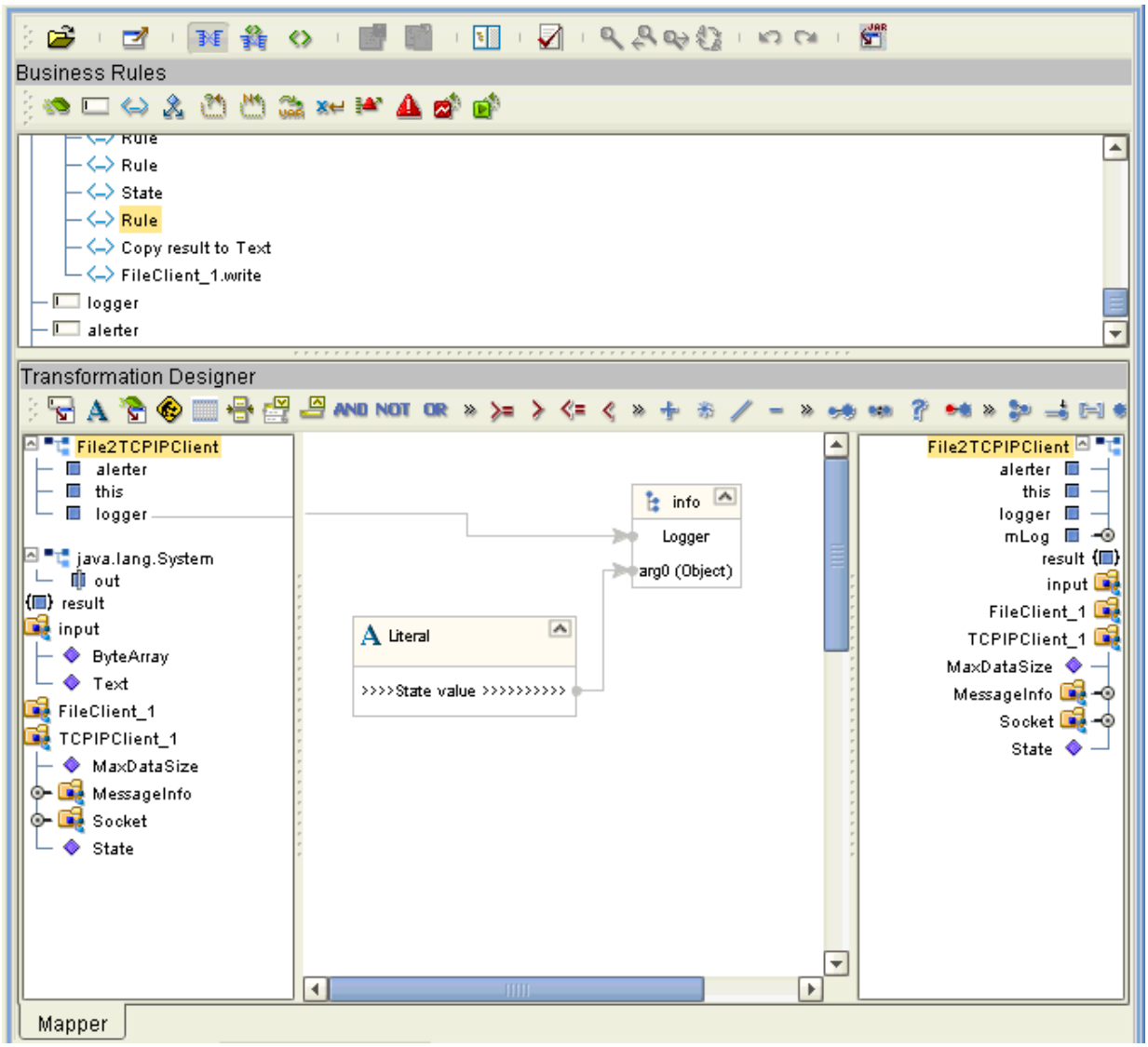


89 For the tenth Business Rule, start from the **logger** node under **File2TCPIPClient** in the left pane and create the method **info**.

90 Create a string **Literal** with the string **State value** including angle brackets, as shown in Figure 34.

91 Drag the string (**State value**) to **arg0 (Object)** under the **info** method.See Figure 34.

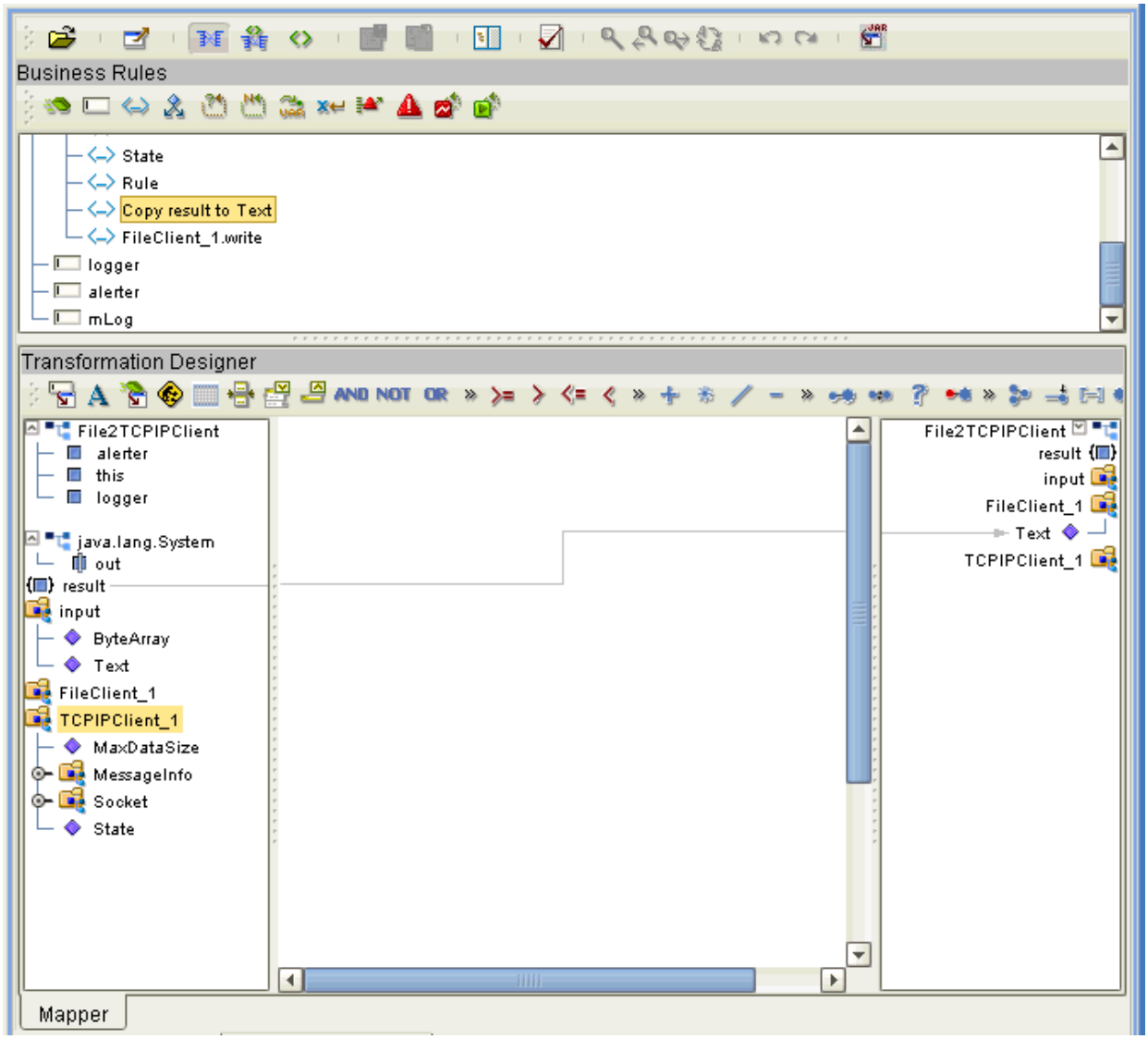
Figure 34 File2TCPIPClient Collaboration Definition: Tenth Business Rule



92 Click **Save** to save your changes.

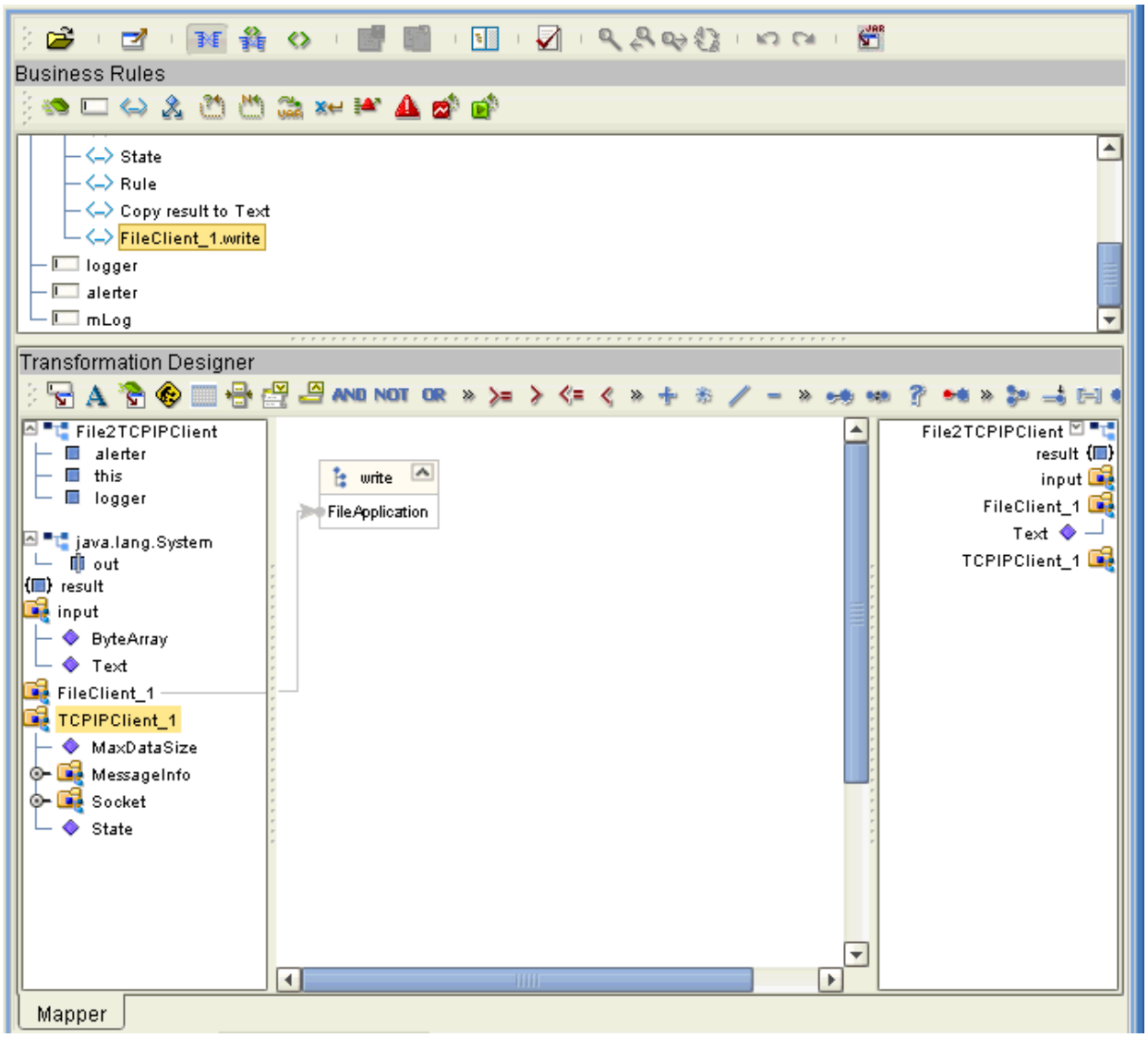
93 For the eleventh Business Rule, drag the **result** node in the left pane to the **Text** node under **FileClient_1** in the right pane. See Figure 35.

Figure 35 File2TCPIPClient Collaboration Definition: Eleventh Business Rule



94 For the twelfth Business Rule, create a **write** method from the **FileClient_1** node. See Figure 36.

Figure 36 File2TCPIPClient Collaboration Definition: Twelfth Business Rule



95 Click **Save** to save your new Java Collaboration Definition. If the validation shows any errors, use the exception information in the Validation pane to troubleshoot the errors.

To create Business Rules within TCPIPServer2File

1 From the **Project Explorer** tree, double-click the **TCPIPServer2File** Collaboration Definition.

The Java Collaboration Editor displays **TCPIPServer2File**.

2 Expand all the nodes in the left pane of the Transformation Designer. There are 10 Business Rules under the **receive()** method in the **Business Rules** pane.

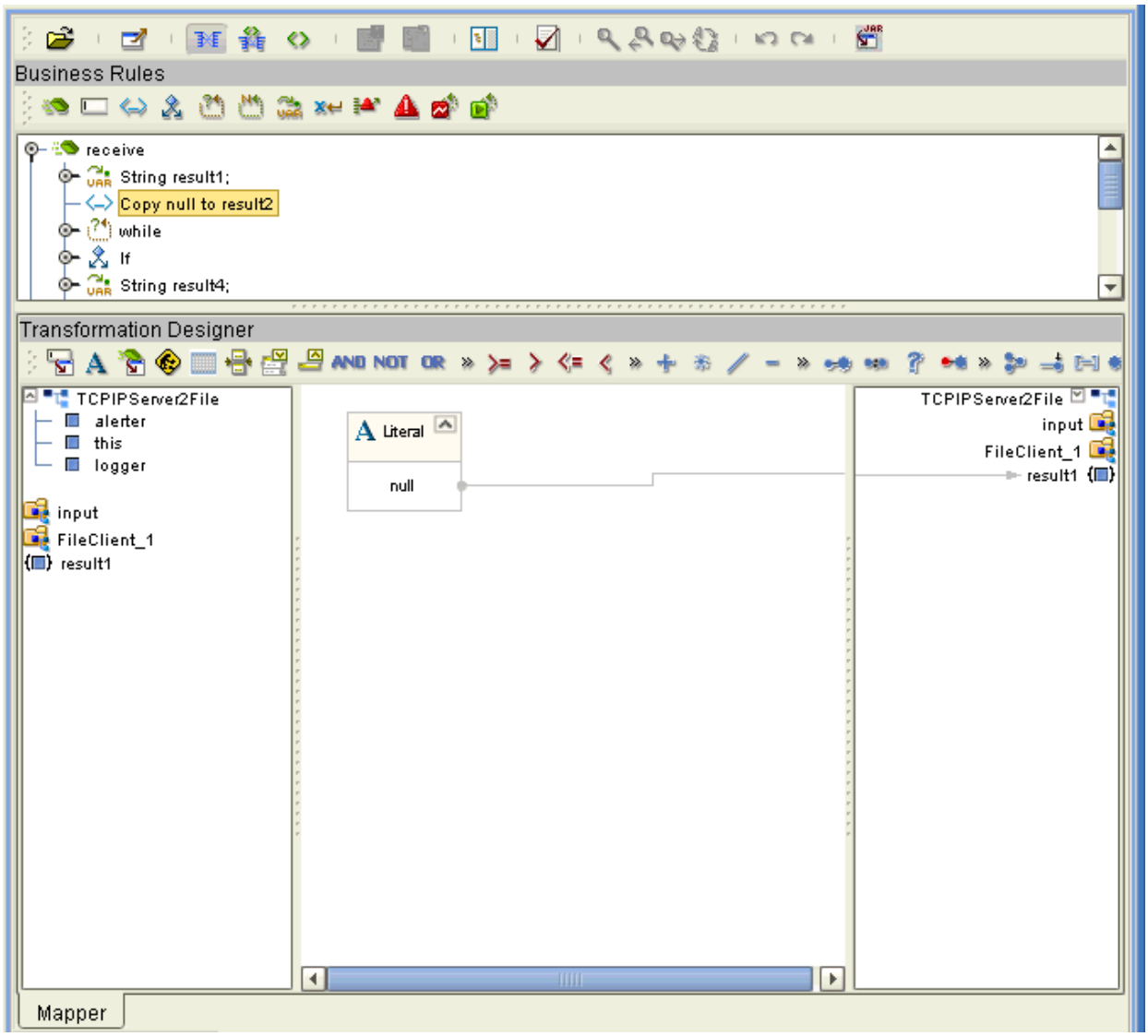
3 For the first Business Rule, create a **local variable** named **String result2**.

4 Enter **initializer** in the **Local Variable** dialog box.

5 For the second Business Rule, create a null **Literal** as a **rule**, and drag the **null** to the **result1** node in the right pane.

6 This action creates the **Copy null to result2** Business Rule. See Figure 37.

Figure 37 TCPIPServer2File Collaboration Definition: Second Business Rule



7 Click **Save** on the Enterprise Designer toolbar to save your changes.

8 For the third Business Rule, create a **while** rule.

9 For the **condition** under the **while** rule, create the following boxes in the Transformation Designer:

- **Literal:** null type
- **Literal: int** type, integer: **0**
- **equal** (comparison operator)
- Another **equal** (comparison operator)
- **OR** (Boolean operator)

10 In addition, from the **result** node under **File2TCPIPClient**, create the method **length**.

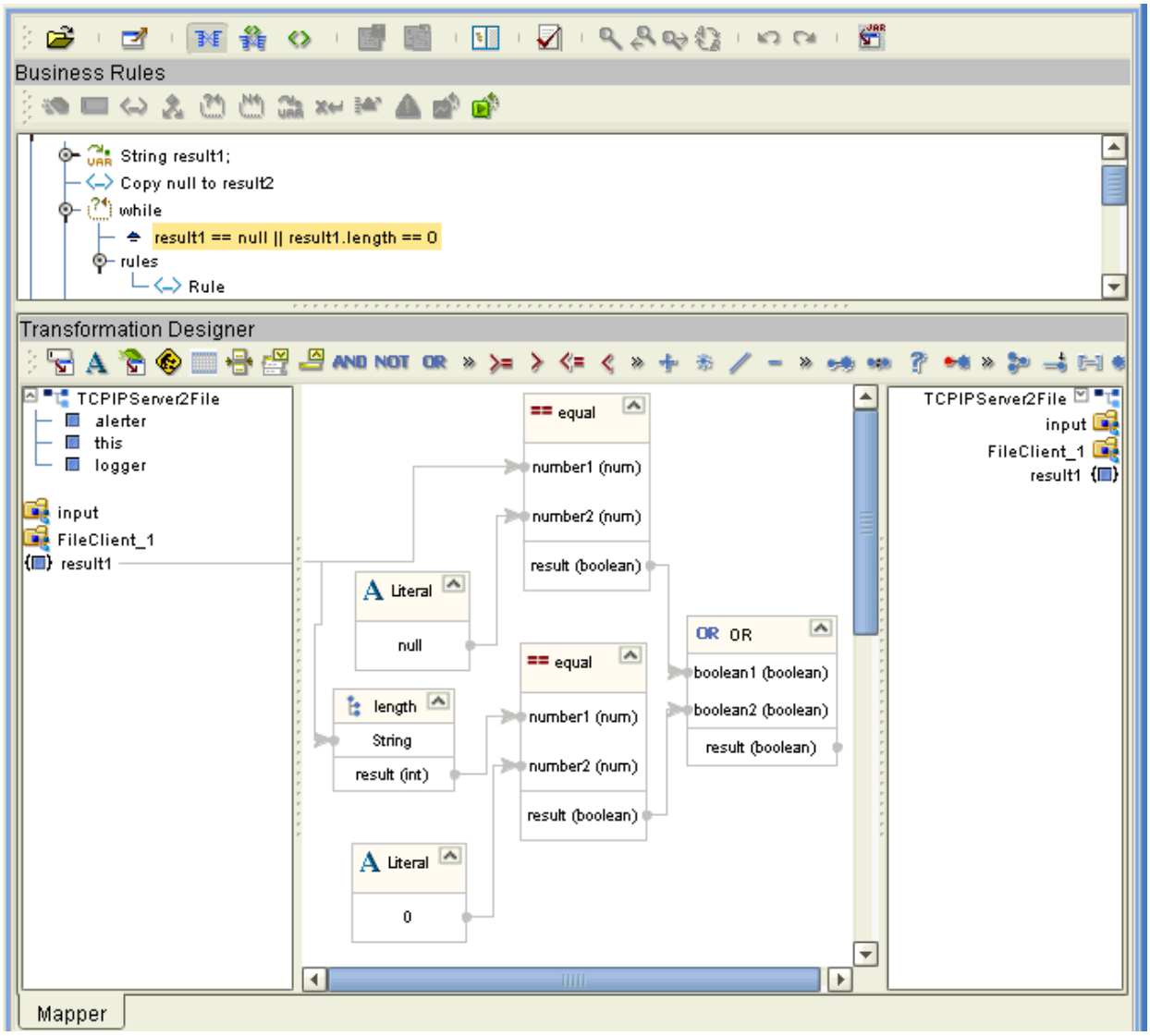
11 Position the boxes in the Transformation Designer as shown in **Figure 38 on page 77**.

12 Create the following connections between the boxes (shown in the previous list), including the **Method** box, in the Transformation Designer:

- Drag the **result** node under **File2TCPIPClient** to **number1 (num)** under **equal** (the upper box).
- Drag **null** under its **Literal** to **number2 (num)** under **equal** (the upper box).
- Drag **0** under its **Literal** to **number2 (num)** under **equal** (the lower box).
- Drag **result(int)** under the **length** method to **number1 (num)** under **equal** (the lower box).
- Drag **result (boolean)** under **equal** (the upper box) to **boolean1 (boolean)** under **OR**.
- Drag **result (boolean)** under **equal** (the lower box) to **boolean2 (boolean)** under **OR**.

Make sure the resulting window appears as shown in **Figure 38 on page 77**.

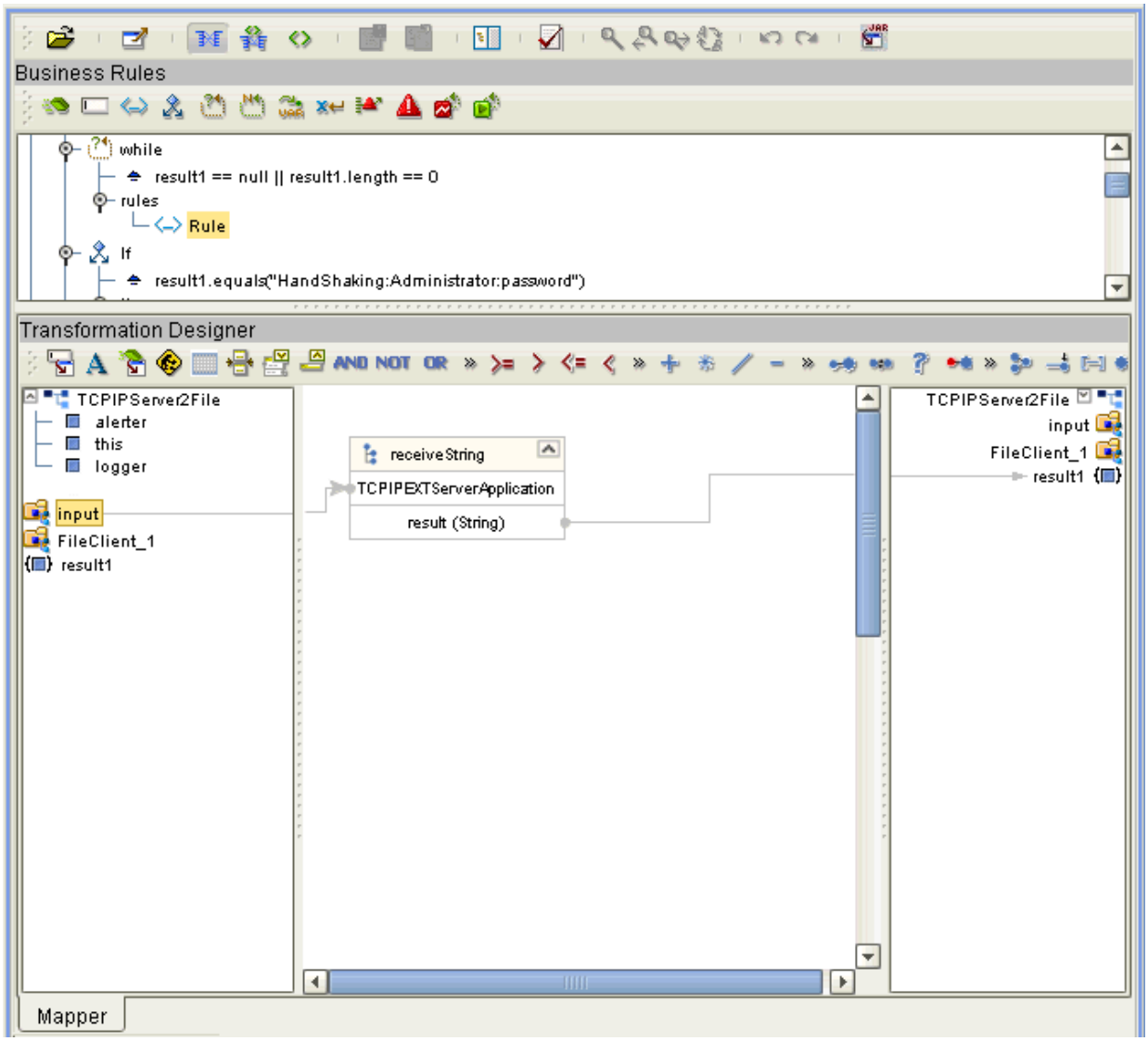
Figure 38 \TCPIPServer2File Collaboration Definition: while Rule Condition



13 To create a **rule** under the **while rule**, start from the **input** node in the left pane and create the method **receiveString**.

14 Drag the method's **result (String)** to the **result1** node in the right pane. See Figure 39.

Figure 39 TCPIPServer2File Collaboration Definition: While rule Sub-rule



15 Click **Save** to save your changes.

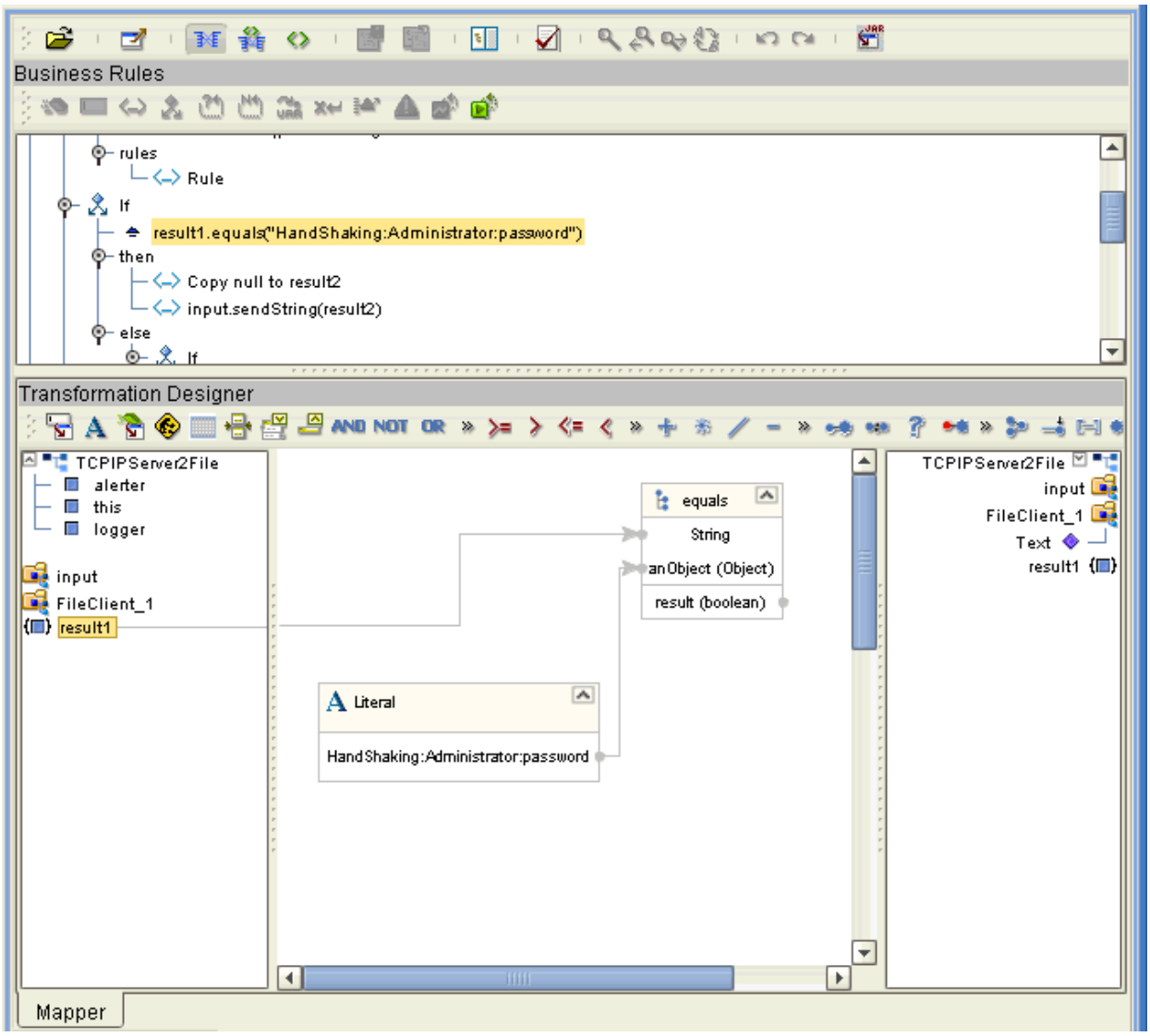
16 For the fourth Business Rule, create an **If rule**.

17 For the **condition** of the **If rule**, create the method **equals** from the **result1** node in the left pane.

18 Create a string **Literal** with the string **HandShaking.Administrator.password**.

19 From the **Literal** box, drag the string (**HandShaking.Administrator.password**) to **anObject (Object)** under the method **equals**. See Figure 40.

Figure 40 TCPIPServer2File Collaboration Definition: If Rule Condition

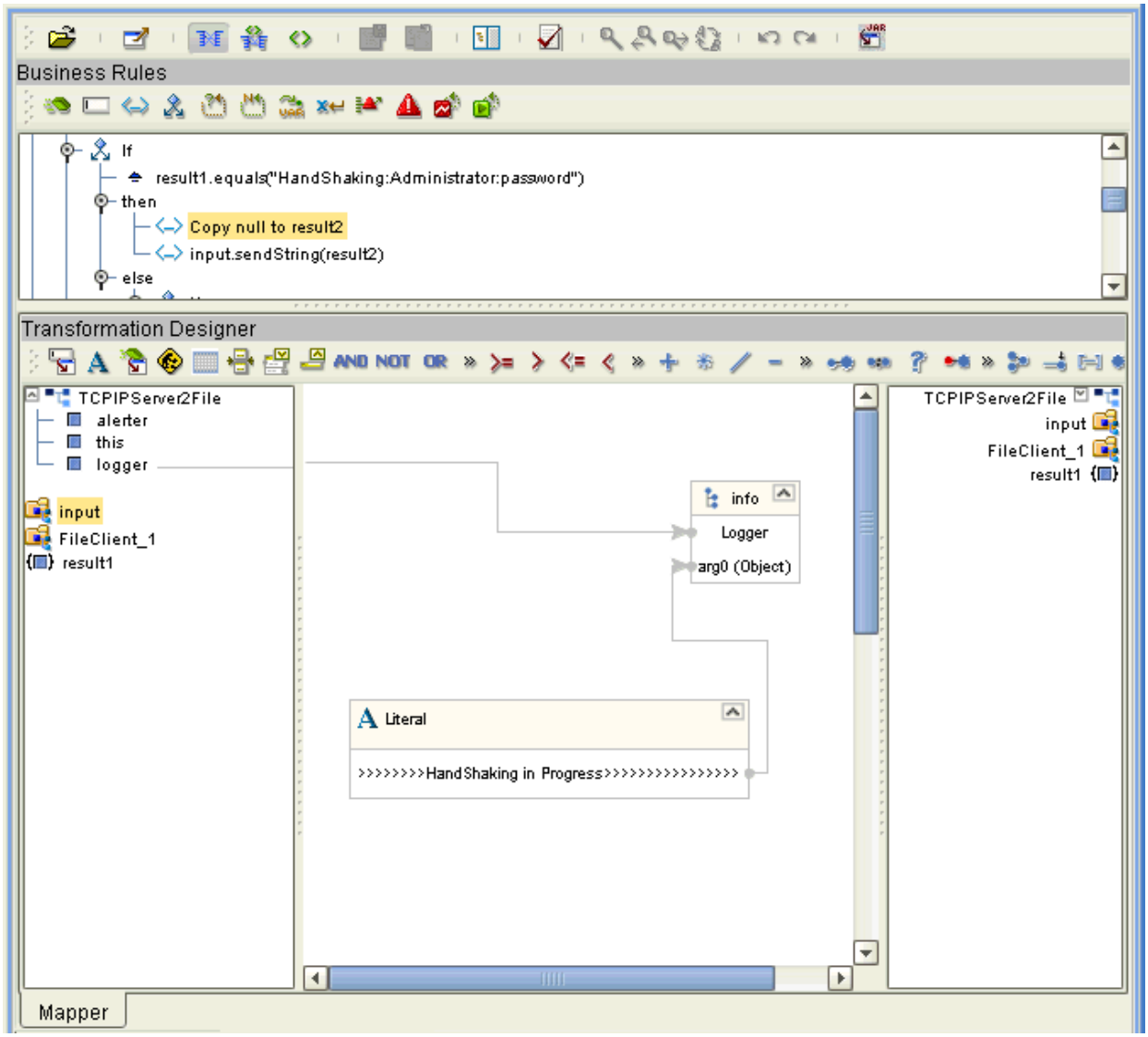


20 For **then** under the **If rule**, start from the **logger** node under **TCPIPServer2File** and create the method **info**.

21 Create a string **Literal** with the string **HandShaking in Progress,** including angle brackets, as shown in Figure 41.

22 From the Literal box in the Transformation Designer, drag the string (**HandShaking in Progress**) to **arg0 (Object)** under **info**. See Figure 41.

Figure 41 TCPIPServer2File Collaboration Definition: then Rule 1

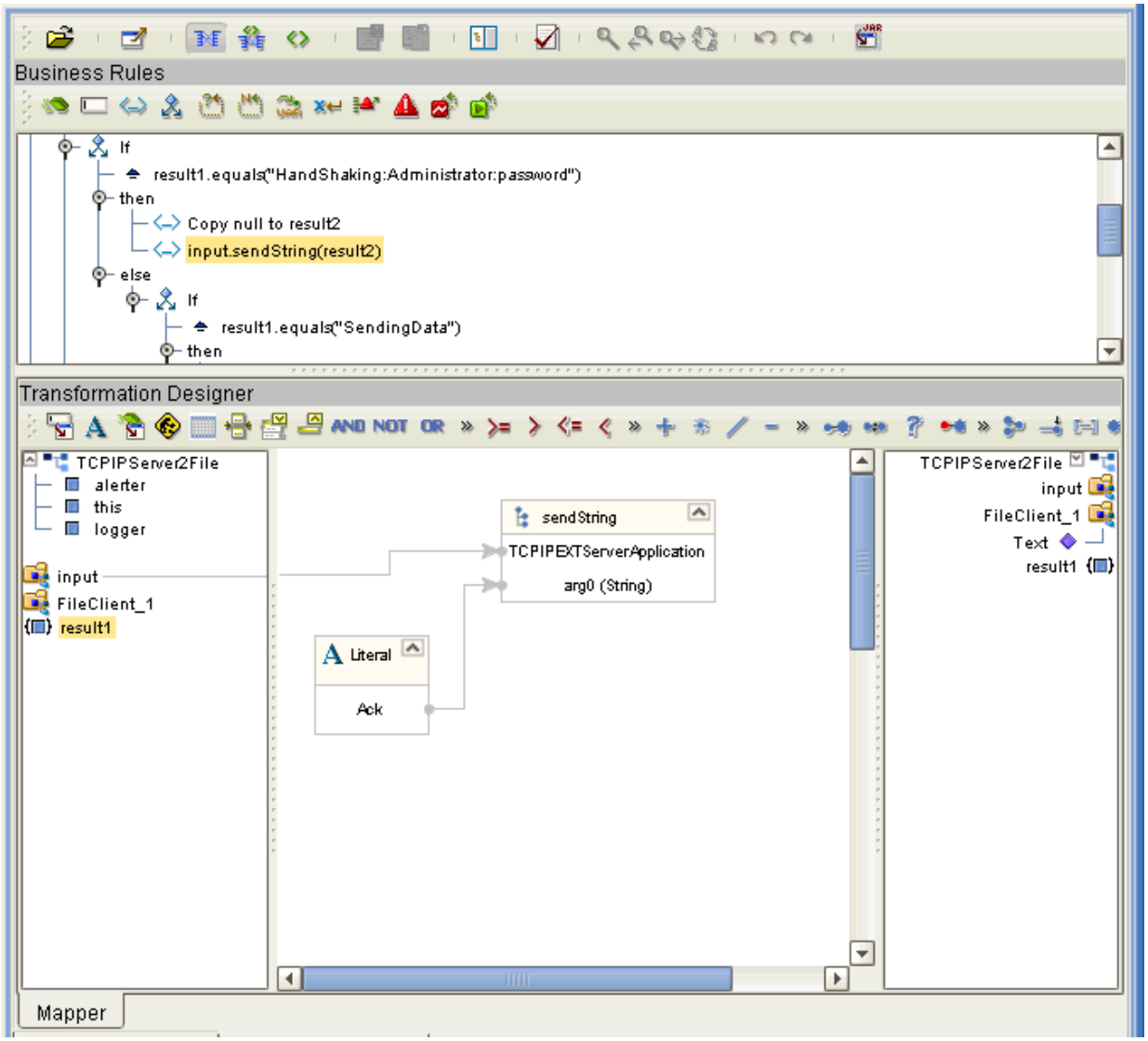


23 From the **input** node, create the method **sendString**.

24 Create a string **Literal** with the string **Ack**.

25 Drag the string (**Ack**) to **arg0 (String)** under the **sendString** method. See Figure 42.

Figure 42 TCPIPServer2File Collaboration Definition: then Rule 2



26 Click **Save** to save your changes.

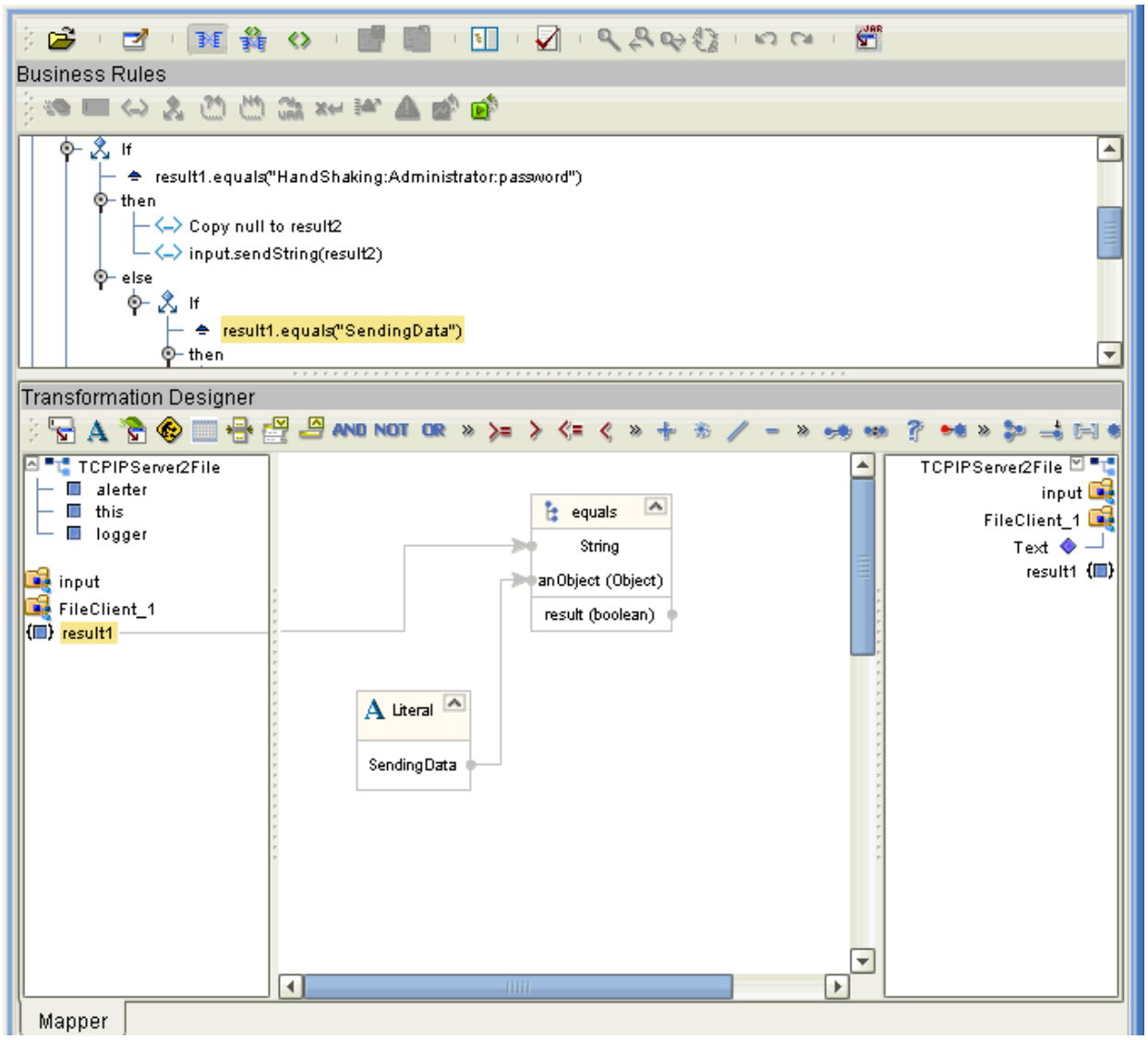
27 Under else (under the previous **If rule**) create a nested **If rule**.

28 For the **condition** of the **If rule**, create the method **equals** from the **result1** node in the left pane.

29 Create a string **Literal** with the string **SendingData**.

30 From the **Literal** box, drag the string (**SendingData**) to **anObject (Object)** under the method **equals**. See Figure 43.

Figure 43 TCPIPServer2File Collaboration Definition: Nested If rule Condition

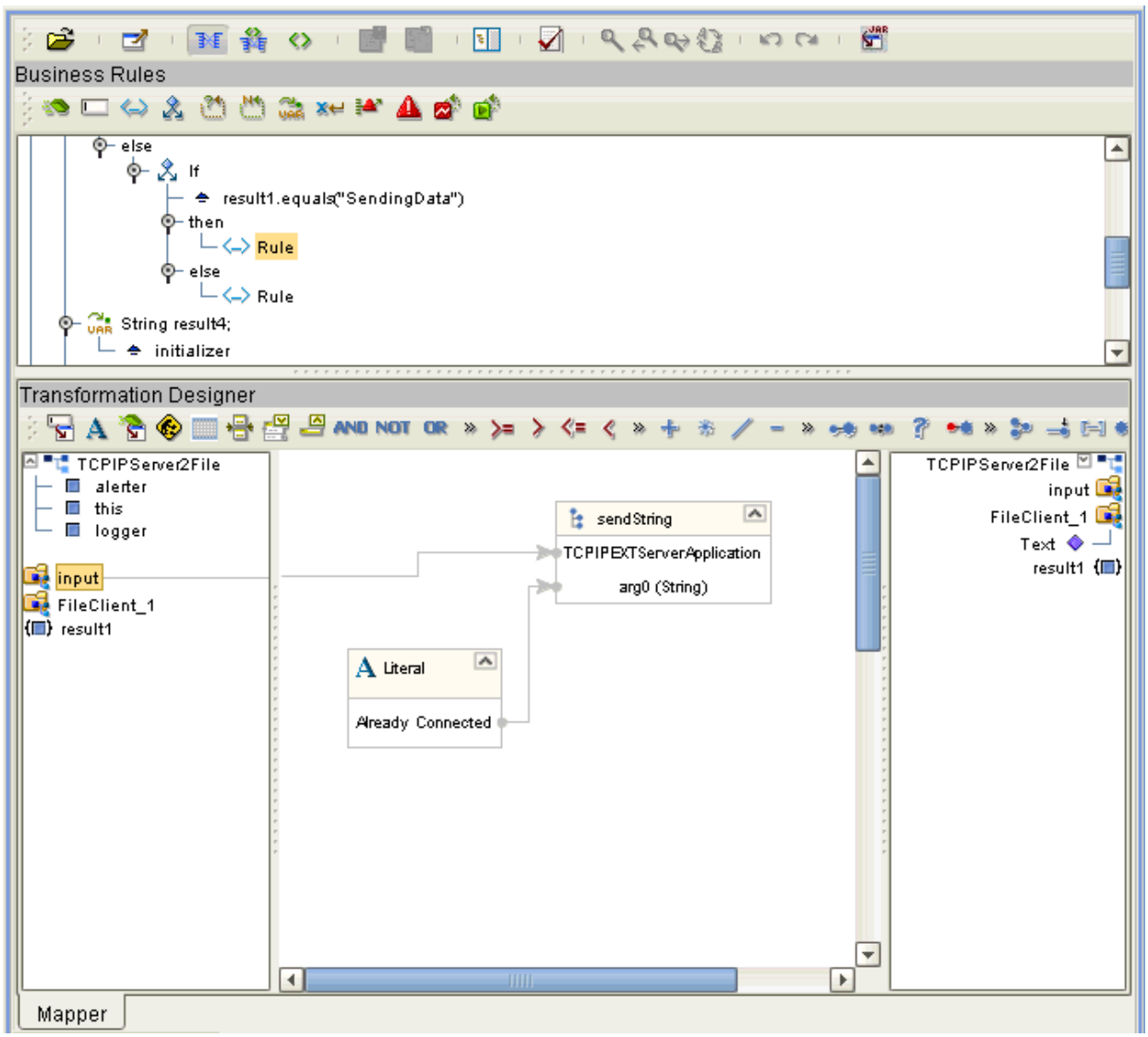


31 For **then** under this **If rule**, create the method **sendString** from the **input** node in the left pane.

32 Create a string **Literal** with the string **Already Connected**.

33 From the **Literal** box, drag the string (**Already Connected**) to **arg0 (String)** under the method **sendString**. See Figure 44.

Figure 44 TCPIPServer2File Collaboration Definition: Nested If rule then

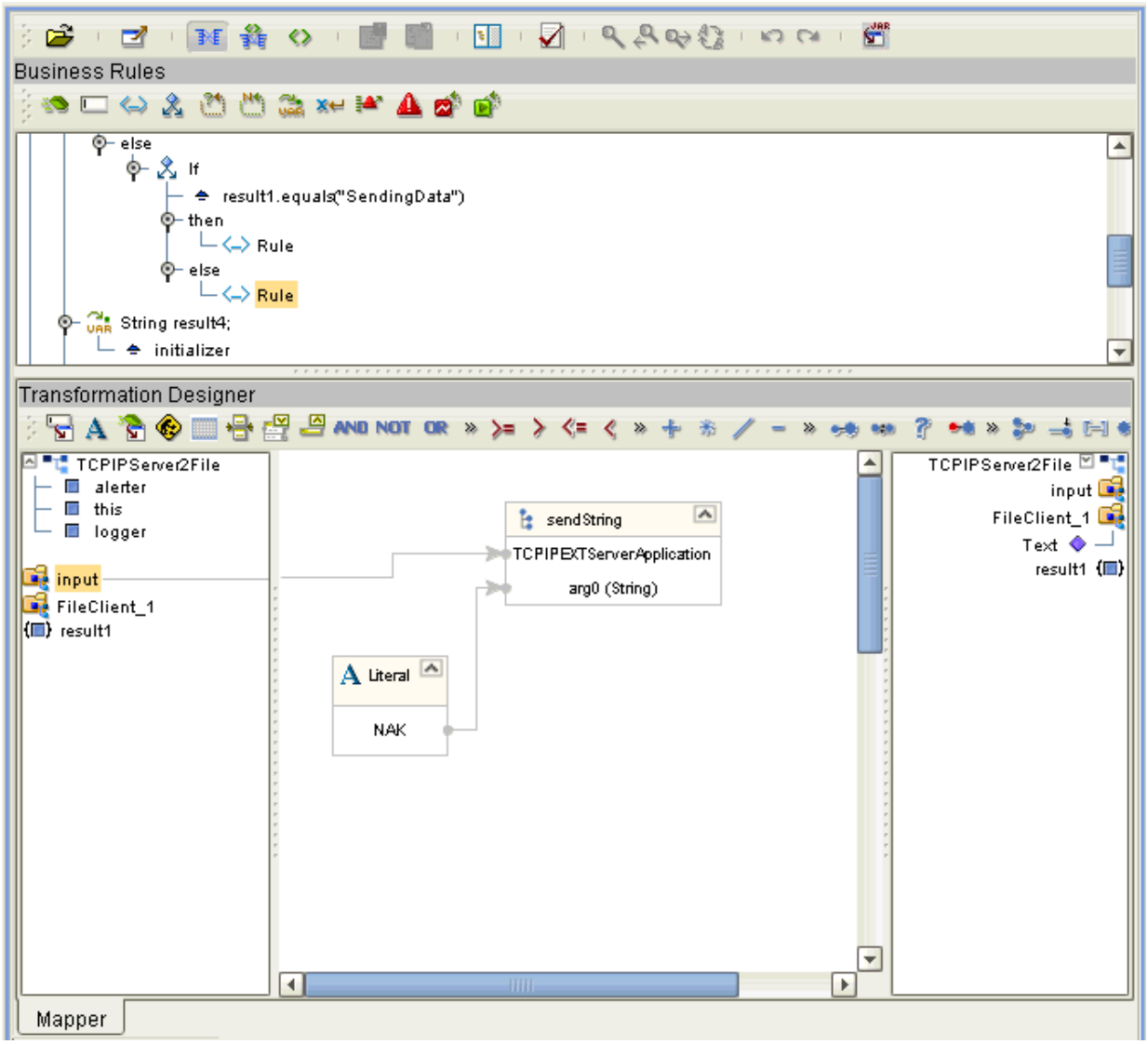


34 For **else** under this **If rule**, create the method **sendString** from the **input** node in the left pane.

35 Create a string **Literal** with the string **NAK**.

36 From the **Literal** box, drag the string (**NAK**) to **arg0 (String)** under the method **sendString**. See Figure 45.

Figure 45 TCPIPServer2File Collaboration Definition: Nested If rule else



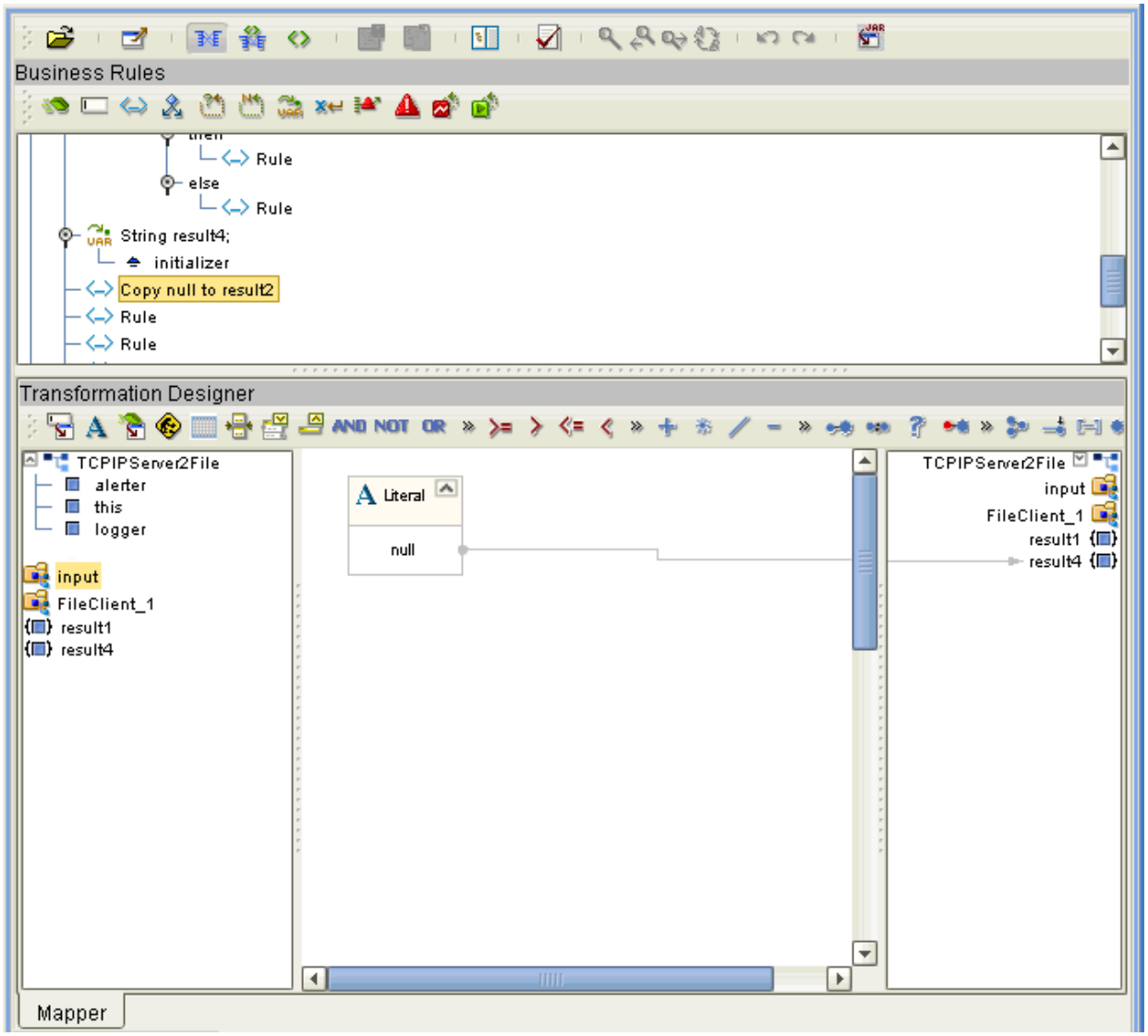
37 Click **Save** to save your changes.

38 For the fifth Business Rule, create a **local variable** named **String result4**.

39 Enter **initializer** in the **Local Variable** dialog box.

40 For the sixth Business Rule, create a null **Literal** and drag the **null** to the **result4** node in the right pane. See Figure 46.

Figure 46 TCPIPServer2File Collaboration Definition: Sixth Business Rule



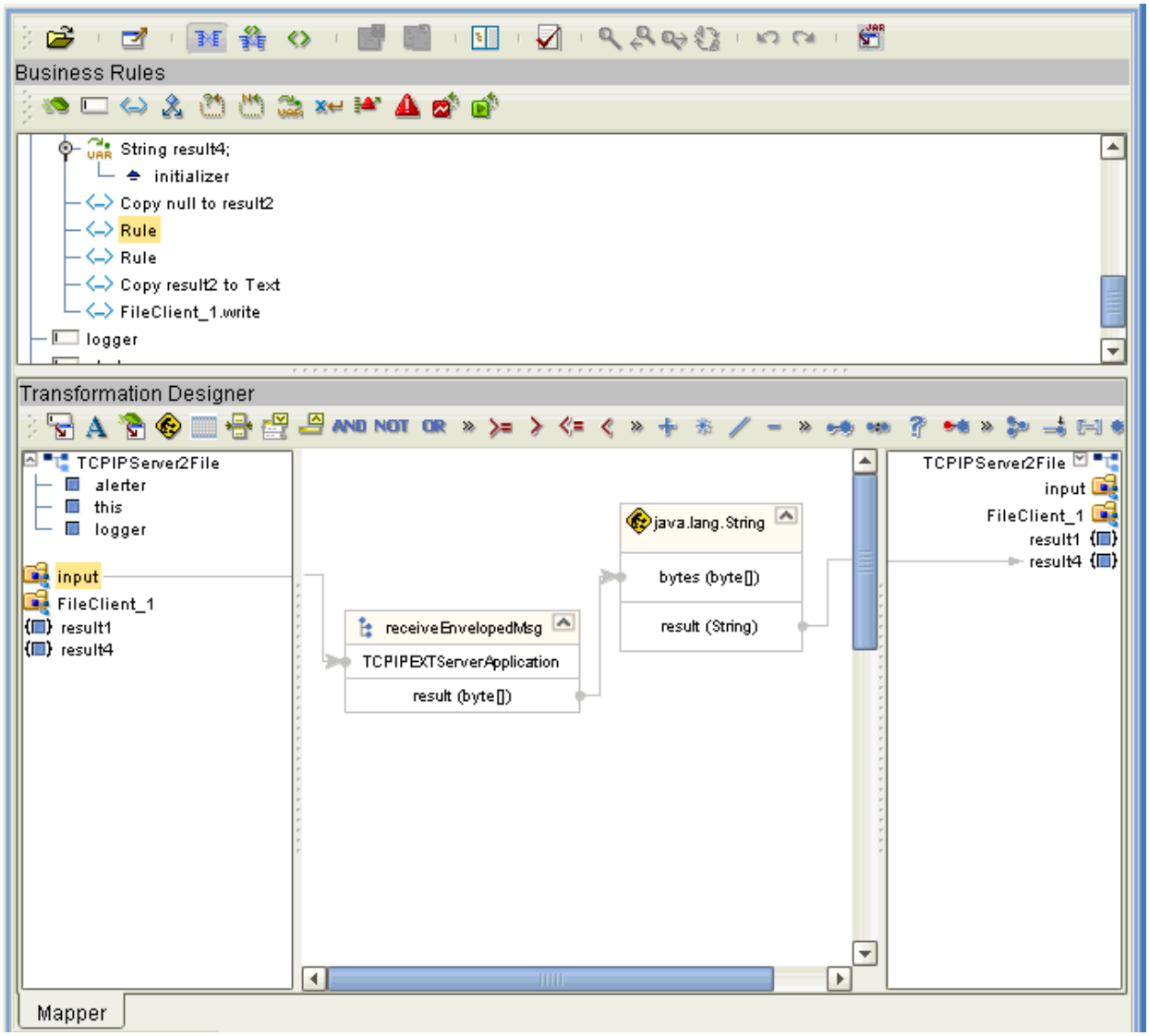
41 For the seventh Business Rule, right-click on the **input** node in the left pane to create a **receiveEnvelopeMsg** method.

42 Call a **New Constructor** under the class **java.lang.String**.

43 Drag the **result (byte[])** to **bytes (byte[])** under **java.lang.String**.

44 Drag the **result (String)** under the **Constructor** to the **result4** node in the right pane. See Figure 47.

Figure 47 TCPIPServer2File Collaboration Definition: Seventh Business Rule



45 For the eighth Business Rule, right-click on the **input** node in the left pane to create a **sendEnvelopeMsg** method.

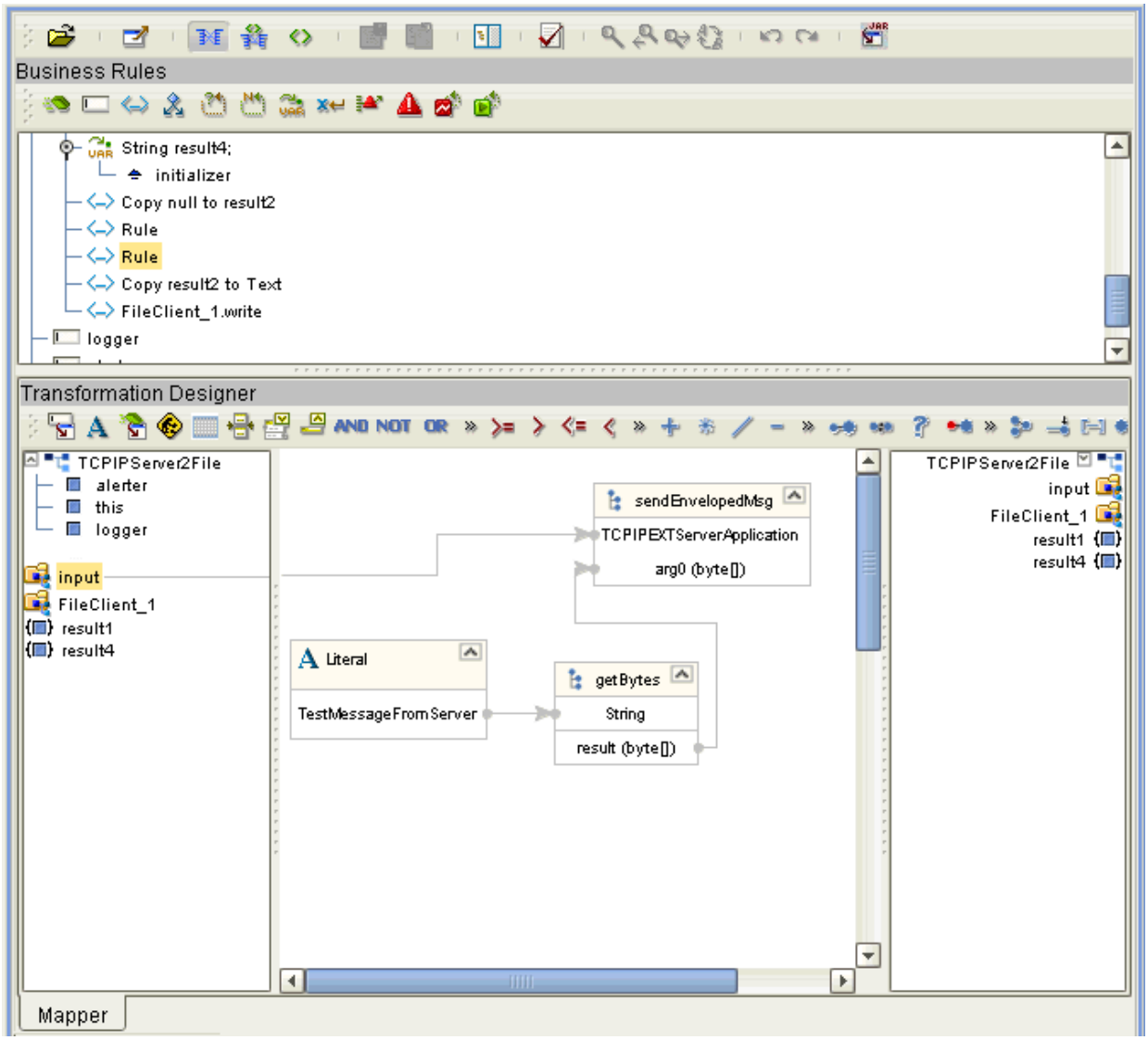
46 From the **input** node, create another method, **getBytes**, then delete its link to the node.

47 Create a string **Literal** with the string **TestMessageFromServer**.

48 Drag the string (**TestMessageFromServer**) to **String** under the **getBytes** method.

49 Drag the **result (byte[])** under the **getBytes** method to **arg0 (byte[])** under the **sendEnvelopeMsg** method. See Figure 48.

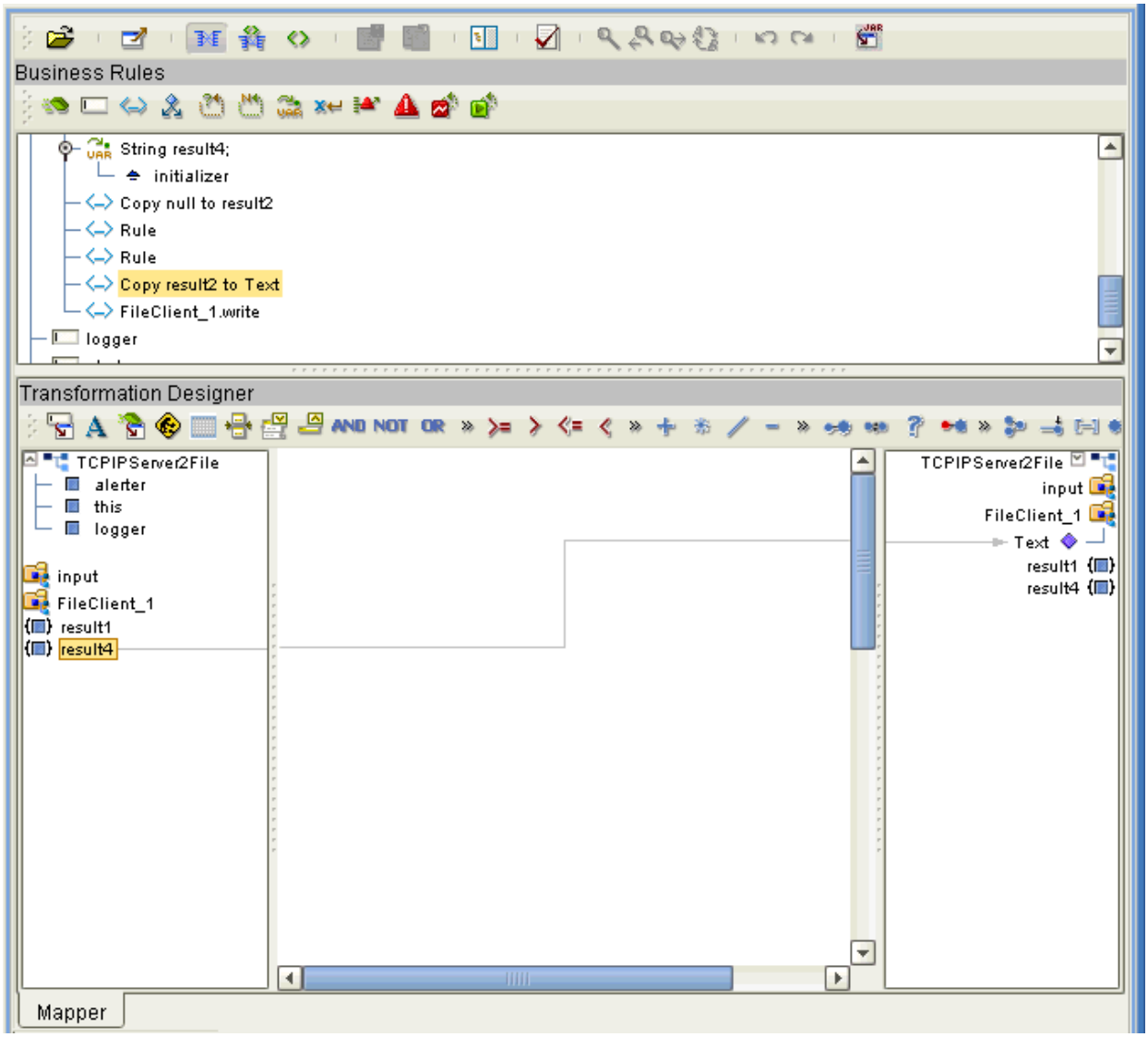
Figure 48 TCPIPServer2File Collaboration Definition: Eighth Business Rule



50 Click **Save** to save your changes.

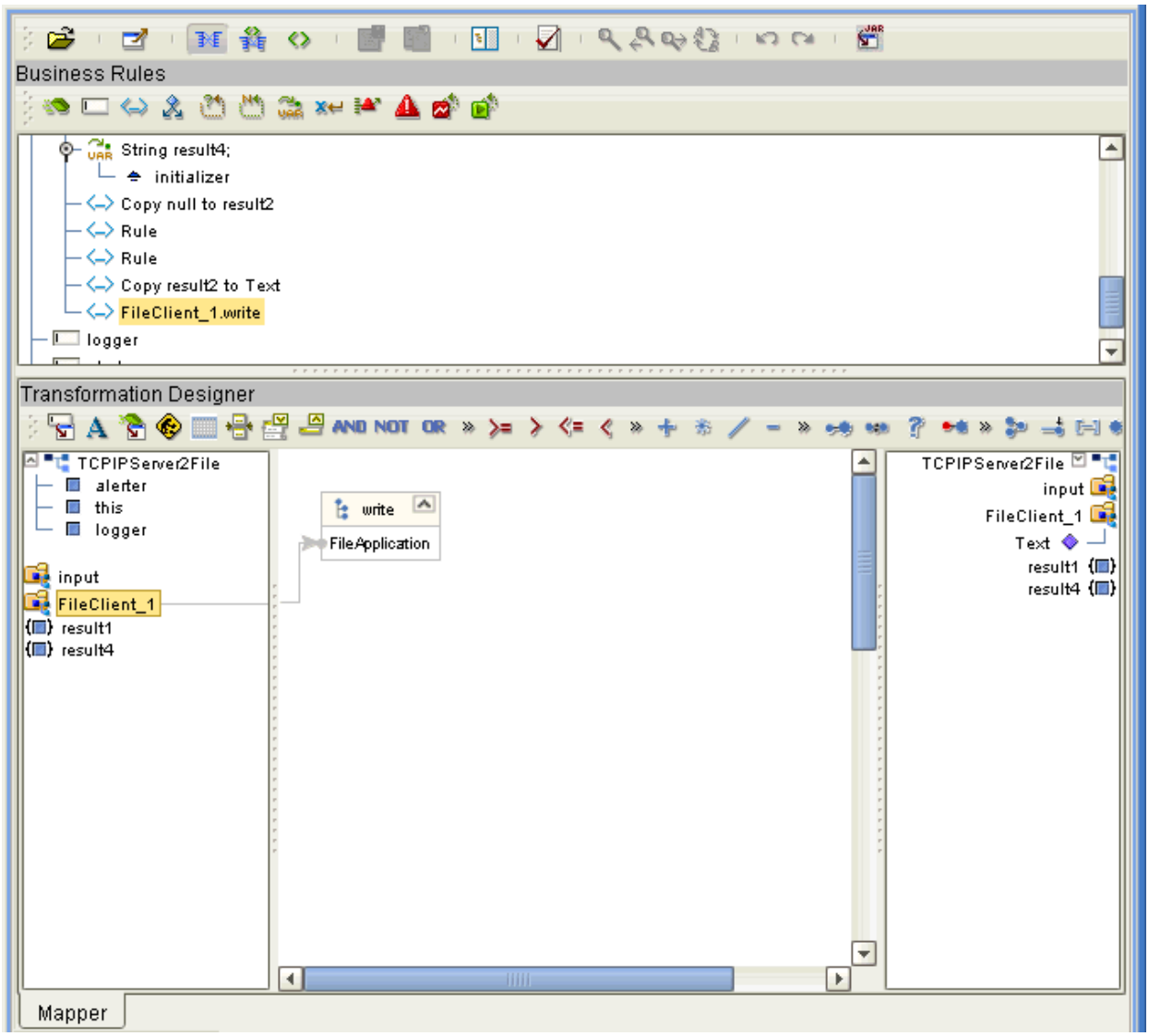
51 For the ninth Business Rule, drag the **result4** node in the left pane to the **Text** node under **FileClient_1** in the right pane. See Figure 49.

Figure 49 TCPIPServer2File Collaboration Definition: Ninth Business Rule



52 For the tenth Business Rule, start from the **FileClient_1** node in the left pane and create a **write** method. See Figure 50.

Figure 50 TCPIPServer2File Collaboration Definition: Tenth Business Rule



53 Click **Save** to save your new Java Collaboration Definition. If the validation shows any errors, use the exception information in the Validation pane to troubleshoot the errors.

4.4.7 Defining Collaborations

After you have finished with the Collaboration Definitions, you need to define all the Collaborations in the Connectivity Map.

To do this operation, you can drag and drop each Java Collaboration Definition icon from the **Project Explorer** tree onto its corresponding Collaboration component in the Connectivity Map as shown in Table 2.

Table 2 TCPIP_Project Collaborations and Their Definitions

Collaboration	Collaboration Definition
file2TCPIPClientCollab	File2TCPIPClient
TCPIPServer2FileCollab	TCPIPServer2File

If the Collaboration is successfully defined, the "gears" icon changes from red to green.

Note: *You can also drag the desired Collaboration Definition onto the Connectivity Map by itself and create a Collaboration without first creating the Service component.*

4.4.8 Binding OTDs in Collaborations

After you have set up all the Collaborations, you need to bind, that is, associate each OTD with its corresponding eGate component.

Using the Collaboration Binding Window

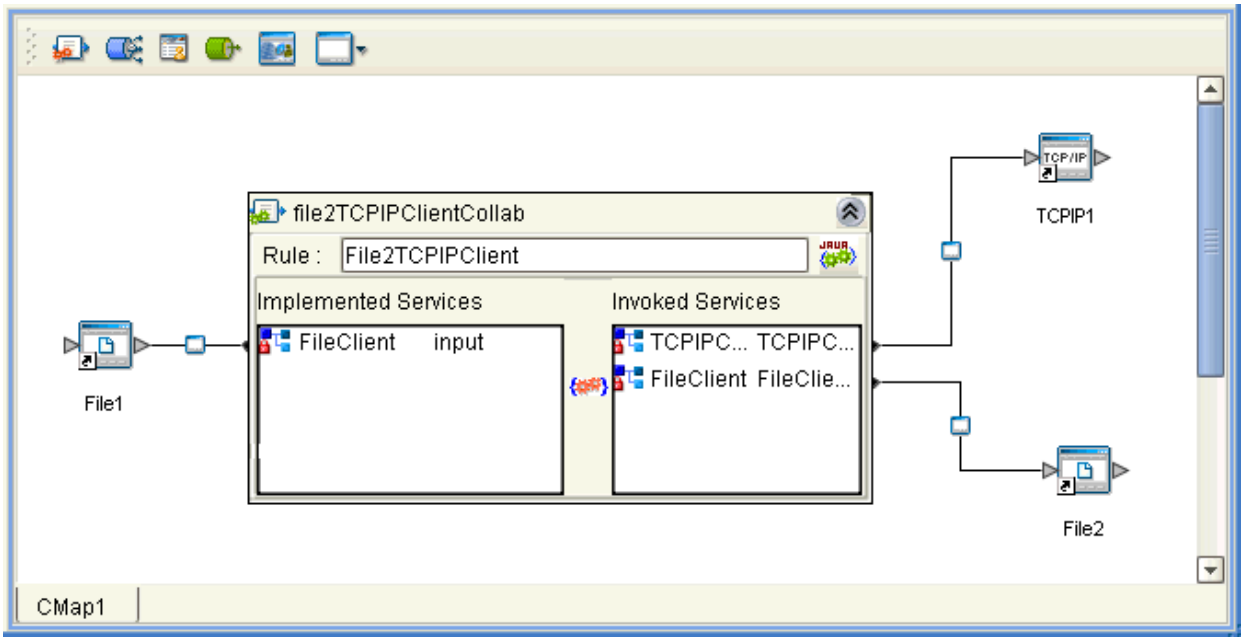
Use the eGate Enterprise Designer's Collaboration Binding window to associate OTDs and their corresponding eWay or Topic components.

To bind the OTDs in Collaborations

1 Open the **CMap1** Connectivity Map for the sample Project.

2 Double-click the **file2TCPIPClientCollab** Collaboration to open the Collaboration Binding window. See Figure 51.

Figure 51 Connectivity Map: Binding file2TCPIPClientCollab



3 Locate the **FileClient input** OTD in the **Implemented Services** pane on the **file2TCPIPClientCollab** window.

4 Click the **FileClient input** OTD and with the left mouse button depressed, drag it onto the Connectivity Map canvas and drop it onto the **File1** external application.

5 Following this same procedure, drag and drop the **TCPIPClient_1** OTD in the **Invoked Services** pane onto the **TCPIP1** external application.

6 Drag and drop the **FileClient_1** OTD in the **Invoked Services** pane onto the **File2** external application.

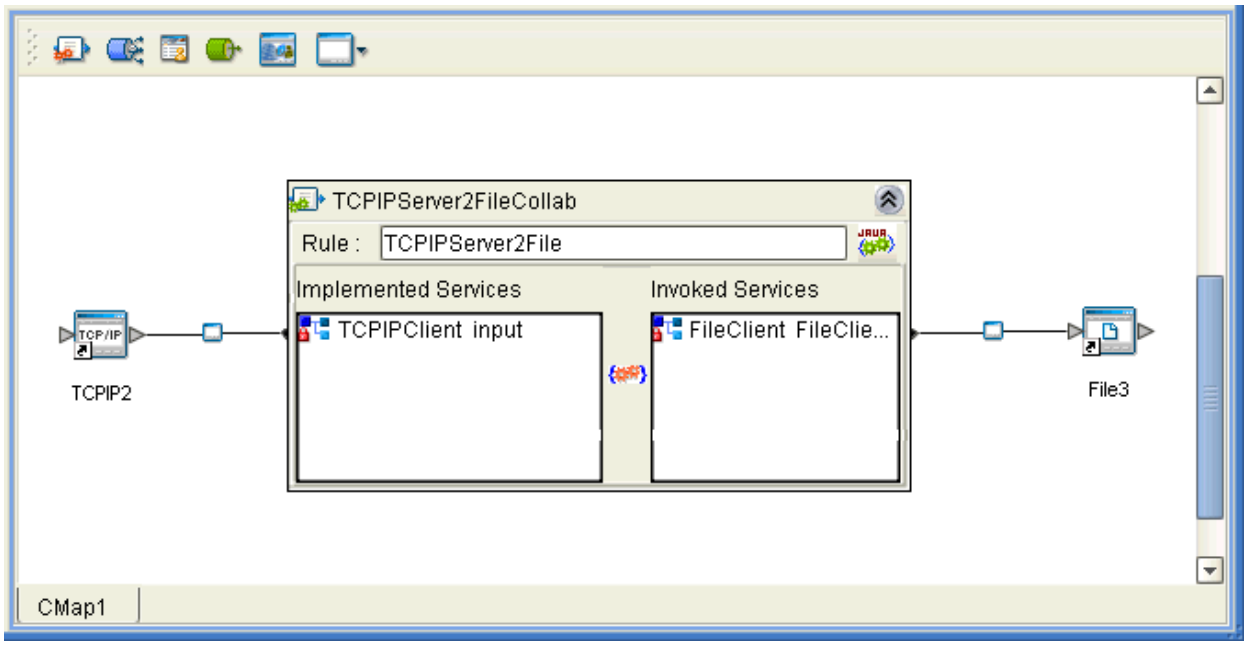
The Collaboration binding appears with the components bound to the appropriate OTDs. If the binding is done correctly, the lines leading out from each OTD definition in the **file2TCPIPClientCollab** window line up with each associated component.

Note: *For more information about assigning Collaboration Definitions and Collaboration Binding, see the* ***eGate Integrator User's Guide****.*

7 Click the **Close** button in the upper right corner of the Collaboration Binding window, to close the window.

8 Double-click the **TCPIPServer2FileCollab** Collaboration to open the Collaboration Binding window. See Figure 52.

Figure 52 Connectivity Map: Binding TCPIPServer2FileCollab



9 Locate the **TCPIPClient** OTD in the **Implemented Services** pane on the **TCPIPServer2FileCollab** window.

10 Click the **TCPIPClient** OTD and with the left mouse button depressed, drag it onto the Connectivity Map canvas and drop it onto the **TCPIP2** external application.

11 Following this same procedure, drag and drop the **FileClient_1** OTD onto the **File3** external application.

12 Close the **TCPIPServer2FileCollab** Collaboration Binding window.

13 To save all of your bindings, click **Save** on the Enterprise Designer toolbar.

4.4.9. Creating the Project's Environment

This section provides general procedures for creating an Environment for your Project. For a complete explanation, see the *eGate Tutorial*.

To create an Environment

1 From the Enterprise Designer, click the **Environment Explorer** tab on the Enterprise Explorer.

2 Under the current **Repository** icon in the **Environment Explorer**, create a new environment for your Project and name it as desired.

3 In the **Environment Explorer**, right-click the **Environment** icon and select the desired external systems from the pop-up menu. Include external systems for the TCP/IP eWays and the File eWays. Give them the same names as you did the corresponding external applications on the Connectivity Map.

4 Use the same pop-up menu to create a Logical Host for your Project, and name it as desired.

5 Click **Save** and return to the **Project Explorer** tab.

4.4.10 Setting eWay Properties

You must set the eWay properties for your specific system and for the current Project, using the eGate Enterprise Designer. For directions on accessing and using the eWay **Properties** sheet, as well as a complete explanation of the TCP/IP eWay properties, see **Chapter 3**.

To set the File eWay properties

1 From the **Project Explorer**, open the **CMap1** Connectivity Map for the sample Project.

2 To change the default properties for the inbound File eWay, click the **File1** external application's eWay icon. For this eWay, select the **Inbound** properties.

The eWay **Properties** sheet appears.

You can accept current default properties settings for the sample Project. However, if you are using different file names and/or folders than the defaults, you must enter the names of the files/folders you are using. For example, if you are using an input text file named **Data.txt**, you can change the **Input file name** property setting to ***.txt**.

Note: *Even if you do not change the eWay's properties, you must open each **Properties** sheet for every eWay and click **OK** to activate the eWay.*

3 Click **OK** to save the settings and close the eWay **Properties** sheet.

4 To change the default properties for the outbound File eWay, click the **File2** external application's eWay icon. For this eWay, select the **Outbound** properties.

Use your file and folder names if they differ from the defaults. For all other properties, you can use the defaults.

If you are using a text file, change the output file name setting from **output%d.dat** to **output%d.txt**.

5 Click **OK** to save the settings, close the window, and save.

6 For the **File3** eWay, follow the **Outbound** procedures given previously.

7 Click **OK** to save the settings, close the window and save.

To set the TCP/IP eWay properties

1. To begin changing the default properties for the TCP/IP eWay, click the **TCPIP1** external application's eWay icon on the Connectivity Map. For this eWay, select the **Client** properties.

   The eWay **Properties** sheet appears.

2. From the upper left pane of the eWay **Properties** sheet, select the desired folder.

3. The properties settings appear in the **Properties** pane on the left.

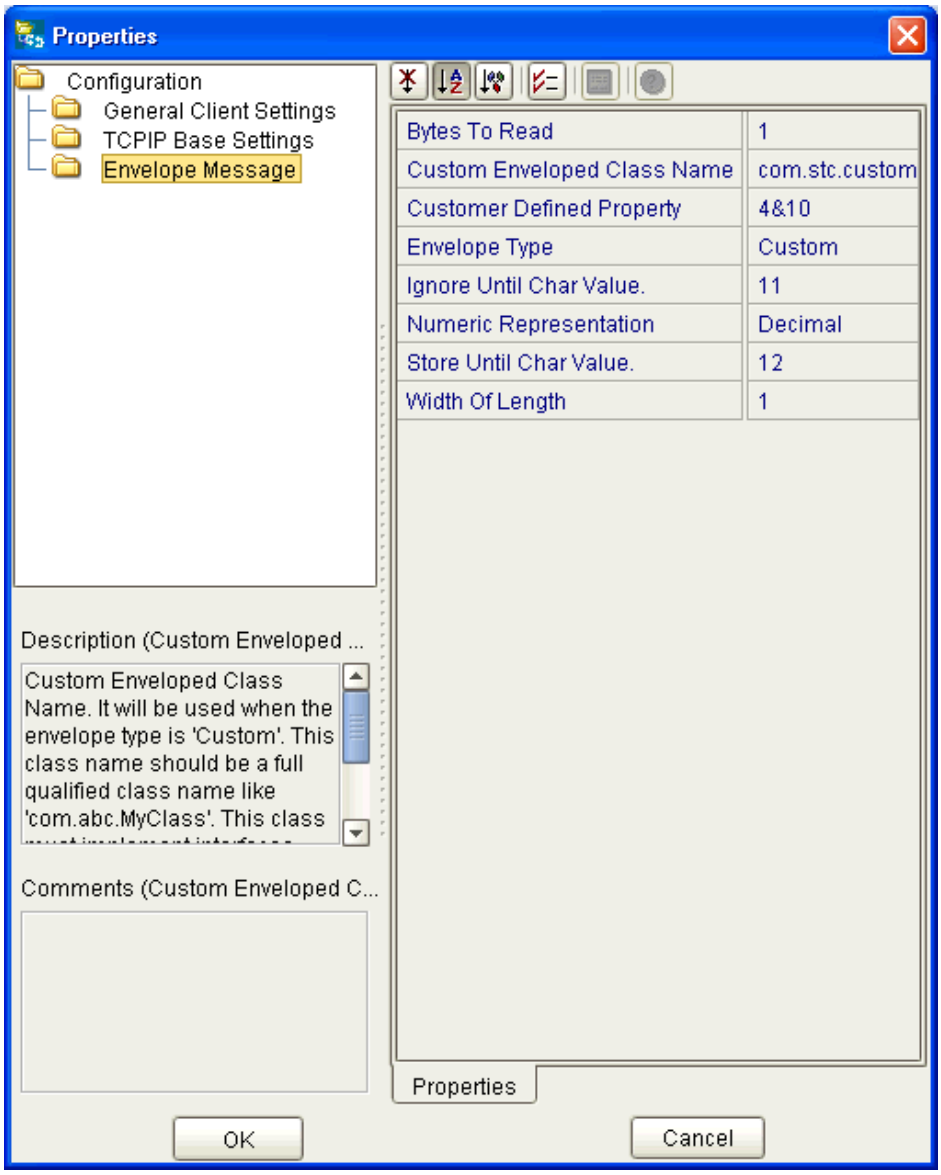
4. You can set the properties to the defaults except for the custom enveloping. Use the custom envelope file provided with the sample. For details on how to create and use this file, see **"Customized Enveloping" on page 30**.

   For both the server and client eWays, set the custom envelope properties as follows:

   - **Custom Enveloped Class Name: com.stc.customenvelope.CustomEnvelopedSample**
   - **Customer Defined Property: 4&10**

   See **Figure 53 on page 95**.

Figure 53 Custom Envelope Properties



5 Click **OK** to close the window and save.

6 To change the default properties for the next TCP/IP eWay, click the **TCPIP2** external application's eWay icon. For this eWay, select the **Server** properties.

The eWay **Properties** sheet appears.

7 From the upper left pane of the eWay **Properties** sheet, select the desired folder.

8 The properties settings appear in the **Properties** pane on the left.

9 Set the properties for file and folder settings in the same way as you did for the client mode. Use the same custom enveloping settings. For the other settings, you can use the defaults.

10 Click **OK** to close the window and save.

11 From the Enterprise Designer, click the **Environment Explorer** tab.

12 In the **Environment Explorer**, right-click the new Environment you created for your Project, and select **New TCPIP External System** for each system, client and server. Give these systems the same names as you gave the corresponding external applications (**TCPIP1** and **TCPIP2**) created on the Connectivity Map in the **Project Explorer**.

If you have already done these operations, you can skip this step.

13 In the left pane, right-click the **TCPIP2** (server) external application icon and select **Properties** from the pop-up menu.

The eWay **Properties** sheet appears.

14 From the upper left pane of the eWay **Properties** sheet, select the **Environment Configuration** folder.

15 The properties settings appear in the **Properties** pane on the left.

16 For the settings, use the defaults, *except* be sure that **ServerPort** is set to **8888**.

17 Click **OK** to close the window and save.

18 In the left pane, right-click the **TCPIP1** (client) external application icon and select **Properties** from the pop-up menu.

The eWay **Properties** sheet appears.

19 For the settings, use the defaults, *except* be sure that **Port** is set to **8888**.

20 Click **OK** to close the window and save.

Project deployment

For information on how to deploy your Project, see the next section.

4.5 Deploying a Project

This section provides general procedures for Project deployment.

4.5.1 Basic Steps

For a complete explanation of how to deploy and run an eGate Project, see the *eGate Tutorial*.

To deploy the Project

1 From the **Project Explorer**, select the current Project and right-click, choosing **New > Deployment Profile** from the pop-up menus.

2 From the **Create a Deployment Profile** dialog box, enter the name of the current Project and select the Environment you created for this Project.

3 Click **OK**.

The Deployment Profile canvas appears as follows:

- The Project's external applications and Services show up as icons on the left side of the canvas.
- The external systems and Logical Host you created under **"Creating the Project's Environment" on page 92** show up as windows on the right side of the canvas.

4 Set up your Deployment Profile by dragging the icons on the left into the corresponding windows on the right.

5 Click **Save All** then **Activate**.

When the Project has been activated, a pop-up message window appears stating the activation was successful.

For additional information, see the *eGate Integrator User's Guide* and *SeeBeyond ICAN Suite Deployment Guide.*

To run the Project

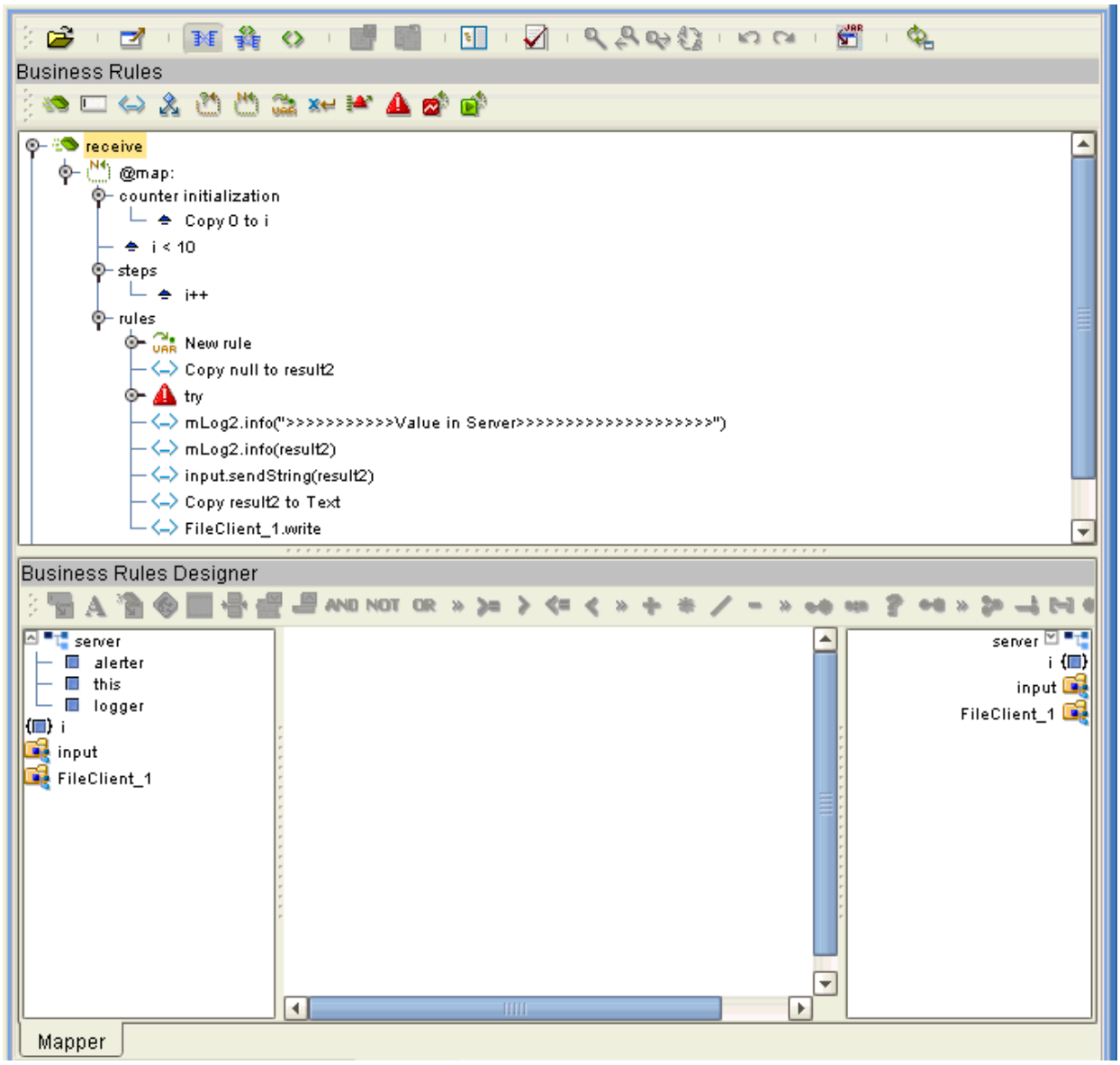
For instructions on how to run a Project, see the *eGate Tutorial*.

4.5.2 For and While Rules for Monitoring

It is a good practice to create a **for** and/or **while** rule (a loop) in any Java Collaboration using the server mode, if you want to start and stop the eWay frequently. This addition allows you to use the eGate Monitor to start and stop the eWay without problems. For more information, see the *eGate Integrator User's Guide*.

See **Figure 54 on page 98** for an example of a Java Collaboration with a **for** rule. For more information on how to create these rules, see the *eGate Integrator User's Guide*.

Figure 54 Java Collaboration With while Rule



4.5.3 Alerting and Logging

eGate provides an alerting and logging feature. This allows monitoring of messages and captures any adverse messages in order of severity based on configured severity level and higher. To enable Logging, see the *eGate Integrator User's Guide*.

4.6 Using the TCP/IP OTDs

You can use the TCP/IP OTDs via the eGate Enterprise Designer to set up the eWay as desired. The Collaboration Definition Editors and OTD Editor allow you to access OTD features via Windows drag-and-drop operations.

4.6.1 OTD Overview

An OTD contains a set of rules that define an object. The object encodes data as it travels through eGate. OTDs are used as the basis for creating Collaboration Definitions for a Project.

Each OTD acts as a template with a unique set of eWay features. The TCP/IP eWay OTD template is not customizable and cannot be edited.

The four parts of an OTD are:

- **Element**: This is the highest level in the OTD tree. The element is the basic container that holds the other parts of the OTD. The element can contain fields and methods.
- **Field**: Fields are used to represent data. A field can contain data in any of the following formats: string, boolean, int, double, or float.
- **Method**: Method nodes represent actual Java methods.
- **Parameter**: Parameter nodes represent the Java methods' parameters.

4.6.2 OTD Components

Each of the OTDs is made up of the following components:

- **OTD Operation**: The OTD is used in a Collaboration Definition to operate with eWays.
- **Definition and eGate Enterprise Designer**: An eWay Properties window provides a central location where you can define the eWay's properties. The Enterprise Designer also allows you to access this window.
- **eWay**: An eWay provides access to the information necessary to interface with a specified external application.

All OTDs must be set up and administered using the Enterprise Designer.

Note: *For complete information on how to use the Enterprise Designer and the eWay Properties window, see the* ***eGate Integrator User's Guide****.*

4.6.3 Basic OTD Operation

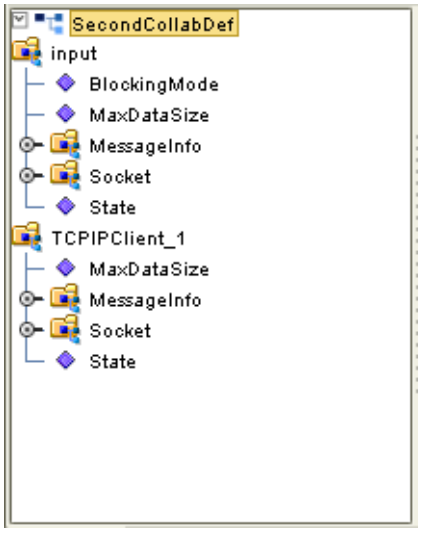
The TCP/IP eWay employs the following OTDs:

- **Server**: For TCP/IP *inbound* communication
- **Client**: For the TCP/IP *outbound* communication

These OTDs can perform inbound and outbound communication in a Collaboration either one direction at a time or simultaneously. Each OTD represents a deployable RA that includes TCP/IP (socket) functions plus an enveloped message model. The eWay's TCP/IP OTDs provide a basic TCP/IP client-and-server model that can handle all of the eWay's Java 2 Enterprise Edition (J2EE)-related connections.

Figure 55 shows the two OTDs' basic structure as it appears in the Java Collaboration Definition Editor window. The OTD with the root node labeled **input** is the server OTD, and the one with the **TCPIPClient_1** root node is the client. You can expand the nodes **MessageInfo** and **Socket** to reveal additional sub-nodes.

Figure 55 Java Collaboration Definition: TCP/IP OTDs



4.6.4 TCP/IP OTD Features

The eWay's TCP/IP server and client OTDs provide 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. See **"Envelope Type" on page 15** 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 30** 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** sheet 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** sheet 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 19** for details.
- **Server Endpoint Property Options**: You can set the eWay's server socket and message properties as desired.

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.

Chapter Topics

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.

5.1.1 Relation to eWay Configuration

The nature of this data transfer depends on the properties you set for the eWay in the eGate Enterprise Designer's eWay **Properties** sheet. For more information on the eWay's properties settings, see **Chapter 3**.

5.1.2 TCP/IP eWay Javadoc

For a complete list of the Java methods within the classes listed in this chapter, refer to the **Javadoc**. You can download the Javadoc while you are installing the eWay. For complete instructions, see the *SeeBeyond ICAN Suite Installation Guide*.

5.2 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.

Appendix A

Methods Not Used in the eWay

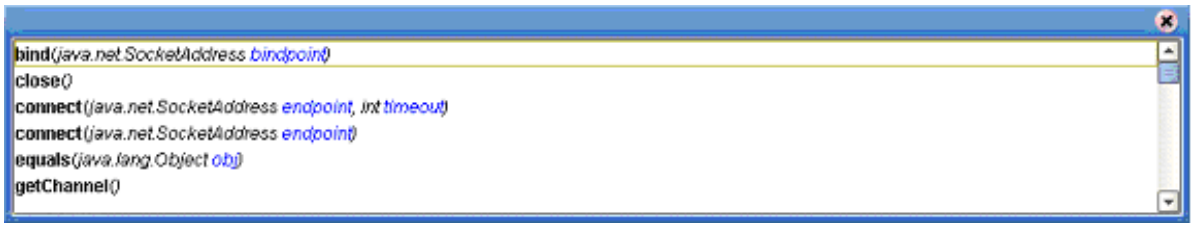
You may see some method nodes exposed in the TCP/IP Object Type Definitions (OTDs), which cannot be used in Java Collaborations.

Figure 56 on page 105 shows the methods available for **Channel** node that cannot be used in Java Collaborations.

Figure 56 Unused Channel Methods

Figure 57 shows the methods exposed in the Socket node that cannot be used by Java Collaborations.

Figure 57 Unused Socket Methods



Index

R

S

T

W