SUN SEEBEYOND

eWAY™ ADAPTER FOR SWIFTALLIANCE GATEWAY USER'S GUIDE

Release 5.1.3

Sun microsystems

Part Number: 820-1003

Version 20070420111640

Contents

Chapter 1

Introduction

This guide explains how to install, configure, and use the SWIFTAlliance Gateway eWay™. This chapter provides a brief overview of SWIFTAlliance Gateway and the SWIFTAlliance Gateway eWay Adapter, as well as an introduction to this document.

What's in This Chapter

1.1 About SWIFTAlliance Gateway

SWIFTAlliance Gateway is a modular software package that is installed on top of the SWIFTNet Link (SNL) software, and is designed to enable application-to-application communication. Using the SWIFTNet interactive services, InterAct and FileAct, messages and files are typically exchanged between a customer application (client) and a central application (server) over the Secure IP Network (SIPN). SWIFTAlliance Gateway can handle large volumes of information and is therefore suitable for use with both client and server applications.

1.1.1 Introduction to SWIFTNet

SWIFTNet is a global business messaging network for secure connectivity between institutions that participate in the financial services industry. As such, SWIFTNet is designed to satisfy institutional community requirements for inter-operability of mission-critical financial software solutions.

SWIFTNet provides an assurance of infrastructure reliability, availability, access control, correspondent and message authentication, message integrity, and confidentiality, to business applications that are interconnected among a community of institutions. Optionally, SWIFTNet also provides non-repudiation support, message validation, store-and-forward, and role-based access control.

1.1.2 SWIFTAlliance Gateway

SWIFTAlliance Gateway is an interface product for SWIFTNet. It incorporates all the functionality of the SWIFTNet Link. Additionally, it provides several different connectivity and usability features for SWIFTNet users, providing solutions to a variety of system integration problems.

SWIFTAlliance Gateway is designed to concentrate traffic from multiple SWIFTAlliance WebStations. It provides a graphical user interface for the administration of the SWIFTAlliance Gateway and related SWIFTNet security administration functions.

SWIFTAlliance Gateway can serve as a message concentrator, receiving messages from various other applications for passage through SWIFTNet. It can receive these messages through host adapters, including a WebSphere MQ host adapter, for interfacing with business applications running on a variety of different types of computing platforms.

1.1.3 SWIFTAlliance Gateway Remote API

SWIFTAlliance Gateway Remote API (RA) is a software package that establishes a communication link with the RA Host Adapter component of SWIFTAlliance Gateway, either from a SWIFTNet application existing on a remote computer or from a SWIFTNet application existing on the computer where SWIFTAlliance Gateway is installed.

Using Remote API, applications developed to run directly on top of SNL software can use SWIFTAlliance Gateway transparently as a concentrator for their SWIFTNet traffic, thereby implementing the single window concept RA offers two sets of APIs: SWIFTNet Link specific, and SWIFTAlliance Gateway specific. Message flow, from an RA instance to SWIFTAlliance Gateway, is managed by the Remote API Host Adapter (RAHA), a sub-component of SWIFTAlliance Gateway's Application Interface (AI).

For more information on configuration scenarios, see the *SWIFTAlliance Gateway Remote API Operations Guide*.

1.1.4 SWIFTNet Messaging Services

SWIFTNet offers four messaging services, SWIFTNet InterAct, FileAct, Browse, and FIN. Of these four, the SWIFTAlliance Gateway specifically addresses FileAct and InterAct in client mode, with both Real Time and Store-and-Forward transfers.

SWIFTNet InterAct

SWIFTNet InterAct provides secure and reliable exchange of individual structured financial messages. SWIFT customers' messaging requirements vary from customer to customer but also from message to message. SWIFTNet InterAct offers you a broad range of telecommunication modes.

Store-and-Forward Messaging

SWIFTNet InterAct's store-and-forward capability is designed for messages that are destined for a large number of correspondents, many of whom may not be online at the time of transmission. It removes the uncertainty and inconvenience of worrying about

whether or not your correspondents are on-line at the time you send the message. The message is delivered as soon as the recipient is ready to receive it. As a result, it provides an ideal way to send individual instructions, confirmations, and reports to large numbers of correspondents, some of whom may be in different time zones.

Real-Time Messaging

Real-time messaging offers a low-cost alternative to store-and-forward for messages which are destined for correspondents that are online at the time of transmission. As a result, it is ideal for sending individual instructions, confirmations, and reports to a few large correspondents, or for messages to market infrastructures.

SWIFTNet FileAct

SWIFTNet FileAct provides secure and reliable transfer of files, such as batches of structured financial messages or large reports. Typical applications include repetitive credit transfers such as pension or salary payments, securities value-added information and reporting, and regulatory reporting. SWIFTNet FileAct offers a variety of messaging modes.

Store-and-Forward File Transfers

SWIFTNet FileAct's store-and-forward capability ensures that your correspondents receive your message whether or they are online at the time of transmission. Messages are delivered when the recipient is ready to receive it. Store-and-Forward is an ideal way to send individual instructions, confirmations and reports to large numbers of correspondents, some of which may be in different time zones.

Real-time File Transfers

Real-time messaging provides a lower-cost alternative to store-and-forward for files that are destined for correspondents that are online at the time of transmission. This makes it ideal for sending files to a few large correspondents or market infrastructures.

Note: *For more information regarding SWIFTNet, SWIFTAlliance Gateway, and InterAct and FileAct services, see the SWIFTNet Service Design Guide and SWIFTAlliance Gateway Operations Guide.*

1.2 About the SWIFTAlliance Gateway eWay

The Sun SeeBeyond eWay Adapter for SWIFTAlliance Gateway (referred to as the **SWIFT AG eWay** throughout this guide) enables the Sun Java™ Composite Application Platform Suite to communication with SWIFTAlliance Gateway 5.0.

The SWIFT AG eWay is comprised of the following components:

- **Connector module**: a JCA 1.5 Resource Adapter, allows you to exchange messages or files across SWIFTNet, SWIFT's secure IP network.
- **NetBeans module**: incorporates the eWay into Java CAPS and provides necessary design time and runtime functionality within the Suite.

- **SWIFT AG Object Type Definition**: exposes SWIFTNet methods and attributes for use within a Java Collaboration to perform connectivity and business logic.

In addition to the OTD, the SWIFT AG eWay provides Connectivity Map and External System configuration for design time configuration.

SWIFT AG eWay Features

The 5.1 SWIFT AG eWay includes the following features:

- Supports InterAct and FileAct Services in client mode, with both Real Time and Store-and-Forward messaging
- Supports both synchronous and asynchronous operation modes
- Provides support for all the SWIFTNet Link (SNL) Primitives
- Supports dynamic configuration of InterAct and FileAct primitive attributes from the Java Collaboration Editor
- Supports dynamic configuration of SWIFT AG Remote API transport properties

SAGOutboundeWay Object Type Definition

The eWay provides a SWIFTAlliance Gateway specific OTD (Object Type Definition), which exposes methods, attributes, and configuration properties. When it is incorporated in a Java Collaboration, the SAGOutboundeWay OTD allows you to build powerful business logic into your Projects.

The SAGOutboundeWay OTD is comprised of the following nodes:

- **Configuration**: enables dynamic configuration of the eWay at runtime
- **Constants**: provides various SNL constants
- **Primitives**: provides all of the SNL Primitives for advanced users
- **RemoteApis**: provides user access to the Remote API's client APIs
- **Services**: provide the InterAct and FileAct client implementations to support Real Time and Store-and-Forward messaging

In addition to the OTD, the SWIFT AG eWay provides Connectivity Map and External System parameters for design time configuration.

1.3 What's New in This Release

The Sun SeeBeyond eWay™ Adapter for SWIFTAlliance Gateway includes the following new features:

New Features for Version 5.1.3

- This is a maintenance release. No new features.

New Features for Version 5.1.2

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

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

1.4 About This Document

This section provides a short description of the SWIFT AG eWay user's guide.

1.4.1 What's in This Document

This document provides information about installing, configuring, and using the SWIFT AG eWay and includes the following chapters:

- **Chapter 1 "Introduction"**provides an overview of SWIFTNet and SWIFTAlliance Gateway, as well as the SWIFT AG eWay and the guide.
- **Chapter 2 "Installing the eWay"**provides the supported operating systems and system requirements for the SWIFT AG eWay. It also includes directions for installing the SWIFT AG eWay and additional files, and accessing the accompanying documentation and sample Projects.
- **Chapter 3 "Configuring the eWay"** defines the SWIFT AG eWay properties and provides directions for configuring the SWIFT AG eWay properties at design time.
- **Chapter 4 "OTD Overview"**generally describes the OTD used by the eWay, and summarizes the OTD structure and functionality.
- **Chapter 5 "Implementing a Project Using Java Collaboration Definitions (JCD)"** describes the implementation and functionality of the SWIFT AG eWay using the eGate Integrator with the eWay's sample Projects.
- **Appendix A "Sample prjSAGCert Project Overview"** provides an overview of the prjSAGCert Project, an example of a Project which demonstrates several important scenarios.

SWIFTAlliance Gateway eWay Javadoc

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

1.4.2 Scope

This guide describes how to install and use the SWIFT AG eWay within the Sun Java Composite Application Platform Suite. Sample Projects are documented to demonstrate how Collaborations and Projects are created and implemented in a typical environment.

1.4.3 Intended Audience

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

1.4.4 Text Conventions

The following conventions are observed throughout this document.

Table 1 Text Conventions

Text Convention	Used For	Examples
Bold	Names of buttons, files, icons, parameters, variables, methods, menus, and objects	▪ Click **OK**. ▪ On the **File** menu, click **Exit**. ▪ Select the **eGate.sar** file.
`Monospaced`	Command line arguments, code samples; variables are shown in ***`bold italic`***	`java -jar ***filename***.jar`
Blue bold	Hypertext links within document	See **Text Conventions** on page 11
Blue underlined	Hypertext links for Web addresses (URLs) or email addresses	http://www.sun.com

1.5 Sun Microsystems, Inc. Web Site

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

http://www.sun.com

1.6 Documentation Feedback

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

CAPS_docsfeedback@sun.com

Chapter 2

Installing the eWay

This chapter explains how to install the SWIFT AG eWay, as well as supported operating systems and system requirements. The chapter also includes necessary post-installation procedures.

What's in This Chapter

2.1 SWIFT AG eWay System Requirements

The SWIFT AG eWay Readme contains the latest information on:

- Supported Operating Systems
- System Requirements
- External System Requirements

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

2.2 Installing the SWIFT AG eWay

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

2.2.1 Installing the eWay on a JavaCAPS Supported System

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

1. From the Installer's **Select Sun Java™ Composite Application Platform Suite Products to Install** table (Administration tab), expand the **eWay** option.
2. Select the products for your Suite and include the following:
   - **FileeWay** (the File eWay is used by most sample Projects)
   - **SwiftAGeWay**

   To upload the SWIFT AG eWay User's Guide, Help file, Javadoc, Readme, and sample Projects, select the following:
   - **SwiftAGeWayDocs**
3. Once you have selected all of your products, click **Next** in the top-right or bottom-right corner of the **Select Sun Java™ Composite Application Platform Suite Products to Install** box.
4. From the **Selecting Files to Install** box, locate and select your first product's SAR file. Select the SAR file and click **Next**. Your next selected product appears. Follow this procedure for each of your selected products. The **Installation Status** window appears and installation begins after the last SAR file has been selected.
5. Once your product's installation is finished, continue installing the Sun Java™ Composite Application Platform Suite as instructed in the *Sun Java™ Composite Application Platform Suite Installation Guide.*

The SWIFT AG eWay also requires additional JNI component files that are included with the installation. For directions on how to install these additional files see **"Adding the SWIFTAlliance Gateway Component Package" on page 16**

Adding the eWay to an Existing Suite Installation

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

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

2.2.2 Installing eWay Enterprise Manager plug-ins

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

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

The **eWay Enterprise Manager plug-ins** are available from the **List of Components to Download** under the Installer's **DOWNLOADS** tab.

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

1. From the Enterprise Manager:
   - A. From the **Enterprise Manager**'s Explorer toolbar, click the **Configuration** icon.
   - B. Click the **Web Applications Manager** tab, go to the **Auto-Install from Repository** tab, and connect to your Repository.
   - C. Select the application plug-ins you require, and click **Install**. The application plug-ins are installed and deployed.
2. From the **Sun Java™ Composite Application Platform Suite Installer:**
   - A. From the Installer's **Download** tab, select the Plug-Ins you require and save them to a temporary directory.
   - B. Log onto the **Enterprise Manager**. From the **Enterprise Manager**'s Explorer toolbar, click the **Configuration** icon.
   - C. Click the **Web Applications Manager** tab and the **Manage Applications** tab.
   - D. Browse for and select the WAR file for the application plug-in that you downloaded, and click **Deploy**. The plug-in is installed and deployed.

SWIFT AG eWay Alert Codes

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

To View the eWay Alert Codes

1. Add the eWay Enterprise Manager plug-in for this eWay.
2. From the Enterprise Manager's **Explorer** toolbar, click the **Configuration** icon.
3. Click the **Web Applications Manager** tab and go to the **Manage Alert Codes** tab. Your installed alert codes are displayed under the **Results** section. If your eWay alert codes are not available displayed under **Results**, do the following
   - A. From the **Install New Alert Codes** section, browse to and select the eWay alert properties file for the application plug-in that you added. The alert properties

files are located in the **alertcodes** folder of your Sun Java Composite Application Platform Suite installation directory.

B Click **Deploy**. The available alert codes for your application are displayed under **Results**. A listing of available this eWay's alert codes is displayed in Table 2.

Table 2 SWIFT AG eWay Alert Codes

Alert Code	Description	User Action
SAGEWAY-CONNECT-FAILED000001=Failed to connect	Failed to establish a SWIFTNet connection.	Typically, detailed error information is placed in the Integration Server log, explaining the failure and the corresponding user actions required.
SAGEWAY-EXCHANGE-FAILED000002=Failed to exchange message	Failed to exchange message with SWIFT.	

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

2.2.3 After Installation

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

2.3 Adding the SWIFTAlliance Gateway Component Package

The SWIFTAlliance Gateway (SAG) Component package provides OS-specific code and JNI-wrapper code, which allows the SWIFT AG eWay to connect properly with SWIFTAlliance Gateway.

After uploading the **SwiftAGeWay.sar** file, download the appropriate SWIFTAlliance Gateway (SAG) Component package for your system from the Sun Java™ Composite Application Platform Suite Installer.

The download package includes the following files:

SWIFTAlliance Gateway (SAG) Component for Solaris or AIX

- libstcsagjni.so
- sagjni.jar

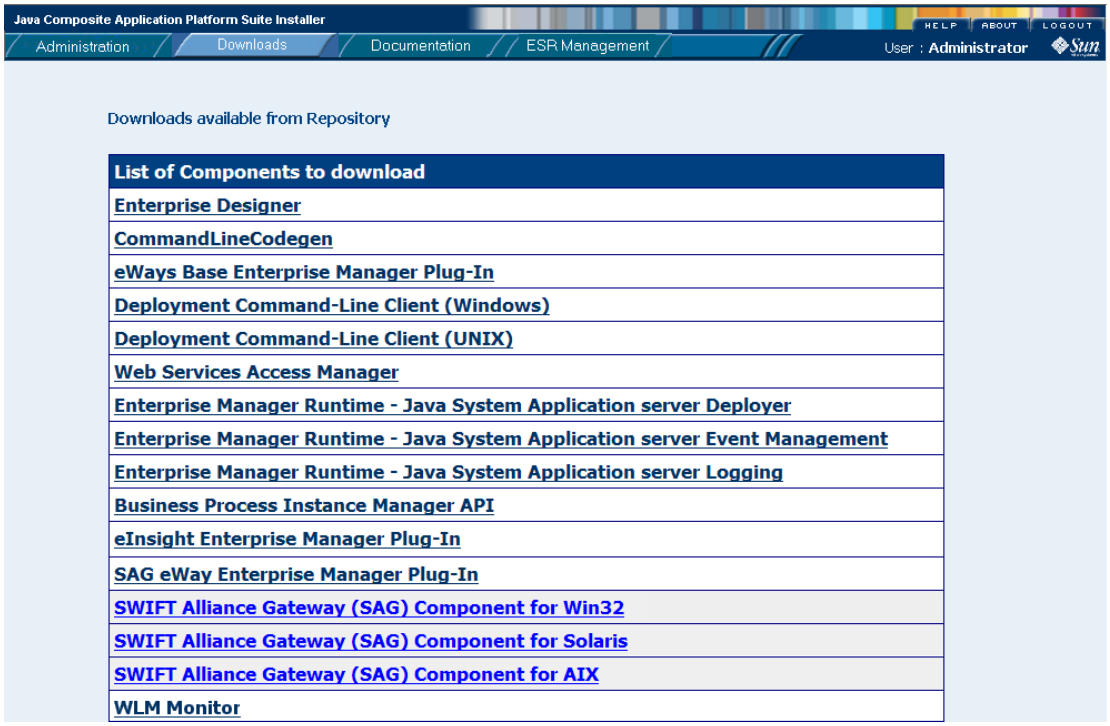
SWIFTAlliance Gateway (SAG) Component for Win32

- stcsagjni.dll
- sagjni.jar

To download the component package for your system, do the following:

1 From the Sun Java™ Composite Application Platform Suite Installer, click the **DOWNLOADS** tab. The List of Components to download displays three system specific SWIFTAlliance Gateway (SAG) Component packages (see Figure 1).

Figure 1 Sun Java Composite Application Platform Suite Installer - Downloads



2 Extract the package for your specific system to a local directory.

3 Copy both files for your system to the following location:

```
<JavaCAPS51>/logicalhost/is/lib
```

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

Note: *Before deploying and running your project on another supported application server, copy both files, sagjni.jar and libstcsagjni.so (for Solaris and AIX) or stcsagjni.dll (for Windows) to the following location: <application server>/lib, where <application server> is the installation directory of your specific application server.*

2.4 Installing and Initializing the SWIFT AG Remote APIs

Install the SWIFT Alliance Gateway Remote API (RA) on your JavaCAPS Integration Server host. Initialize (start) and run the RA prior to starting your JavaCAPS Integration Server (IS) and deploying your Project. Running the JavaCAPS IS on top of the RA fulfills the required environment variables.

Note: *The following directions assume that you will only have a single instance of the Remote API connecting with the Remote API Host Adapter of a single SAG instance. for different scenarios, see the SWIFTAlliance Gateway Remote API Operations Guide for more information.*

Install the Remote API

1. From the JavaCAPS Integration Server host, install the SWIFTAlliance Gateway Remote API. See the *SWIFTAlliance Gateway Remote API Installation Guide* for directions. The installation process creates the configuration file, sagta_ra.cfg, containing the information necessary to connect to the SAG host. You will need the following information to install RA on your JavaCAPS Integration Server host:
   - Hostname or IP Address of the SAG server
   - Port number used by SAG to communicate with remote applications (default is 48002)
   - Port number that SAG uses for remote file transfer (default is 48003)
2. Initialize (start) the RA. This is done using the **swiftnet init** command. The steps to initialize RA vary depending on your operating system. To initialize the RA, do the following:

   For Solaris and AIX:

   A. Open the Korn shell.

   B. Navagate to the `/SWIFTAlliance/RA/bin` directory.

   C. Enter the following swiftnet init command:
   ```
   . ./swiftnet init
   ```
   If you have more than one instance of the Remote API configured, use the following syntax:
   ```
   . ./swiftnet init -S <RA name>
   ```
   where <*RA name*> is the name of your Remote API instance.

   For Windows:

   Your RA installation creates a shortcut on your desktop. Double-click the shortcut to initialize RA. To initialize RA from a command prompt, do the following:

   A. From the command prompt, navagate to the `\SWIFTAlliance\RA\bin` directory.

   B. Enter the following swiftnet init command:
   ```
   swiftnet init
   ```

If you have more than one instance of the Remote API configured, use the following syntax:

```
swiftnet init -S <RA name>
```

where <*RA name*> is the name of your Remote API instance.

3 With your RA initialized, start the JavaCAPS Integration Server.

2.5 Configuring the Application ID

For the sample Projects, the application interface message partner name is **my_AI**. To configure this message partner for SWIFTAlliance Gateway, do the following:

1 From the SAG Admin Interface, click **Application Interface**. The Application Interface module appears.

2 From the Application Interface module toolbar, click **Add a new Message Partner**. the Message Partner Detail dialog box appears.

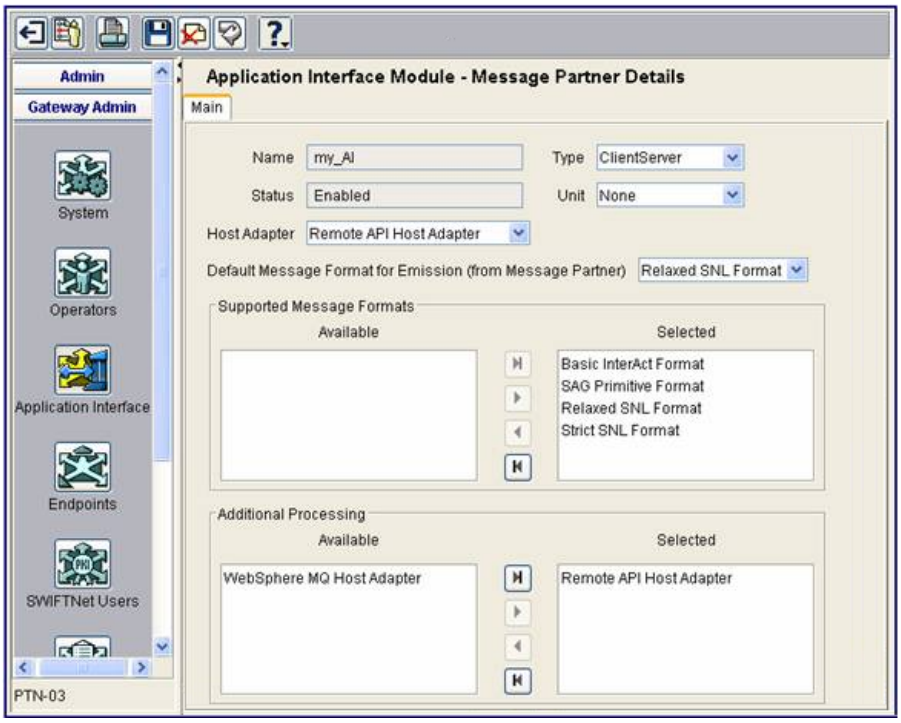
3 From the Message Partner Detail dialog box (see **Figure 2 on page 20**), enter the following values :

- Name: my_AI
- Type: ClientServer
- Unit: None
- Supported Message Formats Selected: Basic InterAct Format
  Sag Primitive Format
  Relaxed SNLFormat
  Strict SNLFormat
- Additional Processing Selected: Remote API Host Adapter

4 Save your changes and close the Message Partner Detail dialog box.

5 From the Application Interface Module, Message Partners tab, right-click the **my_AI** message partner and select **Enable** from the shortcut menu.

Figure 2 SWIFTAlliance Gateway Application Interface Module



For more information on configuring SWIFTAlliance Gateway, see the SWIFTAlliance Gateway Operations Guide.

Chapter 3

Configuring the eWay

This chapter describes how to create and configure the SWIFTAlliance Gateway eWay.

What's in This Chapter

3.1 Configuring the SWIFTAlliance Gateway eWay

All eWays contain a set of properties that are unique to that eWay type. After the eWays are created and a SWIFTAlliance Gateway External System is added to the Project's Environment, the eWay parameters can be modified for your specific system. The SWIFTAlliance Gateway eWay properties are modified from two locations:

- From the **Connectivity Map**. These properties most commonly apply to a specific eWay, and may vary from other eWays (of the same type) in the Project.
- From the **Environment Explorer tree**. These properties are commonly global, and apply to all eWays (of the same type) in the Project. The saved properties are shared by all eWays in the SWIFTAlliance Gateway External System window.

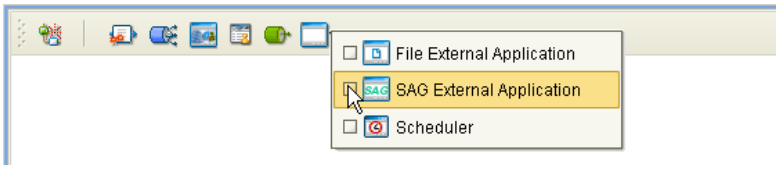
3.1.1 Selecting SWIFTAlliance Gateway as the External Application

To create an SWIFTAlliance Gateway eWay, you must first create an SAG (SWIFTAlliance Gateway) External Application in your Connectivity Map. SWIFTAlliance Gateway eWays are located between the SAG External Application and a Service. Services are containers for Collaborations, Business Processes, eTL processes, and so forth.

Creating the SAG External Application

1. From the Connectivity Map toolbar, click the **External Applications** icon.
2. Select the **SAG External Application** from the menu (see **Figure 3 on page 22**). The selected SAG External Application icon appears on the Connectivity Map toolbar.

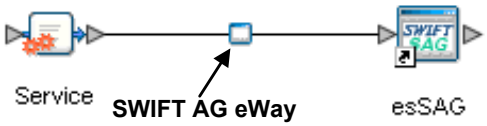
Figure 3 External Applications Selection Menu



3 Drag the new **SAG External Application** from the toolbar onto the Connectivity Map canvas. This represents an external SWIFTAlliance Gateway system.

From the Connectivity Map, you can associate (bind) the External Application with the Service to establish an eWay (see Figure 4).

Figure 4 eWay Location



When SAG is selected as the External Application, it automatically applies the default SWIFTAlliance Gateway eWay properties, provided by the OTD, to the eWay that connects it to the Service. You can then modify these properties for your specific system using the **Properties Editor**.

3.1.2 Modifying the SWIFTAlliance Gateway eWay Properties

A Project's eWay properties can be modified after the eWays have been created in the Connectivity Map and the External Systems have been added to the Project's Environment.

Modifying the SWIFT AG eWay Connectivity Map Properties

1 From the Connectivity Map, double-click the eWay icon located in the link between the associated Service and the External Application. The eWay **Properties Editor** appears containing the Connectivity Map properties.

2 Make any necessary modifications and click **OK.** The Property Editor closes saving your new settings.

Modifying the SWIFT AG eWay Environment Properties

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

2 Make any necessary modifications to the Environment parameters of the SWIFT AG eWays, and click **OK** to save the settings.

3.1.3 Using the Properties Editor

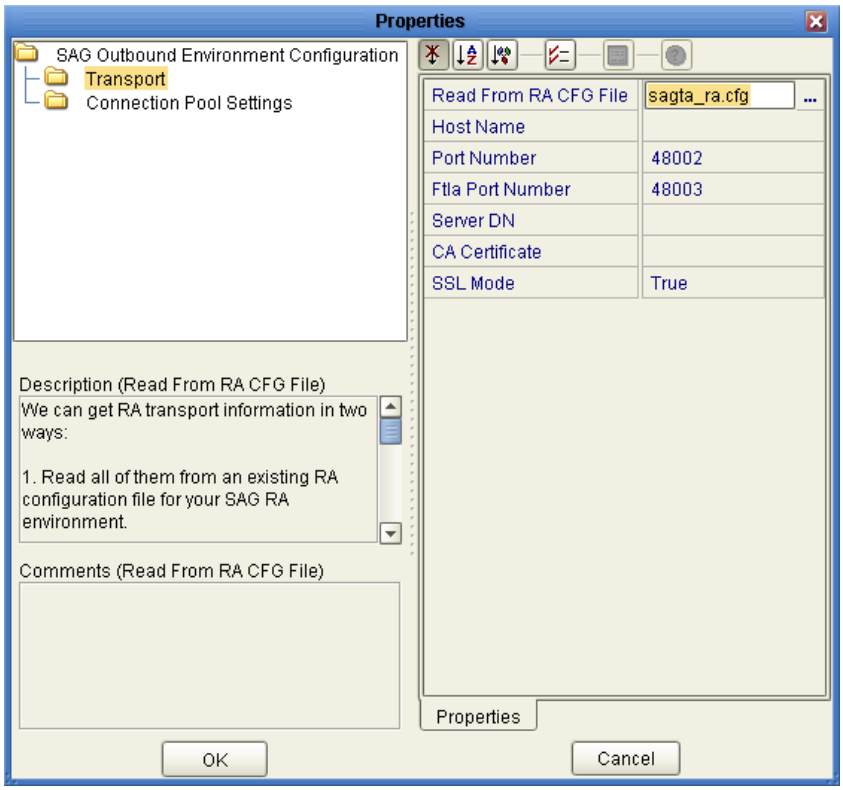
You can modify the current eWay configuration properties from the SWIFT AG eWay Properties Editor.

To modify the eWay properties do the following:

1 Open the SWIFT AG eWay Properties Editor for the eWay you want to edit. The SWIFT AG eWay has two sets of parameters: those specific to that particular eWay (accessed from the **Connectivity Map**), and those that are common to all eWays of this type (accessed from the **Environment Explorer** tree).

2 From the upper-left pane of the Properties Editor, select a properties directory from the Configuration tree. The parameters contained in that directory are now displayed in the right pane of the Properties Editor. For example, from the outbound eWay Connectivity Map Properties, click on **Transport** to display this section's editable parameters in the right pane, as shown in Figure 5.

Figure 5 Properties Editor -- SWIFT AG eWay Environment Properties



3 Click on any property field to make it editable. For example, click on the **Read From RA CFG File** parameter to edit the **Read From RA CFG File** value.

If a parameter's value is true/false or multiple choice, the field, when selected, reveals a submenu of property options. If a parameter requires that you type in a value, such as a name or password, the property field provides space to type in the value and an ellipsis (. . .) button.

Click on the ellipsis (. . .) in the properties field to open a separate configuration dialog box. This is helpful for entering large values that cannot be fully displayed in the parameter's property field. Enter the property value in the dialog box and click **OK**. The value is now displayed in the property field.

4 A description of each parameter is displayed in the **Description** pane when that parameter is selected, providing an explanation of any required settings or options.

5 The **Comments** pane provides an area for recording notes and information regarding the currently selected parameter. This is saved for future reference.

6 After modifying the configuration properties, click **OK** to close the Properties Editor and save the changes.

3.2 SWIFTAlliance Gateway eWay Properties

The SWIFT AG eWay's Properties are organized as follows:

SWIFT AG eWay Connectivity Map Properties on page 25

SWIFT AG eWay Environment Properties on page 40

Important: *Modifying individual OTD configuration settings can override the default eWay OTD configuration settings.*

Note: *For more information on SWIFTAlliance Gateway configration properties in the context of SWIFTAlliance Gateway, refer to the SWIFTAlliance Gateway and SNL user documentation provided by SWIFT.*

3.3 SWIFT AG eWay Connectivity Map Properties

The SWIFTAlliance Gateway eWay Connectivity Map Properties include the following parameter sections:

3.3.1 Envelope

The **Envelope** section of the SWIFT AG eWay Connectivity Map properties contains the top-level parameters displayed in Table 3.

Table 3 Connectivity Map - Envelope

Name	Description	Required Value
Application ID	Specifies the name of the Application Interface Message Partner. The Message Partner basically identifies an application that sends and receives messages on behalf of a user.	The name of the Application Interface Message Partner. This property is mandatory
Context ID	Specifies the cryptographic mode for Relaxed SNL Protocol: ▪ Empty (blank) indicates automatic mode. ▪ Advanced: indicates advanced mode. Relaxed SNL Protocol does not support manual cryptographic mode.	**Advanced** or ***blank*** indicating the selected Context ID. The configured default is **Advanced**

Table 3 Connectivity Map - Envelope (Continued)

Name	Description	Required Value
Msg Format	Specifies the format of the current message. Each of the names of the required values denotes a SWIFTAlliance Gateway message format.	The default value is Sag:RelaxedSNL. ***Note:*** This property is "grayed out" to indicate that the property is not configurable.
Sender	Specifies the name of the SWIFTAlliance Gateway Sender. This is an SAG operator.	The name of the sender.
Sender Auth	Specifies the password of the SAG operator.	The SAG operator password.

3.3.2 Primitive Control

The **Primitive Control** section of the SWIFT AG eWay Connectivity Map properties contains the top-level parameters displayed in Table 5.

Table 4 Connectivity Map - Primitive Control

Name	Description	Required Value
Include XML Attributes in SNL Primitive	Specifies the construction/marshalling of SNL primitives. When the SWIFT AG eWay constructs/marshals the SNL primitives, this flag indicates whether the eWay includes the XML attributes in the primitives. For example: ▪ For element SwInt:Requestor, including XML attributes, may be as follows: `<SwInt:Requestor type="Sw.Gbl.DN" version="4.0.0">o=swift,o=swift</ SwInt:Requestor>` ▪ If you do not include XML attributes;, it may be as follows: <SwInt:Requestor>o=swift,o=swift</ SwInt:Requestor> if not include XML attributes. Note that all XML attributes in SNL Primitives are defined in SNL Specification.	Select **True** or **False**. True indicates that the eWay includes the XML attributes in the primitives. The configured default is **FALSE**.

3.3.3 RemoteApi Base settings

The **RemoteApi Base settings** section of the SWIFT AG eWay Connectivity Map properties contains the top-level parameters displayed in Table 5.

Table 5 Connectivity Map - RemoteApi Base settings

Name	Description	Required Value
Client Handle Timeout	Specifies the maximum time (in milliseconds) allowed between a request message and its corresponding response message.	An integer indicating the maximum time allowed between a request message and its corresponding response message in milliseconds (for example, 300000 equals 5 minutes). The configured default is **300000**.
Read Blocking Timeout	Specifies the maximum time (in milliseconds) to block a read operation	An integer indicating the maximum time in milliseconds to block a read operation (for example, 60000 equals 1 minute). The configured default is **60000**.

3.3.4 InterAct Client

This section maps to Primitive **SwInt:ExchangeRequest**. For parameters SwXXX under this section, refer to the SNL specification for more detailed descriptions. The **InterAct Client** section of the SWIFT AG eWay Connectivity Map properties contains the top-level parameters displayed in Table 5.

Table 6 Connectivity Map - InterAct Client

Name	Description	Required Value
User DN	Specifies the SwSec_UserDN, full distinguished name used to identify and authenticate the principal. Size is limited to 100 bytes.	The User DN (distinguished name) ending with o=<BIC8>,o=swift For example: cn=dandrews,o=bankn14b,o=swift
Requestor DN	Specifies the SwInt_Requestor, used to identify the Application entity name. Size is limited to 100 bytes.	The Requestor DN (distinguished name) ending with o=<SWIFTNet institution BIC-8>,o=swift For example: ou=management,o=bankn14b,o=swift
Responder DN	Specifies the SwInt_Responder, used to identify the intended responder. Size is limited to 100 bytes.	The Responder DN (distingusihed name) ending with o=<SWIFTNet institution BIC-8>,o=swift For example: cn=management,o=swift,o=swift
Service Name	Specifies the SwInt_Service, the Service Name containing the SWIFTNet service used. The size is limited to 30 bytes.	The Service Name. For example, swift.cte, swift.generic.ia!x, swift.generic.iast!x, and so forth.
Request Type	Specifies the SwInt_RequestType, used to identify the message type of the XML message using the standard message code. The size is limited to 30 bytes.	The Request Type. For example: camt.005.001.02 (GetTransaction)
User Reference	Specifies the SwInt_RequestRef. This is used to associate a request message with subsequent response or error messages. The size is limited to 30 bytes.	A user reference of 30 bytes or less.
Signed?	Specifies whether the request contains Crypto operations to be performed. Only the last Crypto block is analyzed. SwInt_RequestCrypto (digital signature).	Select **True** or **False**. **True** indicates that the request contains Crypto operations to be performed. The configured default is **FALSE**.

Table 6 Connectivity Map - InterAct Client (Continued)

Name	Description	Required Value
Priority	Specifies the SwInt_Priority. The priority of delivery. In the future, SWIFTNet may implement this priority through schemes such as top-queuing, dedicated server processes, network transport priority. In the current implementation, differentiation on priority may be used for store-and-forward delivery.	**Normal** or **Urgent**. The configured default is **Normal**.
Non-Repudiation?	Specifies whether SwInt_NRIndicator, non-repudiation support is being requested.	Select **True** or **False**. True indicates that non-repudiation is requested. The configured default is **FALSE**.
Delivery Notification Queue Name	Specifies the SwInt_NotifQueue, store-and-forward delivery mode. When a value is present, it indicates SnF delivery mode and the queue where SnF delivery notifications are received. In the case of non-delivery (Rejected or Failed message), an SnF failed notification is always generated in this notification queue. In the case of delivery (Accepted or Duplicated message) an SnF delivery notification is optionally generated in this queue.	For store-and-forward delivery mode, enter the name of the delivery notification queue. Size is limited to 30 bytes. For example: ptsauszz_generic!x. (see Sw:DeliveryNotif)
Ask Positive Delivery Notification?	Specifies Sw_DeliveryNotif, delivery notification. This is for store-and-forward delivery mode only. Indicates whether a delivery notification is required in case of successful delivery (Accepted or Duplicated).	Select **True** or **False**. **True** indicates that a delivery notification is required. The configured default is **FALSE**.

3.3.5 InterAct Client > Store and Forward

This section maps to the parameter **Sw:AcquireSnFRequest** of the Primitive **Sw:ExchangeSnFRequest**. For parameters SwXXX under this section, refer to the SNL specification for more detailed descriptions. The **InterAct Client > Store and Forward** section of the SWIFT AG eWay Connectivity Map properties contains the top-level parameters displayed in Table 7.

Note: *For more information on SWIFTAlliance Gateway configration properties in the context of SWIFTAlliance Gateway, refer to the SWIFTAlliance Gateway and SNL user documentation provided by SWIFT.*

Table 7 Connectivity Map - InterAct Client > Store and Forward

Name	Description	Required Value
Queue Name To Acquire	Specifies the name of the queue (SwInt_Queue).	The User SwInt_Queue For example: ptsauszz_generic!x Size is limited to 30 bytes.
Force Acquire?	Specifies the Sw_ForceAcquire. Indicates whether an acquisition request must be accepted in case the queue is already acquired.	Select **TRUE** or **FALSE**. **True** indicates that an acquisition request must be accepted. The configured default is **TRUE**.
Session Mode	Specifies the usage mode of the queue. This property is purposely disabled.	The default setting is **Pull**. Pull is the correct setting for *client* mode ***Note:*** This property is purposely disabled.
Order By	Specifies the Sw_OrderBy. Indicates the order priority in which the messages are retrieved on the queue. The options are: ▪ InterAct ▪ Blank: indicates FIFO (first in, first out). ▪ FileAct ▪ Urgent	Select **InterAct**, ***blank***, **FileAct**, or **Urgent**. The configured default is **InterAct**.
Recovery Mode?	Specifies the Sw_RecoveryMode. Indicates whether the session must be opened in recovery mode. Messages that already have an output sequence number are sent first, before considering the selected order.	Select **TRUE** or **FALSE**. **True** indicates that the session must be opened in recovery mode. The configured default is **FALSE**.

3.3.6 FileAct Client

This section maps to the Primitive **Sw:ExchangeFileRequest**. For parameters SwXXX under this section, refer to the SNL specification for more detailed descriptions. The **FileAct Client** section of the SWIFT AG eWay Connectivity Map properties contains the top-level parameters displayed in Table 8.

Table 8 Connectivity Map - FileAct Client

Name	Description	Required Value
User DN	Specifies the SwSec_UserDN. Used to identify and authenticate the principal. Size is limited to 100 bytes.	The User DN (distinguished name) ending with o=<BIC8>,o=swift For example: cn=dandrews,o=bankn14b,o=swift
Requestor DN	Specifies the SwInt_Requestor, the Application entity name. Size is limited to 100 bytes.	The Requestor DN (distinguished name) ending with o=<SWIFTNet institution BIC-8>,o=swift For example: ou=management,o=bankn14b,o=swift
Responder DN	Specifies the SwInt_Responder, the name of the intended responder. Size is limited to 100 bytes.	The Responder DN (distingusihed name) ending with o=<SWIFTNet institution BIC-8>,o=swift For example: cn=management,o=swift,o=swift
Service Name	Specifies the SwInt_Service, the Service Name containing the SWIFTNet service used. The size is limited to 30 bytes.	The Service Name. For example, swift.cte, swift.generic.fa!x, swift.generic.fast!x, and so forth.
Request Type	Specifies the SwInt_RequestType, used to identify the message type of the XML message using the standard message code. The size is limited to 30 bytes.	The Request Type. For example: camt.005.001.02 (GetTransaction)
User Reference	Specifies the SwInt_RequestRef. This is used to associate a request message with subsequent response or error messages. The size is limited to 30 bytes.	A user reference of 30 bytes or less.
Signed?	Specifies whether the request contains Crypto operations to be performed. Only the last Crypto block is analyzed. SwInt_RequestCrypto (digital signature).	Select **True** or **False**. **True** indicates that the request contains Crypto operations to be performed. The configured default is **FALSE**.
Priority	Specifies the SwInt_Priority. The priority of delivery. In the future, SWIFTNet may implement this priority through schemes such as top-queuing, dedicated server processes, network transport priority. In the current implementation, differentiation on priority may be used for store-and-forward delivery.	**Normal** or **Urgent**. The configured default is **Normal**.

Table 8 Connectivity Map - FileAct Client (Continued)

Name	Description	Required Value
Non-Repudiation?	Specifies whether SwInt_NRIndicator, non-repudiation support is being requested.	Select **True** or **False**. True indicates that non-repudiation is requested. The configured default is **FALSE**.
Delivery Notification Queue Name	Specifies the SwInt_NotifQueue, store-and-forward delivery mode. When a value is present, it indicates SnF delivery mode and the queue where SnF delivery notifications are received. In the case of non-delivery (Rejected or Failed message), an SnF failed notification is always generated in this notification queue. In the case of delivery (Accepted or Duplicated message) an SnF delivery notification is optionally generated in this queue.	For store-and-forward delivery mode, enter the name of the delivery notification queue. Size is limited to 30 bytes. For example: ptsauszz_generic!x. (see Sw:DeliveryNotif)
Ask Positive Delivery Notification?	Specifies Sw_DeliveryNotif, delivery notification. This is for store-and-forward delivery mode only. Indicates whether a delivery notification is required in case of successful delivery (Accepted or Duplicated).	Select **True** or **False**. **True** indicates that a delivery notification is required. The configured default is **FALSE**.

Table 8 Connectivity Map - FileAct Client (Continued)

Name	Description	Required Value
Remote File Handler TransferEP	Specifies the Sw_TransferEP, the Transfer EndPoint name used for the remote file handler. Value options: ▪ Blank value: indicates that the file transfer does not use the remote file handler. ▪ Transfer EndPoint name: When this value is present, the file transfer uses the remote file handler. Before the file transfer starts you need to start the remote file handler process with the specified Transfer EndPoint name on your system. The command to start the remote file handler (swfa_handler) requires the command-line arguments: swfa_handler <HostName>:<PortNumber>[:ssl] <TransferEndpoint> [<Process ID>] Here are some examples: swfa_handler snlhost:48003:ssl MyUniqueEndpoint 23450 swfa_handler snlhost:48003 MyUniqueEndpoint 23450 swfa_handler snlhost:48003 MyUniqueEndpoint As for the syntax details and operational guidelines of remote file handler, refer to the SWIFTNet Service Design Guide or consult your system person.	The Transfer EndPoint name used for the remote file handler. Size is limited to 30 bytes.
Block File Transfer?	Specifies BlockFileTransfer. This indicates whether the function call will finish when a final file status is returned. This may be useful for the large file transfers. The final status values for file transfer are: **Completed**, **Duplicated**, **Aborted**, **Failed**, **Rejected**, and **Unknown**.	Select **True** or **False**. **True** indicates that the function call will finish when a final file status is returned. The configured default is **True**.

3.3.7 FileAct Client > Store and Forward

This section maps to parameter **Sw:AcquireSnFRequest** of Primitive **Sw:ExchangeSnFRequest**. For parameters SwXXX under this section, refer to the SNL specification for more detailed descriptions. The InterAct Client > Store and Forward section of the SWIFT AG eWay Connectivity Map properties contains the top-level parameters displayed in Table 7.

Table 9 Connectivity Map - FileAct Client > Store and Forward

Name	Description	Required Value
Queue Name To Acquire	Specifies the SwInt_Queue. For example, ptsauszz_generic!x. Size is limited to 30 bytes.	The User SwInt_Queue For example: ptsauszz_generic!x
Force Acquire?	Specifies the Sw_ForceAcquire. Indicates whether an acquisition request must be accepted if a queue is already acquired.	Select **TRUE** or **FALSE**. True indicates that an acquisition request must be accepted. The configured default is **TRUE**.
Session Mode	Specifies the usage mode of the queue. This property is purposely disabled.	The default setting is **Pull**. Pull is the correct setting for *client* mode ***Note:*** This property is purposely disabled.
Order By	Specifies the Sw_OrderBy. Indicates the order priority in which the messages are retrieved on the queue. The options are: ▪ InterAct ▪ Blank: indicates FIFO (first in, first out). ▪ FileAct ▪ Urgent	Select **InterAct**, ***blank***, **FileAct**, or **Urgent**. The configured default is **FileAct**.
Recovery Mode?	Specifies the Sw_RecoveryMode. Indicates whether the session must be opened in recovery mode. Messages that already have an output sequence number are sent first, before considering the selected order.	Select **TRUE** or **FALSE**. **True** indicates that the session must be opened in recovery mode. The configured default is **FALSE**.

3.3.8 FileAct Client > Get File

This section maps to the parameter **Sw:GetFileRequest** of the Primitive **Sw:ExchangeFileRequest**. For parameters SwXXX under this section, refer to the SNL Specification for more detailed descriptions. The FileAct Client > Get File section of the SWIFT AG eWay Connectivity Map properties contains the top-level parameters displayed in Table 10.

Table 10 Connectivity Map - FileAct Client > Get File

Name	Description	Required Value
Transfer Description	Specifies the Sw_TransferDescription, user information about the file transfer. Free text. Size is limited to 256 bytes.	The transfer description.
Transfer Info	Specifies the Sw_TransferInfo, user information about the file transfer. Structured data that can be analyzed by the server. Size is limited to 256 bytes.	The transfer information.
Logical File Name	Specifies the Sw_LogicalName, the logical name of the file to get. This name is communicated to the server application. By default, this name is the physical name without path. Size is limited to 254 bytes.	The logical file name.
Physical File Name	Specifies the Sw_PhysicalName, the full physical name where the file to get must be stored. If the file already exists it is overwritten. Size is limited to 254 bytes.	The physical file name.
Maximum File Size	Specifies the Sw_MaxSize, the maximum accepted file size. When a value is present, the maximum size is a field that is communicated from the client application to the server application. This property is optional. Size range is 0 to 250 Megabytes.	The maximum file size.

3.3.9 FileAct Client > Put File

This section maps to parameter **Sw:PutFileRequest** of the Primitive **Sw:ExchangeFileRequest**. For parameters SwXXX under this section, refer to the SNL Specification for more detailed descriptions. The **FileAct Client > Put File** section of the SWIFT AG eWay Connectivity Map properties contains the top-level parameters displayed in Table 10.

Table 11 Connectivity Map - FileAct Client > Put File

Name	Description	Required Value
Transfer Description	Specifies the Sw_TransferDescription, This is information about the file transfer. Free text. Size is limited to 256 bytes.	The transfer description.
Transfer Info	Specifies the Sw_TransferInfo, information about the file transfer. This is structured data that can be analysed by the server. Size is limited to 256 bytes.	The transfer information.
Logical File Name	Specifies the Sw_LogicalName, the logical name of the file to put. This name is communicated to the server application. By default, this name is the physical name without path. Size is limited to 254 bytes.	The logical file name.
Physical File Name	Specifies the Sw_PhysicalName, the full physical name of the file to put. Size is limited to 254 bytes.	The physical file name.
File Description	Specifies the Sw_FileDescription. This is user information about the file. Free text. Size is limited to 256 bytes.	The file description.
File Info	Specifies the Sw_FileInfo, user information about the file. This is structured data that can be analysed by the server. Size is limited to 256 bytes.	The file information.

3.3.10 FileAct Client > SnF Fetch File

This section maps to Primitive **Sw:FetchFileRequest**. For parameters SwXXX under this section, refer to the SNL Specification for more detailed descriptions. The FileAct Client > SnF Fetch File section of the SWIFT AG eWay Connectivity Map properties contains the top-level parameters displayed in Table 10.

Table 12 Connectivity Map - FileAct Client > SnF Fetch File

Name	Description	Required Value
Physical File Name	Specifies the Sw_PhysicalName, the full physical name where the file is stored locally. Size is limited to 254 bytes.	The physical file name.

3.3.11 Connection Establishment

The **Connection Establishment** section of the SWIFT AG eWay Connectivity Map properties defines configuration parameters used to control the connection establishment. It contains the top-level parameters displayed in Table 13.

Table 13 Connectivity Map - Connection Establishment

Name	Description	Required Value
Always Create New Connection	Specifies whether to ALWAYS try to create a new connection for a connection establishment request. The options are: ▪ **True**: a new connection is always created without trying to match an existing connection. ▪ **False**: an attempt to match an existing connection (managed by container) is made.	Select **True** or **False**. The configured default is **FALSE**.
Auto Disconnect Connection	Specifies whether the eWay closes the connection automatically after the work is finished on the connection. ▪ **True**: the connection is not re-used. ▪ **False**: the connection is returned to the pool for reuse.	Select **True** or **False**. The configured default is **FALSE**.

3.4 SWIFT AG eWay Environment Properties

The SWIFTAlliance Gateway eWay Environment properties are organized into the following sections:

- **Transport** on page 40
- **Connection Pool Settings** on page 42

3.4.1 Transport

The **Transport** section of the **SWIFT AG eWay Environment properties** contains the top-level properties displayed in Table 14.

Table 14 Environment - Transport

Name	Description	Required Value
Read From RA CFG File	Specifies the manner in which you provide the transport information. You can get RA transport information in two ways: 1 Enter the RA (resource adapter) configuration file name to read all transport information from an existing RA configuration file for your SAG RA environment. 2 Get them one by one from the eWay configuration parameters defined in the rest of this section. If this parameter is specified (not blank), it indicates that you are choosing the first option, and the RA configuration file name is expected for this parameter (for example, sagta_ra.cfg). The other parameters in this section (Host Name, Port Number, etc.) will be ignored. If this parameter is not specified (blank), it indicates that you are choosing the second option, the other parameters in this section (Host Name, Port Number, etc.) must be specified to provide the required transport information.	One of the Following: - Leave the value empty (blank) to use the transport information specified in the rest of this section (Host Name, Port Number, etc.). - Enter the RA configuration file name. All transport information is taken from the existing RA configuration file for your SAG RA environment.
Host Name	Specifies the name or IP address of the host to which you are connecting.	The host name or IP address.

Table 14 Environment - Transport (Continued)

Name	Description	Required Value
Port Number	Specifies the port number of the SAG host to which the RA connects.	The port number of the SAG host to which the RA connects. The configured default is 48002.
Ftla Port Number	Specifies the Ftla port number, the number of the port on the SAG host through which File Transfers will take place.	The Ftla port number. The configured default is 48003.
Server DN	Specifies the Server DN, Distinguished Name used for SWIFTAlliance Gateway authentication.	The Server DN.
CA Certificate	Specifies the file that contains the Certification Authority (CA) certificate.	The CA Certificate.
SSL Mode	Specifies whether the current connection is using data encryption (SSL).	**True** or **False** depending upon whether data encryption is used. **True** indicates that encryption is used, The configured default is **True**.

3.4.2 Connection Pool Settings

The Connection Pool Settings section is specific for the RA connection pool of Sun Java System Application Server or Sun SeeBeyond Integration Server only. Please refer to the corresponding documentations along with your product for more details.

The **Connection Pool Settings** section of the **SWIFT AG eWay Environment properties** contains the top-level properties displayed in Table 15.

Table 15 Environment - Connection Pool Settings

Name	Description	Required Value
Steady Pool Size	Specifies the steady pool size. The steady pool size represents the minimum number of RA connections to be maintained. When it is set to greater than 0, the container not only pre-populates the RA connection pool with the specified number, but also attempts to ensure that there is always this many RA connections in the free pool. This ensures that there are enough RA connections in the ready to serve state to process user requests. This parameter does not necessarily guarantee that no more than steady-pool-size instances exist at a given time. It only governs the number of instances that are pooled over a long period of time. For example, suppose an idle stateless session container has a fully-populated pool with a steady-pool-size of 10. If 20 concurrent requests arrive for the RA connection component, the container creates 10 additional instances to satisfy the burst of requests. The advantage of this is that it prevents the container from blocking any of the incoming requests. However, if the activity dies down to 10 or fewer concurrent requests, the additional 10 instances are discarded.	An integer indicating the steady pool size. The configured default is **1**.
Max Pool Size	Specifies the maximum pool size. This number represents the maximum number of RA connections in the pool. A value of **0** indicates that the pool is unbounded.	An integer indicating the maximum pool size. A value of **0** indicates that the pool is unbounded. The configured default is **32**.

Table 15 Environment - Connection Pool Settings (Continued)

Name	Description	Required Value
Max Wait Time in Millis	Specifies the maximum wait time in milliseconds. If an RA connection is not available, the caller must wait this long before another RA connection is created. A value of **0** indicates that an exception is thrown if there is no RA connection available. If the pool is completely utilized and the timer expires, an exception will be delivered to the application. ***Note:*** This element is deprecated for the bean pool container for Sun Java System Application Server.	An integer indicating the maximum wait time in milliseconds. The configured default is **60000**.
Pool Idle Timeout In Seconds	Specifies the pool idle timeout in seconds. This serves as a hint to the server. A timer thread periodically removes unused RA connections. This parameter defines the interval at which this thread runs. This thread removes unused RA connection that have an expired timeout. This allows you to specify the amount of time that an RA connection instance can be idle in the pool. When pool-idle-timeout-in-seconds is set to greater than 0, the container removes or destroys any RA connection instance that is idle for this specified duration. It is the maximum time that a component can remain idle in the pool. After this amount of time, the pool can remove this bean. A value of 0 specifies that idle RA connections can remain in the pool indefinitely.	An integer indicating the pool idle timeout in seconds. A value of 0 indicates that an idle RA connection may remain in the pool indefinitely. When the value is greater than 0, the container removes or destroys any RA connection instance that is idle for this specified duration. The configured default is 300

Chapter 4

OTD Overview

This chapter provides an overview of the SWIFT AG eWay OTD.

What's in This Chapter

4.1 Introduction to SWIFT AG eWay OTD

The SWIFT AG eWay includes the SAGOutboundeWay Object Type Definition.

The **SAGOutboundeWay** OTD structure is organized into five sections: Configuration, Constants, Primitives, RemoteApis (Remote APIs), and Services (see Figure 6).

Figure 6 SAGOutboundeWay OTD

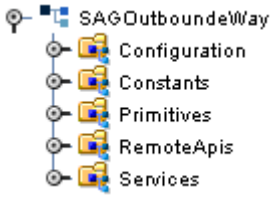


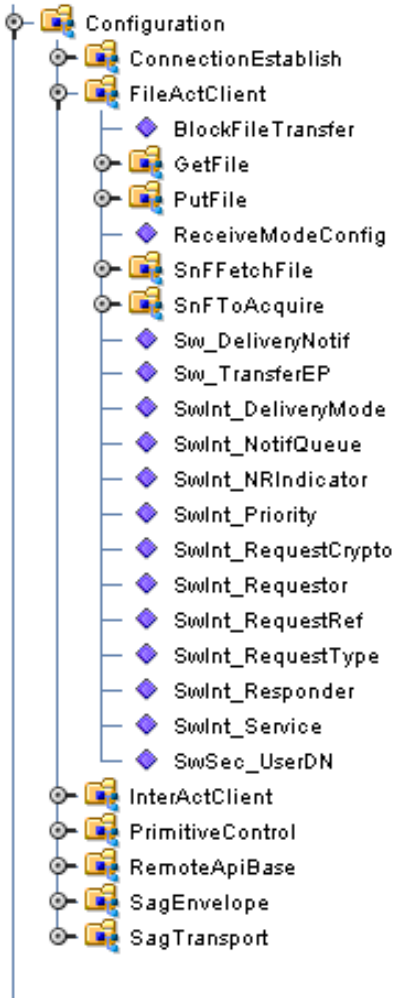
Figure 6 shows the **SAGOutboundeWay** OTD as displayed in the Collaboration Editor.

Configuration Node

The Configuration node directly corresponds to the eWay Connectivity Map and Environment Configuration properties. The OTD Configuration node offers dynamic configuration (configuration on the fly). Dynamic configuration allows you to edit the configuration, based on your Collaboration's Business Rule logic, from the Java Collaboration Editor, dynamically changing a parameter without shutting down your Project.

As displayed in Figure 7, the Configuration section of the OTD is a Java representation of the SWIFT AG eWay Configuration file. For more information regarding any of these configuration parameters, refer to the corresponding property in Properties Chapter, **"Configuring the eWay" on page 21**. displays The Configuration section with the expanded FileActClient node and sub-nodes is displayed in **Figure 7 on page 45**.

Figure 7 SAGOutboundeWay OTD - Configuration Node

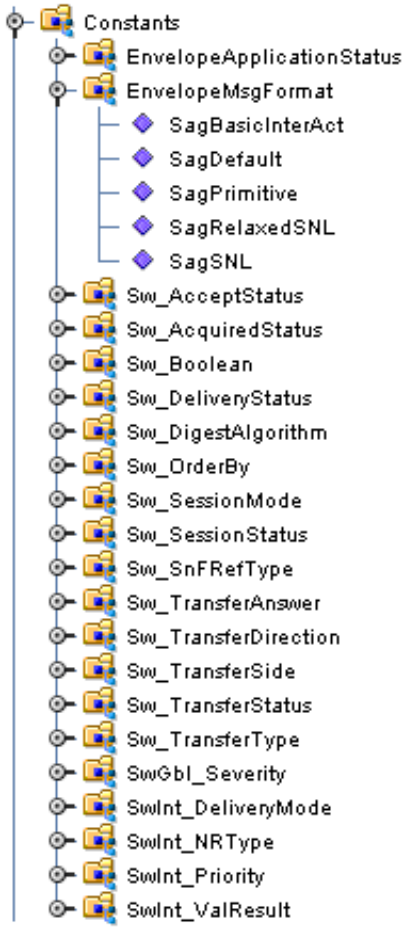


Constants Node

The Constants node provides a convenient way to select SNL related constants. Constants are literal values that have a name (see **Figure 8 on page 46**).

OTD Constants are presented in the Collaboration Editor so that you can simply drag and drop the Constant to a Business Rule, avoiding possible case or spelling errors.

Figure 8 SAGOutboundeWay OTD - Constants Node

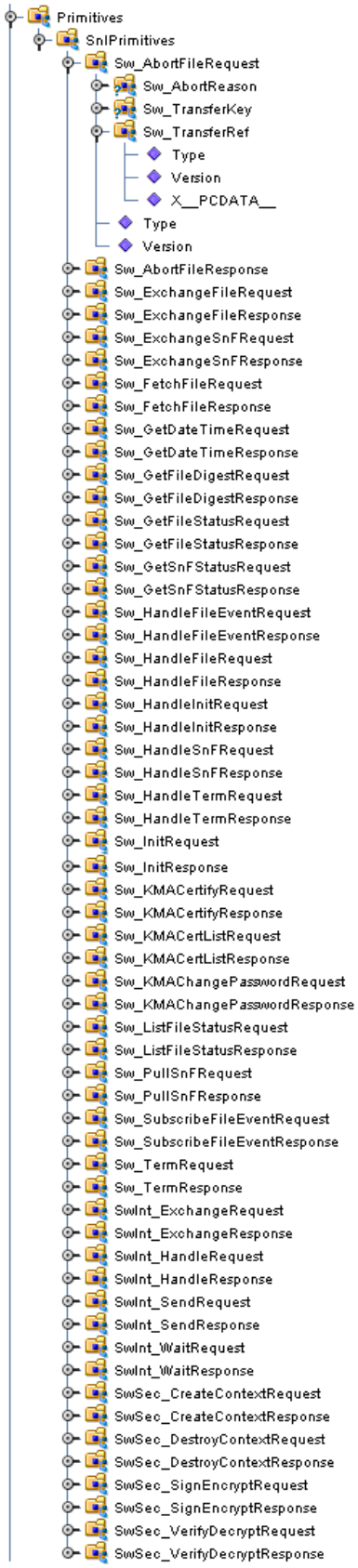


Primitives Node

The Primitives node provides the full set of SNL Primitives as defined by the SNL specification. For information regarding any of the SNL Primitives, refer to the SWIFTAlliance Gateway Documentation. The SNL Primitives node and sub-nodes are displayed in **Figure 9 on page 47**.

Advanced users can construct their own Primitives and send the Primitive using the SWIFT AG eWay API, directly communicating with SWIFTNet. Once they get a response to their request, they can parse the response based on their Primitives. The parser is provide in the OTDs Primitives section. The response can be dragged to the appropriate node to parse the response.

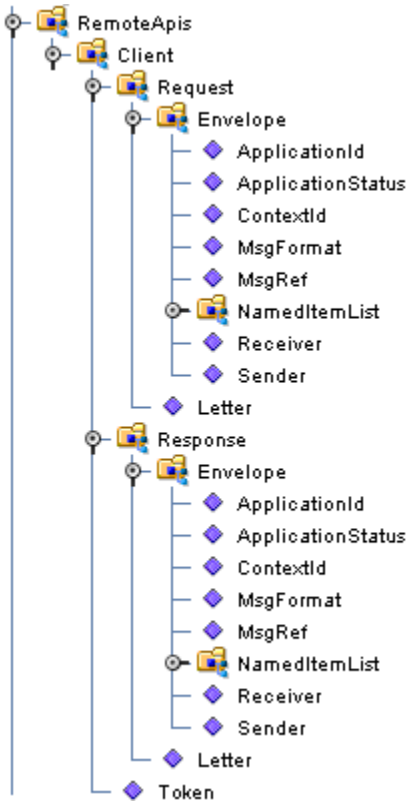
Figure 9 SAGOutboundeWay OTD - Primitives Node



Remote APIs Node

The SAGOutboundeWay OTD's RemoteApis node exposes the SWIFT Remote API's client APIs. Just as the Primitives section provide a "message structure", the RemoteApis section provides a "communication function structure". The Remote APIs allow you to perform special lower level communication functions.

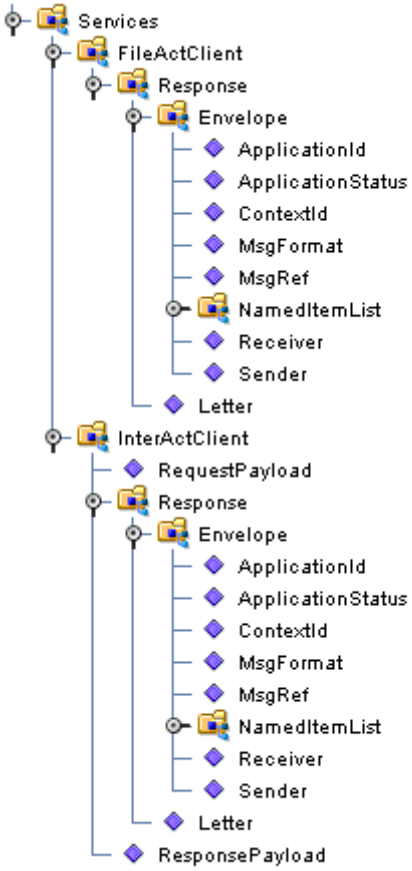
Figure 10 SAGOutboundeWay OTD - Remote APIs Node



Service Node

The Service section of the OTD allows you to perform higher level message and communication functions. Right-click the **FileActClient** or **InterActClient** node in the Collaboration to view the available methods to perform your business functions (exchange message, get file, put file, queue access, and so forth).

Figure 11 SAGOutboundeWay OTD - Services Node



See the sample Projects for an example of how this OTD is used to create your business logic. The prjSAGCert Project demonstrates several business functions with one Collaboration. For directions on importing the prjSAGCert sample Project, see **"Importing a Sample Project" on page 52**.

Chapter 5

Implementing a Project Using Java Collaboration Definitions (JCD)

This chapter provides an introduction to the SWIFTAlliance Gateway eWay's components and demonstrates on how these components are created and implemented in an eGate Project. It is assumed that the reader understands the basics of to create a Project using the Enterprise Designer. For more information on creating an eGate Project see the *Sun SeeBeyond eGate™ Tutorial* and the *Sun SeeBeyond eGate™ Integrator User's Guide*.

What's in This Chapter

5.1 SWIFT AG eWay Components

This chapter presents a sample SWIFTAlliance Gateway eWay Project created using the same procedures as the sample end-to-end Project provided in the *eGate Integrator Tutorial*. The components of the SWIFT AG eWay include the following:

SAGOutboundeWay OTD

The **SAGOutboundeWay OTD** is a natural point of connection between the eWay and SWIFTNet. Within a Java Collaboration Definition, this OTD defines the methods to connect and utilize the SWIFTAlliance Gateway (see **"OTD Overview" on page 44**).

SWIFT AG eWay Properties File

The properties configuration file for the SWIFT AG eWay directly corresponds to the properties of the SAGOutboundeWay OTD. The Properties Editor allows you to preset the parameters for your system to connect and communicate with SWIFTAlliance Gateway (see **"Configuring the eWay" on page 21**).

5.2 The SWIFT AG eWay Sample Projects

The SWIFT AG eWay includes three sample Projects that use Java Collaboration Definitions for their Business.

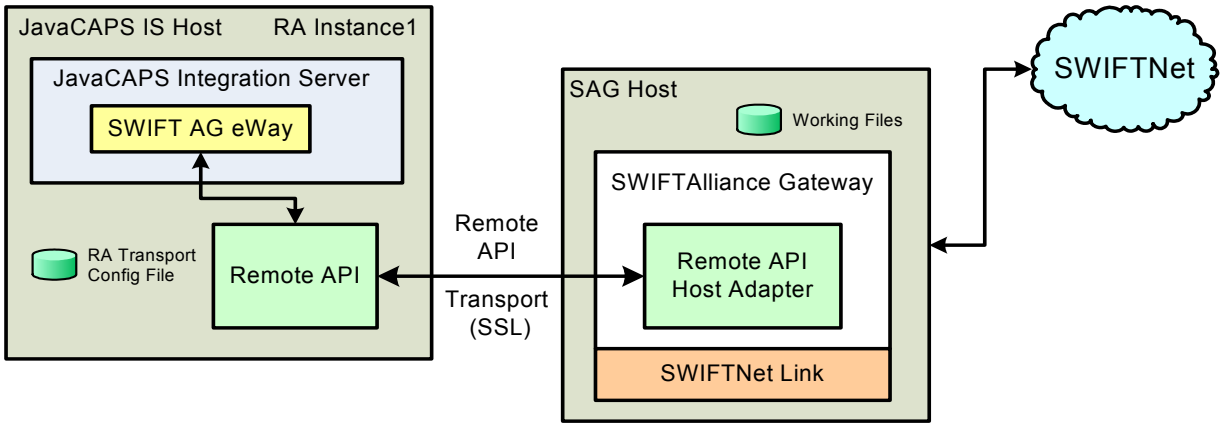
- **prjSagFA**: demonstrates support for the SWIFT FileAct service.
- **prjSagIA**: demonstrates support for the SWIFT InterAct service.
- **prjSAGCert**: provided as an example of a SWIFT AG eWay Project which demonstrates several important scenarios.

The **prjSagFA** and **prjSagIA** sample Projects can be imported in a near-complete state, and can be run with minimal configuration.

The sample Projects are designed to run using a single Remote API instance connecting with SWIFTAlliance Gateway (see Figure 12).

Important: *All projects created in previous versions, must have the Connectivity Map configuration and Environment configuration files opened before building the projects in 5.1.3.*

Figure 12 Running the Sample Projects



prjSagFA Sample Overview

The **prjSagFA** Project demonstrates the following:

- The inbound File eWay subscribes to an external input directory. When a target message is present, the File eWay picks up the message and triggers the jcdSagFA Java Collaboration.
- The jcdSagFA Collaboration connects to the SWIFT AG server through the SWIFT AG eWay. The Collaboration uses the FileAct service to get a file, then writes several relevant Primitives to an output file.
- The outbound File eWay publishes the new message to an output directory.

prjSagIA Sample Overview

The **prjSagIA** Project demonstrates the following:

- The inbound File eWay subscribes to an external input directory. When a target message is present, the File eWay picks up the message and triggers the jcdSAGIA Java Collaboration.
- The jcdSAGIA Collaboration connects to the SWIFT AG server through the SWIFT AG eWay. The Collaboration uses the InterAct service to exchange a message, then writes several relevant Primitives to an output file.
- The outbound File eWay publishes the new message to an output directory.

prjSAGCert Project

The **prjSAGCert** Project is presented as an example of a SWIFT AG eWay Project that demonstrates several important scenarios with one Collaboration. An outline of these scenarios is available at **jcdSAGCert Collaboration Definition Scenarios** on page 84, as well as the Collaboration's Java code.

5.3 Importing a Sample Project

To import a sample Project to the Enterprise Designer do the following:

1. The sample files are uploaded with the eWay's documentation SAR file and downloaded from the Sun Java Composite Application Platform Suite Installer's **Documentation** tab. Save and extract the SWIFT AG eWay sample Project package, **SWIFT_AG_Sample.zip,** to a local directory. The SWIFT_AG_Sample.zip file contains the following files:
   - Sample Projects:
     - prjSagFA.zip
     - prjSagIA.zip
     - prjSAGCert.zip
   - Sample input and output files:
     - input_SAG_FA_.txt~in
     - input_SAG_IA_.txt~in
     - outputSagFA1.dat
     - outputSAGIA1.dat
   - Working and primitive files for the sample Projects:
     - SAG_Cert.zip. These files must be extracted to the Swift Remote API machine or the SWIFT AG server (see **"Extracting the Working Files" on page 53**)

   Save and extract the SWIFT_AG_Sample.zip to a local directory.

2 Save any unsaved work before importing a Project.

3 From the Enterprise Designer's Project Explorer pane, right-click the Repository and select **Import** from the shortcut menu. The **Import Manager** appears.

4 Browse to the directory that contains the sample Project zip file. Select the sample file (for this sample, **SWIFT_AG_Sample_JCD.zip**) and click **Import**. After the sample Project is successfully imported, click **Close**.

5 Before an imported sample Project can be run you must do the following:

- Create an **Environment** (see **"Creating an Environment" on page 66**)
- Configure the eWays for your specific system (see **"Configuring the eWays" on page 66**)
- Create a **Deployment Profile** (see **"Creating the Deployment Profile" on page 69**)
- Initialize your Remote API (see **"Initializing your Remote API" on page 70**)
- Create and start a domain (see **"Creating and Starting the Domain" on page 70**)
- Build and deploy the Project (see **"Building and Deploying the Project" on page 71**)

5.4 Extracting the Working Files

The SWIFT AG eWay Sample Project package includes a ZIP file, **SAG_Cert.zip**, that contains working files for the sample Projects. These files are extracted to the **SWIFTAlliance Gateway server host** prior to running the sample Project. You can run the Project with the Working files loaded on the Remote API host, but this requires that you also run the **SWIFT Remote File Handler.**

Extracting the files to the SWIFT AG Server

Copy the working files to the **SWIFT AG** server. To do this, extract the files to the SAG server host in the following location:

```
/temp/SAG_Cert folder
```

Extracting the files to the Remote API computer

If you choose to run the sample Projects with the working files located on the **SWIFT Remote API computer**, where your Logical Host is located, do the following:

1 Extract the working files to the following location:

```
/temp/SAG_Cert folder
```

2 Start the **SWIFT Remote File Handler**. If you are running the Logical Host on the **SWIFT Remote API computer**, configure the sample Projects to use the Remote File Handler (specify the Transfer Endpoint) and start **swfa_handler.exe** before running the Projects. For directions on creating an Endpoint or running the Remote File

Handler Process, refer to the *SWIFTNet Service Design Guide* available as part of the *SWIFTAlliance Gateway Developers Toolkit*.

5.5 Creating the prjSagFA Project

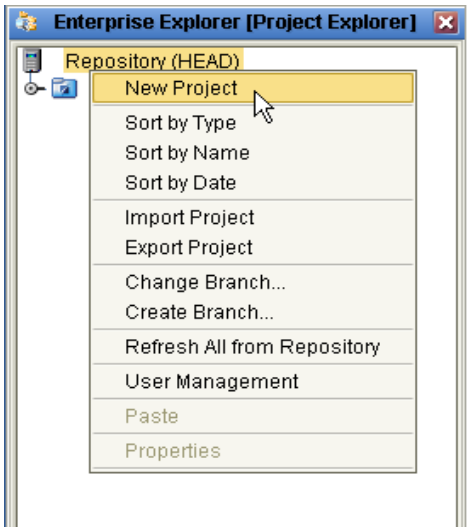
The following pages provide step by step directions for manually creating the prjSagFA Project's components.

5.5.1 Creating a Project

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

1 From the Enterprise Designer's Project Explorer tree, right-click the Repository and select **New Project** (see Figure 13). A new Project (**Project1**) appears on the Project Explorer tree.

Figure 13 Enterprise Explorer - New Project



2 Rename the Project to **prjSagFA**.

5.5.2 Creating the Collaboration Definition

The next step in the sample is to create the **jcdSagFA** Java Collaboration using the Collaboration Definition Wizard (Java). Once the Collaboration has been created, the Collaboration's Business Rules can be written using the Collaboration Editor (Java).

Creating the jcdSagFA Collaboration Definition

The **jcdSagFA** Collaboration defines transactions from the inbound File eWay to the SWIFT AG eWay and the outbound File eWay.

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

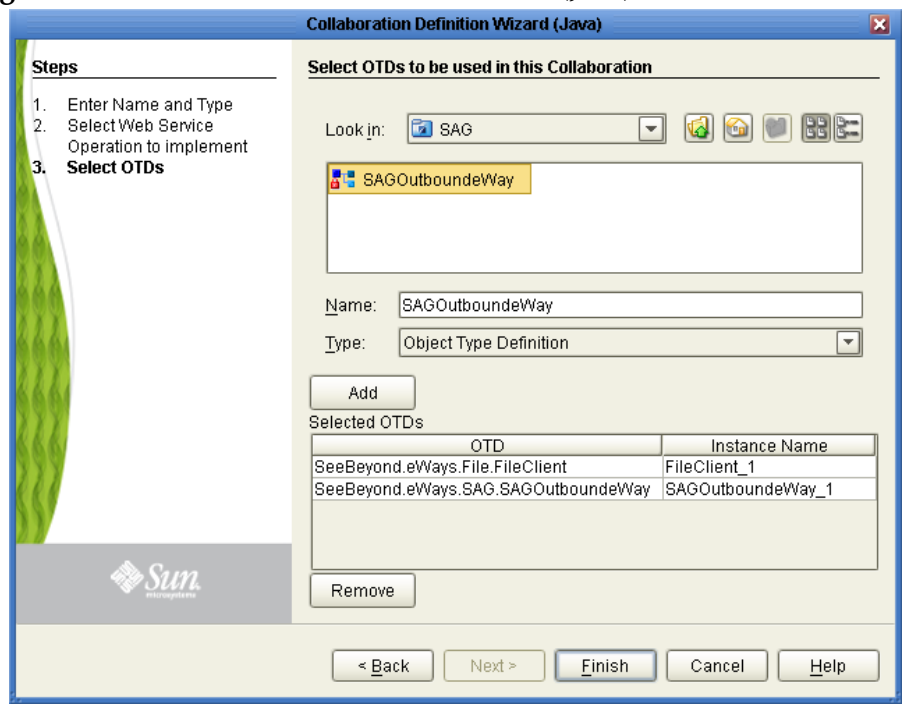
2 Enter a Collaboration Definition name (for this sample **jcdSagFA**) and click **Next**.

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

4 For Step 3 of the wizard, from the Select OTDs selection window, double-click **Sun SeeBeyond** > **eWays** > **File** > **FileClient**. The **FileClient_1** OTD is added to the Selected OTDs field.

5 Click **Up One Level** to return to the Repository. From the Select OTDs selection window, double-click **Sun SeeBeyond** > **eWays** > **SAG** > **SAGOutboundeWay**. The **SAGOutboundeWay_1** OTD is added to the Selected OTDs field (see Figure 14).

Figure 14 Collaboration Definition Wizard (Java) - Select Web Service



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

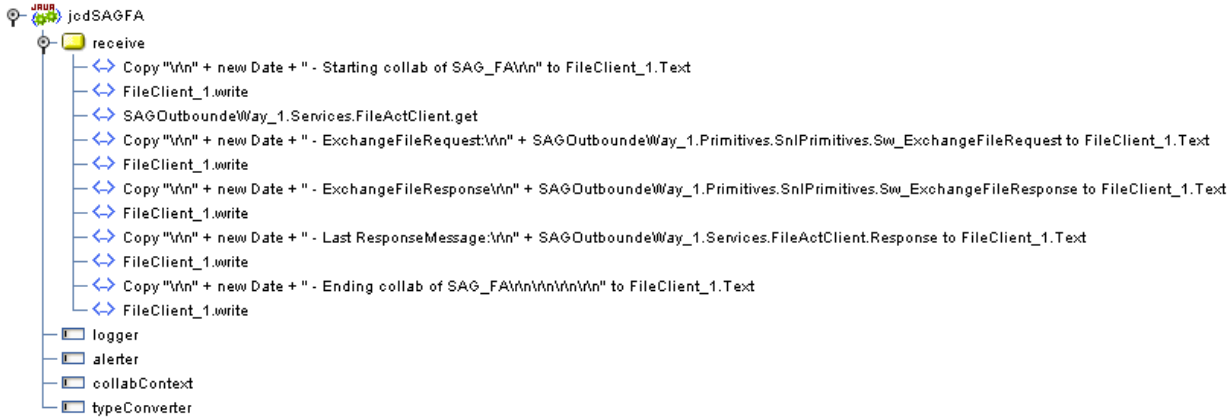
5.5.3 Using the Collaboration Editor (Java)

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

Creating the jcdSagFA Business Rules

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

Figure 15 jcdSagFA Collaboration Business Rules



The Java Collaboration Editor features a graphical interface, the **Business Rules Designer**, that allows you to select menu options and use drag-and-drop to graphically create your Business Rules. You can also create your Business Rules using the editors **Java Source Editor**, which allows you to simply enter you Business Rules in Java. To create the **jcdSagFA** Collaboration Business Rules using the Java Collaboration Editor's Business Rules Designer, do the following:

1. To open Collaboration Editor to the Sun **jcdSagFA** Collaboration, double-click **jcdSagFA** in the Project Explorer tree.
2. If you choose to create comments for the Business Rules, click the comment icon on the Business Rules toolbar. The **Enter a Comment** dialog box appears. Enter the comment and click **OK**. The comment is placed on the Business Rules tree under the last selected item. You can move (drag) the new comment up or down the Business Rules tree.
3. The **Copy "\r\n" + new Date + " - Starting collab of SAG_FA\r\n" to FileClient_1.Text** rule creates a "Starting collab of SAG_FA" comment and includes the starting time stamp.

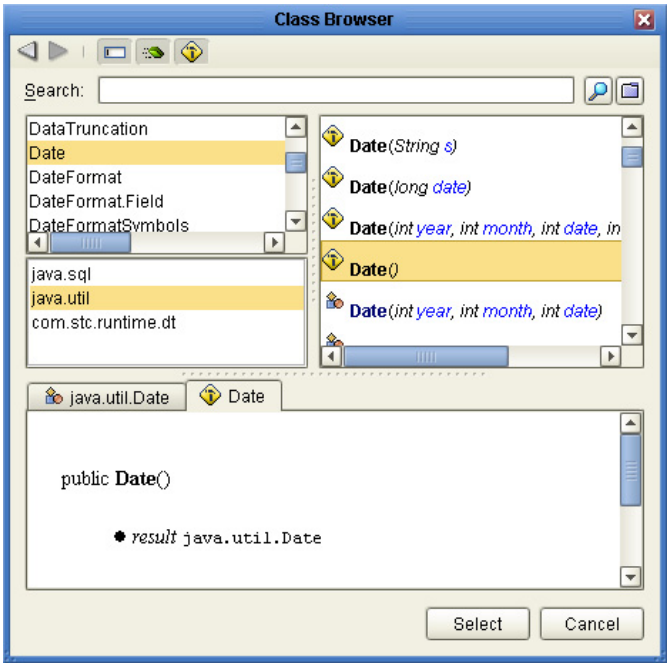
   Create the **Copy "\r\n" + new Date + " - Starting collab of SAG_FA\r\n" to FileClient_1.Text** rule:

   A. From the Business Rules toolbar, click the **rule** icon to add a new rule under **receive** on the Business Rules tree.

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

C From the Class Browser, select **Date** as the class, **java.util** as the package, and **Date()** as the constructor (see Figure 16). Click **Select**. The **Date** constructor box is added to the Business Rules Designer mapping canvas.

Figure 16 jcdSagFA Collaboration Business Rules - Class Browser



D From the Business Rules Designer's **Math** menu, select **Add**. The **Add** box appears.

E From the Business Rules Designer's **String** menu, select **Literal String**. The **String** Literal box appears. Enter **\r\n** as the Literal value.

F Map the **"\r\n"** output node of the **String** Literal box to the **Value1** input node of the **Add** box.

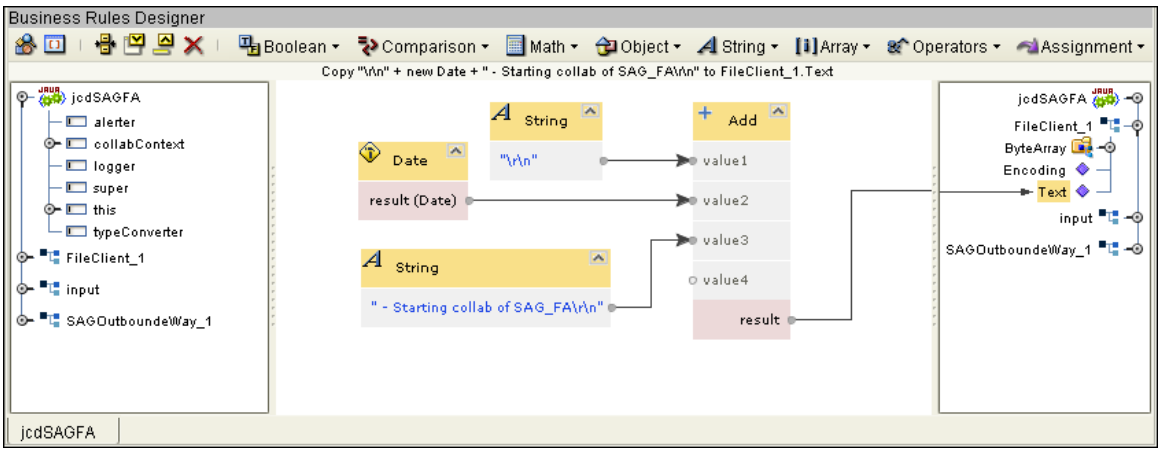
G Map the **result (Date)** output node of the **Date** constructor box to the **value2** input node of the **Add** box. To do this, click on the **result (Date)** output node of the **Date** constructor box, and drag your cursor to the input node of the **Add** box. A link now connects the two nodes.

H From the Business Rules Designer's **String** menu, select **Literal String**. The **String** Literal box appears. Enter **- Starting collab of SAG_FA\r\n** as the Literal value.

I Map the **" - Starting collab of SAG_FA\r\n"** output node of the second **String** Literal box to the **Value3** input node of the **Add** box.

J Map the **result** output node of the **Add** box, to Text under FileClient_1 in the right pane of the Business Rules Designer (see **Figure 17 on page 58**).

Figure 17 jcdSagFA Collaboration Business Rules - Create Variable



4 The **FileClient_1.write** rule writes data value to the file.

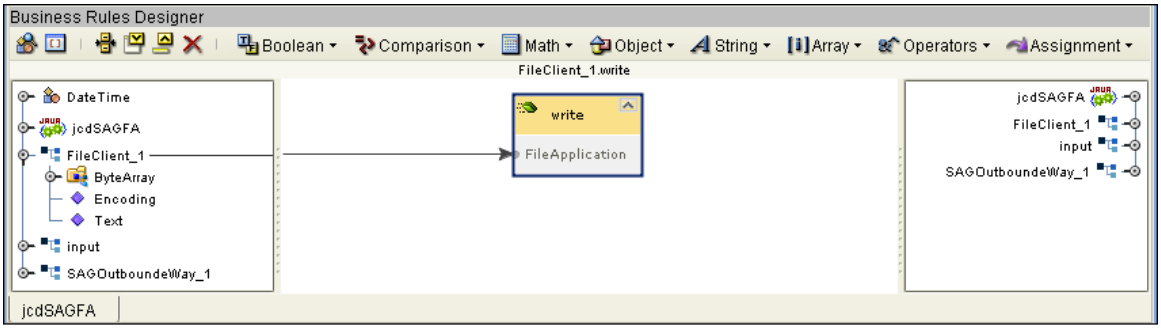
Create the **FileClient_1.write** rule:

A From the Business Rules toolbar, click the **rule** icon to add a new rule.

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

C Select **write()** from the method selection window. The **write** method box appears (see Figure 18).

Figure 18 Java Collaboration Editor - jcdSagFA Business Rules



5 The **SAGOutboundeWay_1.Services.FileActClient.get** rule connects to the FileAct Client service.

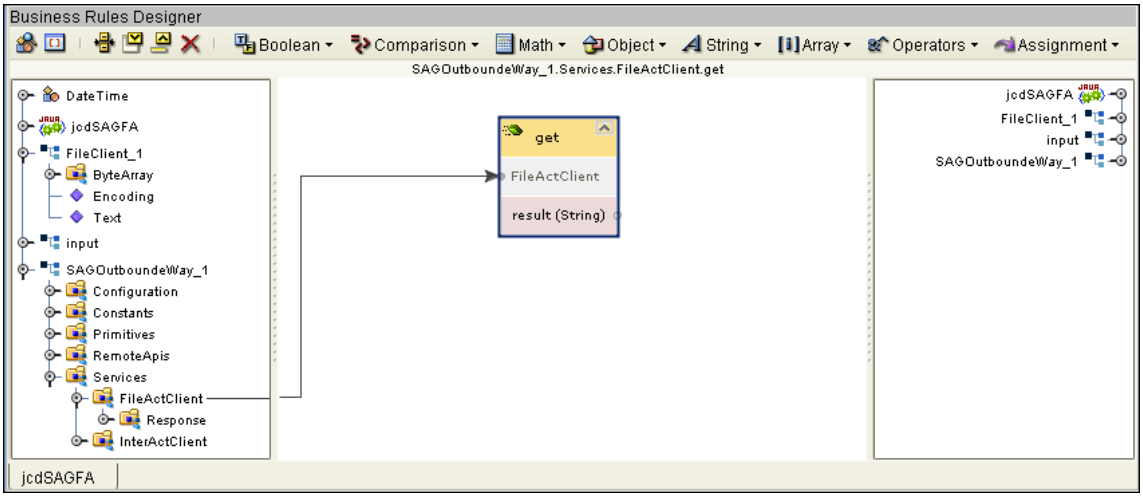
Create the **SAGOutboundeWay_1.Services.FileActClient.get** rule:

A From the Business Rules toolbar, click the **rule** icon to add a new rule.

B Right-click **FileActClient** under **SAGOutboundeWay_1 > Services** in the left pane of the Business Rules Designer, and click **Select method to call** from the shortcut menu.

C Select **get()** from the method selection window. The **get** method box appears (see **Figure 19 on page 59**).

Figure 19 Java Collaboration Editor - jcdSagFA Business Rules



6 The **Copy "\r\n" + new Date + " - ExchangeFileRequest:\r\n" + SAGOutboundeWay_1.Primitives.SnlPrimitives.Sw_ExchangeFileRequest to FileClient_1.Text** rule gets the Primitive value “File Request,” from the SAG server and includes time stamp.

Create the **Copy "\r\n" + new Date + " - ExchangeFileRequest:\r\n" + SAGOutboundeWay_1.Primitives.SnlPrimitives.Sw_ExchangeFileRequest to FileClient_1.Text** rule:

A From the Business Rules toolbar, click the **rule** icon to add a new rule under **receive** on the Business Rules tree.

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

C From the Class Browser, select **Date** as the class, **java.util** as the package, and **Date()** as the constructor. Click **Select**. The **Date** constructor box is added to the Business Rules Designer mapping canvas.

D From the Business Rules Designer’s **Math** menu, select **Add**. The **Add** box appears.

E From the Business Rules Designer’s **String** menu, select **Literal String**. The **String** Literal box appears. Enter **\r\n** as the Literal value.

F Map the **"\r\n"** output node of the **String** Literal box to the **Value1** input node of the **Add** box.

G Map the **result (Date)** output node of the **Date** constructor box to the **value2** input node of the **Add** box.

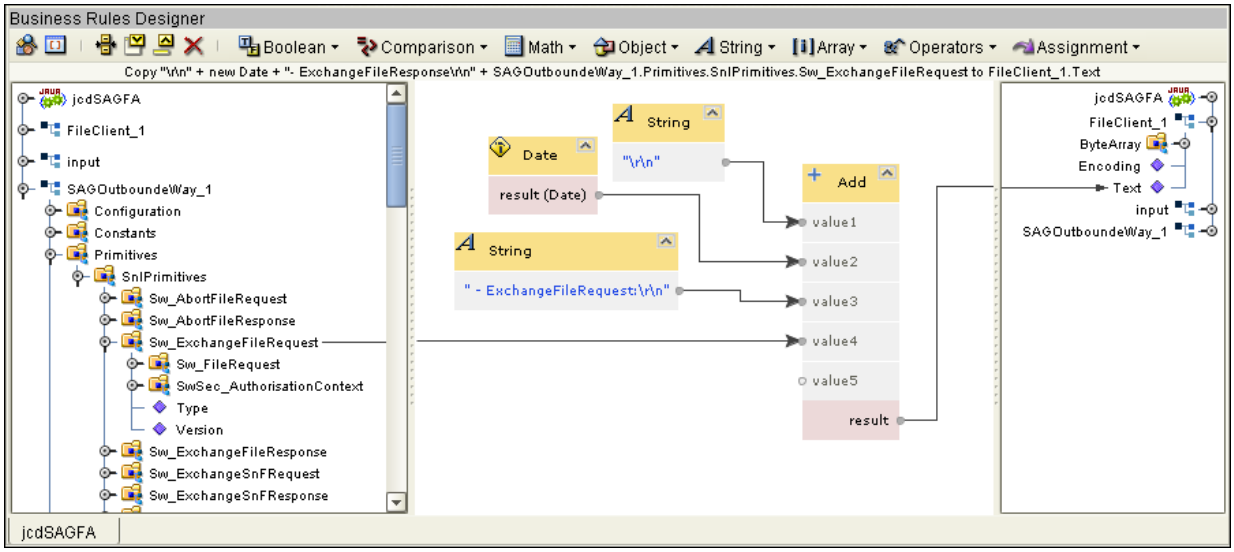
H From the Business Rules Designer’s **String** menu, select **Literal String**. The **String** Literal box appears. Enter **- ExchangeFileRequest\r\n** as the Literal value.

I Map the **" - ExchangeFileRequest\r\n"** output node of the second **String** Literal box to the **Value3** input node of the **Add** box.

J Map **Sw_ExchangeFileRequest** under **Primitives > SnlPrimitives** in the left pane of the Business Rules Designer, to the **value4** input node of the **Add** box.

K Map the **result** output node of the **Add** box, to **Text** under **FileClient_1** in the right pane of the Business Rules Designer (see Figure 20).

Figure 20 Java Collaboration Editor - jcdSagFA Business Rules



7 Create the **FileClient_1.write** rule:

A Follow the directions provided in step 4 of this section to create another **FileClient_1.write** rule.

8 The **Copy "\r\n" + new Date + " - ExchangeFileResponse\r\n" + SAGOutboundeWay_1.Primitives.SnlPrimitives.Sw_ExchangeFileResponse to FileClient_1.Text** rule gets the Primitive value, “File Response,” from the SAG server, and includes the time stamp.

Create the **Copy "\r\n" + new Date + " - ExchangeFileResponse\r\n" + SAGOutboundeWay_1.Primitives.SnlPrimitives.Sw_ExchangeFileResponse to FileClient_1.Text** rule:

A From the Business Rules toolbar, click the **rule** icon to add a new rule.

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

C From the Class Browser, select **Date** as the class, **java.util** as the package, and **Date()** as the constructor. Click **Select**. The Date constructor box is added to the Business Rules Designer mapping canvas.

D From the Business Rules Designer’s **Math** menu, select **Add**. The **Add** box appears.

E From the Business Rules Designer’s **String** menu, select **Literal String**. The **String** Literal box appears. Enter **\r\n** as the Literal value.

F Map the **"\r\n"** output node of the **String** Literal box to the **Value1** input node of the **Add** box.

G Map the **result (Date)** output node of the **Date** constructor box to the **value2** input node of the **Add** box.

H From the Business Rules Designer's **String** menu, select **Literal String**. The **String** Literal box appears. Enter **- ExchangeFileResponse\r\n** as the Literal value.

I Map the **" - ExchangeFileResponse\r\n"** output node of the second **String** Literal box to the **Value3** input node of the **Add** box.

J Map **Sw_ExchangeFileResponse** under **Primitives > SnlPrimitives** in the left pane of the Business Rules Designer, to the **value4** input node of the **Add** box.

K Map the **result** output node of the **Add** box, to **Text** under **FileClient_1** in the right pane of the Business Rules Designer.

9 The **FileClient_1.write** rule writes the data to the message.

Create the **FileClient_1.write** rule:

A Follow the directions provided in step 4 of this section to create another **FileClient_1.write** rule.

10 The **Copy "\r\n" + new Date + " - Last ResponseMessage:\r\n" + SAGOutboundeWay_1.Services.FileActClient.Response to FileClient_1.Text** rule creates the Response Message and includes the starting time and date.

Create the **Copy "\r\n" + new Date + " - Last ResponseMessage:\r\n" + SAGOutboundeWay_1.Services.FileActClient.Response to FileClient_1.Text** rule:

A From the Business Rules toolbar, click the **rule** icon to add a new rule.

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

C From the Class Browser, select **Date** as the class, **java.util** as the package, and **Date()** as the constructor. Click **Select**. The Date constructor box is added to the Business Rules Designer mapping canvas.

D From the Business Rules Designer's **Math** menu, select **Add**. The **Add** box appears.

E From the Business Rules Designer's **String** menu, select **Literal String**. The **String** Literal box appears. Enter **\r\n** as the Literal value.

F Map the **"\r\n"** output node of the **String** Literal box to the **Value1** input node of the **Add** box.

G Map the **result (Date)** output node of the **Date** constructor box to the **value2** input node of the **Add** box.

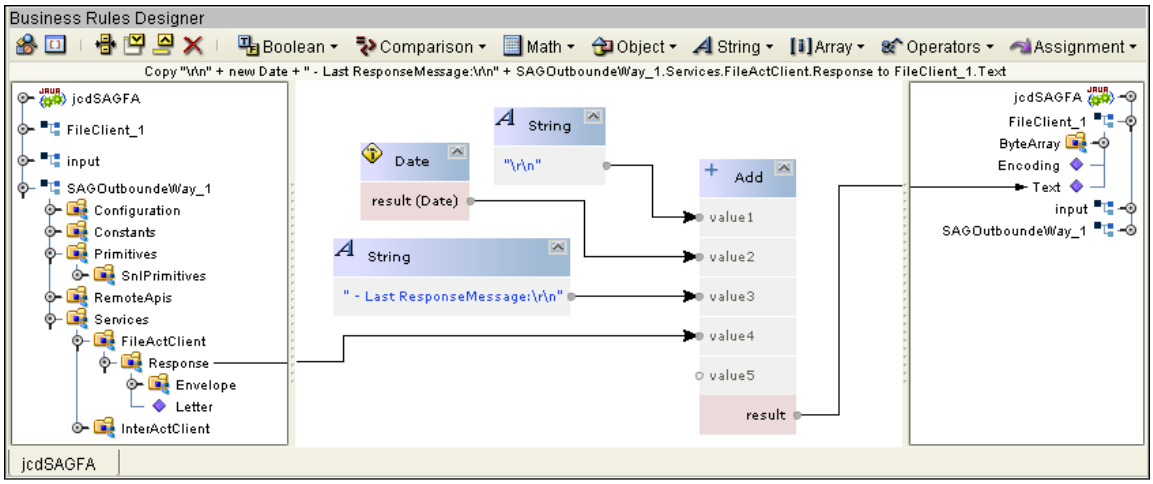
H From the Business Rules Designer's **String** menu, select **Literal String**. The **String** box appears. Enter **- Last ResponseMessage:\r\n** as the Literal value.

I Map the **" - Last ResponseMessage:\r\n"** output node of the **String** Literal box to the **Value3** input node of the **Add** box.

J Map **Response** under **Services > FileActClient** in the left pane of the Business Rules Designer, to the **value4** input node of the **Add** box.

K Map the **result** output node of the **Add** box, to **Text** under **FileClient_1** in the right pane of the Business Rules Designer (see Figure 21).

Figure 21 Java Collaboration Editor - jcdSagFA Business Rules



11 The **FileClient_1.write** rule writes the data to the message.

Create the **FileClient_1.write** rule:

A Follow the directions provided in step 4 of this section to create another **FileClient_1.write** rule.

12 The **Copy "\r\n" + new Date + " - Ending collab of SAG_FA\r\n\r\n\r\n\r\n" to FileClient_1.Text** rule creates the "Ending collab of SAG_FA" comment and includes the starting time and date.

Create the **Copy "\r\n" + new Date + " - Ending collab of SAG_FA\r\n\r\n\r\n\r\n" to FileClient_1.Text** rule:

A From the Business Rules toolbar, click the **rule** icon to add a new rule.

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

C From the Class Browser, select **Date** as the class, **java.util** as the package, and **Date()** as the constructor. Click **Select**. The Date constructor box is added to the Business Rules Designer mapping canvas.

D From the Business Rules Designer's **Math** menu, select **Add**. The **Add** box appears.

E From the Business Rules Designer's **String** menu, select **Literal String**. The **String** Literal box appears. Enter **\r\n** as the Literal value.

F Map the **"\r\n"** output node of the **String** Literal box to the **Value1** input node of the **Add** box.

G Map the **result (Date)** output node of the **Date** constructor box to the **value2** input node of the **Add** box.

H From the Business Rules Designer's **String** menu, select **Literal String**. The **String** Literal box appears. Enter **- Ending collab of SAG_FA\r\n\r\n\r\n\r\n** as the Literal value.

I Map the **" - Ending collab of SAG_FA\r\n\r\n\r\n\r\n"** output node of the **String** Literal box to the **Value3** input node of the **Add** box.

J Map the **result** output node of the **Add** box, to **Text** under **FileClient_1** in the right pane of the Business Rules Designer.

13 The **FileClient_1.write** rule writes the data to the message.

Create the **FileClient_1.write** rule:

A From the Business Rules toolbar, click the **rule** icon to add a new rule.

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

C Select **write()** from the method selection window. The **write** method box appears.

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

15 Save your current changes to the repository.

For more information on how to create Business Rules using the Collaboration Editor see the *Sun SeeBeyond eGate™ Integrator User's Guide*.

5.5.4 Creating a Connectivity Map

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

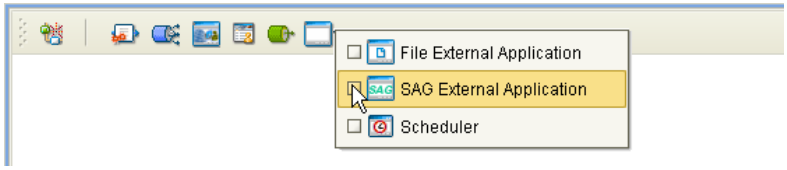
1 From the Project Explorer tree, right-click the new **prjSagFA** Project and select **New > Connectivity Map** from the shortcut menu.

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

Selecting the External Applications

In the Connectivity Map, eWays are associated with External Applications. For example, to establish a connection to an external SWIFTAlliance Gateway application, you must first select SAG External Application as an External Application to use in your Connectivity Map (see Figure 22).

Figure 22 Connectivity Map - External Applications



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

2 Select the External Applications that are necessary to create your Project (for this sample, **SAG External Application** and **File**). Icons representing the selected External Applications are added to the Connectivity Map toolbar.

Populating the Connectivity Map (Manually)

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

Note: *You can also Populate and Bind the Connectivity Map components using the Connectivity Map Generator. See* **"Using the Connectivity Map Generator" on page 65** *for more information.*

To populate your Connectivity Map, do the following

1 From the Project Explorer tree, drag and drop your **jcdSagFA** Collaboration onto the Connectivity Map Editor canvas.

2 From the Connectivity Map Editor toolbar, drag the following components to the Connectivity Map canvas as displayed in Figure 23:

- **File External Application** (2 for this sample).
- **SWIFTAlliance Gateway External Application** (1 for this sample)

Figure 23 Connectivity Map with Components

3 Rename the components on the Connectivity Map as follows:

- **File1** External Application to **eaFileIn**
- **File2** External Application to **eaFileOut**
- **SAG1** External Application to **eaSAG**

4 Save your current changes to the Repository.

5.5.5 Binding the eWay Components

Next, the components are associated and Bindings are created in the Connectivity Map.

1 From the Connectivity Map Editor, double-click the **cmSagFA_jcdSagFA** service. The **cmSagFA_jcdSagFA** binding dialog box appears using the **jcdSagFA** Rule.

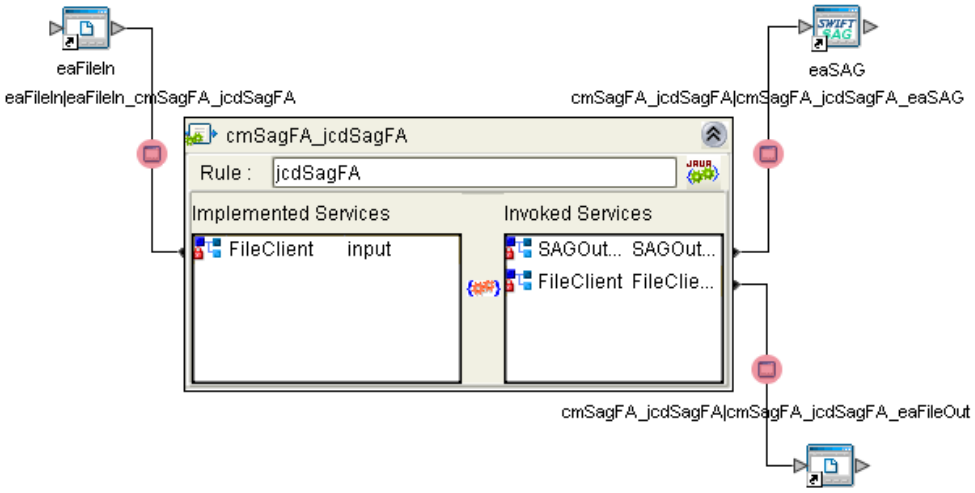
2 From the **cmSagFA_jcdSagFA** binding dialog box, drag **FileClient Input** (under Implemented Services) to the output node of the **eaFileIn** (File) External

Application. A link now appears between the jcdSAGCert binding dialog box and the **eaFileIn** eWay.

3 From the **cmSagFA_jcdSagFA** binding dialog box, drag **FileClient_1** (under Invoked Services) to the input node of the **FileOut** External Application

4 From the **cmSagFA_jcdSagFA** dialog box, drag **SAGOutboundeWay_1** (under Invoked Services) to the input node of the **eaSAG** External Application (see Figure 24).

Figure 24 Connectivity Map - Binding the Project's Components



5 Save your current changes to the Repository.

5.5.6 Using the Connectivity Map Generator

The Connectivity Map Generator provides an alternative to manually populating and Binding the Connectivity Map components. The Connectivity Map Generator populates the Connectivity Map and binds the components using the information provided by your Java Collaboration Definition. To populate the **cmSAGFA** Connectivity Map using the Connectivity Map Generator, do the following.

1 Create the **cmSAGFA** Connectivity Map as described in **"Creating a Connectivity Map" on page 63**, Steps 1 and 2.

2 From the Project Explorer tree, drag and drop the **jcdSagFA** Collaboration onto the **cmSAGFA** Connectivity Map Editor canvas.

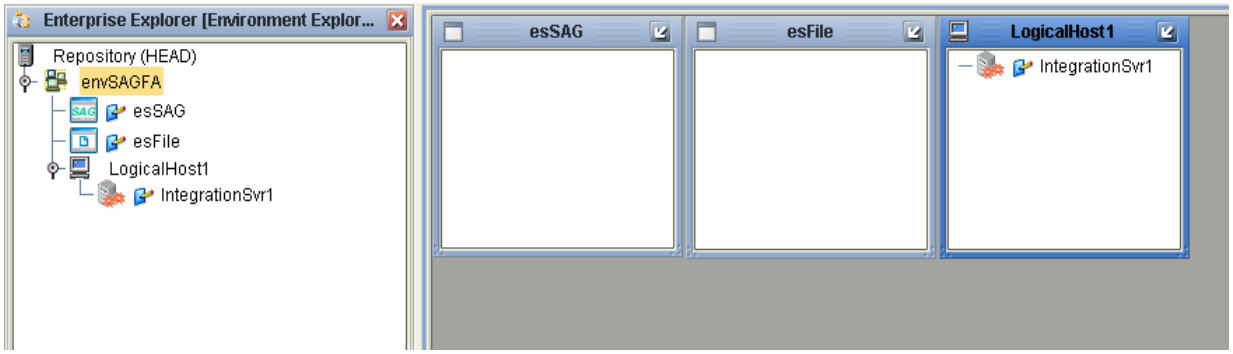
3 From the Connectivity Map Editor toolbar, click the **Connectivity Map Generator**. The Generator populates the Connectivity Map with the necessary components and binds the components according to your Collaboration Definition.

5.5.7 Creating an Environment

Environments include the external systems, Logical Hosts, integration servers and message servers used by a Project and contain the configuration information for these components. Environments are created using the Environment Editor.

1. From the Enterprise Explorer, click the **Environment Explorer** tab.
2. Right-click the Repository and select **New Environment**. A new Environment is added to the Environment Explorer tree.
3. Rename the new Environment to **envSAGFA**.
4. Right-click **envSAGFA** and select **New > SAG External System**. Name the External System **esSAG**. Click **OK**. **esSAG** is added to the Environment Editor.
5. Right-click **envSAGFA** and select **New > File External System**. Name the External System **esFile**. Click **OK**. **esFile** is added to the Environment Editor.
6. Right-click **envSAGFA** and select **New > Logical Host**. **LogicalHost1** is added to the Environment Editor.
7. From the Environment Explorer tree, right-click **LogicalHost1** and select **New > Sun SeeBeyond Integration Server**. A new Integration Server (**IntegrationSvr1**) is added to the Environment Explorer tree under LogicalHost1 (seeFigure 25).

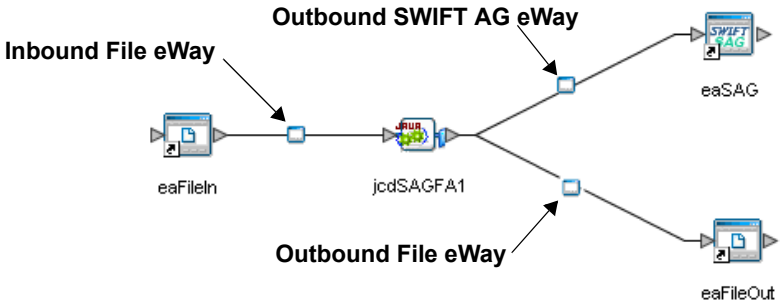
Figure 25 Environment Editor



5.5.8 Configuring the eWays

The prjSagFA Project uses three eWays, each represented in the Connectivity Map as a node between an External Application and a Service (see **Figure 26 on page 67**). eWays facilitate communication and movement of data between the external applications and the eGate system.

Figure 26 eWay Properties - Connectivity Map



Configuring the File eWay Properties

eWay properties are set in both the Connectivity Map and the Environment Explorer.

Modify the File eWay Connectivity Map Properties

1 Double-click the inbound **FileIn eWay**. The **Properties Editor** opens to the inbound File eWay properties. Modify the Inbound File eWay properties for your system, including the settings in Table 17, and click **OK**..

Table 16 Inbound File eWay Settings

Inbound eWay Connectivity Map Properties	
Input file name	input_SAG_FA_.txt

2 In the same way, open and modify the outbound File eWay properties for your system, including the settings in Table 17, and click **OK**.

Table 17 Outbound File eWay Settings

Outbound eWay Connectivity Map Properties	
Output file name	outputSagFA%d.dat

Modify the File eWay Environment Explorer Properties

1 From the **Environment Explorer** tree, right-click the File eWay External System (**esFile** in this sample), and select **Properties** from the shortcut menu. The Properties Editor appears.

2 Modify the File eWay Environment properties for your system, including the settings in Table 18, and click **OK**.

Table 18 File eWay Environment Settings

File eWay Environment Properties	
Inbound File eWay > Parameter Settings - Set as directed, otherwise use the default settings	
Directory	An input directory (for example C:/temp
Outbound File eWay > Parameter Settings - Set as directed, otherwise use the default settings	
Directory	An output directory (for example C:/temp

Configuring the SWIFT AG eWay Properties

The SWIFT AG eWay properties are set in both the Connectivity Map and the Environment Explorer. For more information on the SWIFT AG eWay properties and the Properties Editor, see **"Configuring the eWay" on page 21**.

Modify the SWIFT AG eWay Connectivity Map Properties

1. From the **Connectivity Map**, double-click the **SWIFT AG eWay**. The Properties Editor opens to the SWIFT AG eWay properties.
2. Modify the SWIFT AG eWay Connectivity Map properties for your system, including the settings in Table 19, and click **OK**.

Table 19 SWIFT AG eWay Connectivity Map Properties

SWIFT AG eWay Connectivity Map Properties	
Envelope Settings - Set as directed, otherwise use the default settings	
Application Id	My_AI
Sender	*SAG Operator name*
Sender Auth	*The password for the SAG Operator*
FileAct Client - Set as directed, otherwise use the default settings	
User DN	*SwSec_UserDN. Size limit: 100 bytes. For example, cn=John,o=ptsauszz,o=swift*
Requestor DN	*SwInt_Requestor. Size limit: 100 bytes. For example, o=ptsauszz,o=swift*
Responder DN	*SwInt_Responder. Size limit: 100 bytes. For example, cn=management,o=swift,o=swift*
Service Name	*SwInt_Service. Size limit: 30 bytes. For example, swift.cte, swift.generic.fa!x, swift.generic.fast!x, etc.*
User Reference	*SwInt_RequestRef. Size limit: 30 bytes.*
Remote File Handler TransferEP	*If you are using the Remote File Handler, you must specify your transfer EndPoint name.* *See* **"Remote File Handler TransferEP" on page 34**.
FileAct Client > Get File - Set as directed, otherwise use the default settings	
Transfer Description	Get a file
Transfer info	operation=get
Logical File Name	selftest
Physical File Name	*Sw_PhysicalName. Size limit: 254 bytes.* *For example:* c:\work\SAG_FA\getFromCTE.txt

Modify the SWIFT AG eWay Environment Explorer Properties

1. From the **Environment Explorer** tree, right-click the SWIFT AG eWay External System **(esSunJavaSys** in this sample), and select **Properties** from the shortcut menu. The Properties Editor appears.

2 Modify the SWIFT AG eWay Environment properties for your system, including the settings in Table 20, and click **OK**.

Table 20 SWIFT AG eWay Environment Properties

SWIFT AG eWay Environment Properties	
Transport - Set as directed, otherwise use the default settings	
Read From RA CFG File	sagta_ra.cfg
Host Name	*The Host Name*
Port Number	*The Port Number*

5.5.9 Configuring the Integration Server

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

1 From the Environment Explorer, right-click **IntegrationSvr1** under your **Logical Host**, and select **Properties** from the shortcut menu. The Integration Server Properties Editor appears.

2 Click the **Password** property field under **Sun SeeBeyond Integration Server Configuration**. An ellipsis appears in the property field.

3 Click the ellipsis. The **Password Settings** dialog box appears. Enter **STC** as the **Specific Value** and as the **Confirm Password**, and click **OK**.

4 Click **OK** to accept the new property and close the Properties Editor.

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

5.5.10 Creating the Deployment Profile

Deployment Profiles are specific instances of a Project in a particular Environment. A Deployment Profile is created using the Enterprise Designer's Deployment Editor.

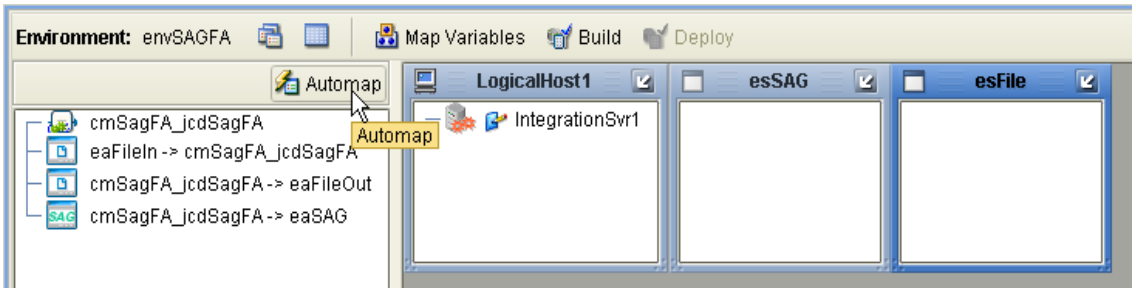
To create a Deployment Profile, do the following:

1 From the Enterprise Explorer's Project Explorer, right-click the Project and select **New** > **Deployment Profile**.

2 From the **Create Deployment Profile** dialog box, enter a name for the Deployment Profile (for this example, **dpSAGFA**). Select the appropriate Environment (**envSAGFA)** and click **OK**.

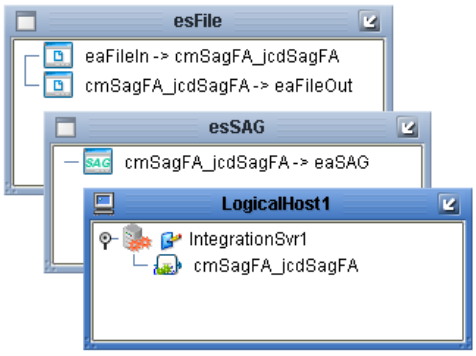
3 Click the **Automap** icon as displayed in **Figure 27 on page 70**.

Figure 27 Deployment Profile - Auto Map



4 The Project's components are automatically mapped to their system windows as seen in Figure 28.

Figure 28 Deployment Profile



5 Click **Activate**. When activation succeeds, save the changes to the Repository.

5.5.11 Initializing your Remote API

Initialize (start) the Remote API instance on your JavaCAPS Integration Server (IS) host prior to starting the JavaCAPS Integration Server and deploying your Project. Running the JavaCAPS IS on top of the RA fulfills the required environment variables. For directions, see **Installing and Initializing the SWIFT AG Remote APIs** on page 18.

5.5.12 Creating and Starting the Domain

To deploy your Project, you must first create a domain. A domain is an instance of a Logical Host.

Create and Start the Domain

1 Navigate to your **<*JavaCAPS51*>\logicalhost** directory (where <*JavaCAPS51*> is the location of your Sun Java Composite Application Platform Suite installation.

2 Double-click the **domainmgr.bat** file. The **Domain Manager** appears.

3 If you have already created a domain, select your domain in the Domain Manager and click the **Start an Existing Domain** button. Once your domain is started, a green check mark indicates that the domain is running.

4 If there are no existing domains, a dialog box indicates that you can create a domain now. Click **Yes**. The **Create Domain** dialog box appears.

5 Make any necessary changes to the **Create Domain** dialog box and click **Create**. The new domain is added to the Domain Manager. Select the domain and click the **Start an Existing Domain** button. Once your domain is started, a green check mark indicates that the domain is running.

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

5.5.13 Building and Deploying the Project

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

Build the Project

1 From the Deployment Editor toolbar, click the **Build** icon.

2 If there are any validation errors, a **Validation Errors** pane will appear at the bottom of the Deployment Editor and displays information regarding the errors. Make any necessary corrections and click **Build** again.

3 After the Build has succeeded you are ready to deploy your Project.

Deploy the Project

1 From the Deployment Editor toolbar, click the **Deploy** icon. Click **Yes** when the **Deploy** prompt appears.

2 A message appears when the project is successfully deployed.

5.5.14 Running the Project

To run your deployed sample Project do the following

1 From your configured input directory, paste (or rename) the sample input file to trigger the eWay. The trigger input file, **input_SAG_FA.txt**, is picked up by the file eWay.

2 The processed output file is published to your output directory. Verify the output data. A sample output data file, **outputSAGFA1.dat** is included in the sample package download for comparison.

5.6 The prjSagIA Sample Project

The prjSagIA Sample Project is similar to the prjSagFA Project, but uses the InterAct service portion of the OTD.

5.6.1 Creating the prjSagIA Project

The first step is to create your Project.

1. From the Enterprise Designer's Project Explorer tree, right-click the Repository and select **New Project**.
2. From the Project Explorer tree, rename the new Project to **prjSagIA**.

5.6.2 Creating the Collaboration Definition

The next step in the sample is to create the **jcdSAGIA** Java Collaboration using the Collaboration Definition Wizard (Java). Once the Collaboration has been created, the Collaboration's Business Rules can be written using the Collaboration Editor (Java).

Creating the jcdSagIA Collaboration Definition

The **jcdSAGIA** Collaboration defines transactions from the inbound File eWay to the SWIFT AG eWay and the outbound File eWay.

1. From the Project Explorer, right-click your new Project and select **New > Collaboration Editor (Java)** from the shortcut menu. The **Collaboration Definition Wizard (Java)** appears.
2. Enter a Collaboration Definition name (for this sample **jcdSAGIA**) and click **Next**.
3. For Step 2 of the wizard, from the Web Services Interfaces selection window, double-click **Sun SeeBeyond** > **eWays** > **File** > **FileClient** > **receive**. The File Name field now displays **receive.** Click **Next**.
4. For Step 3 of the wizard, from the Select OTDs selection window, double-click **Sun SeeBeyond** > **eWays** > **File** > **FileClient**. The **FileClient_1** OTD is added to the Selected OTDs field.
5. Click **Up One Level** to return to the Repository. From the Select OTDs selection window, double-click **Sun SeeBeyond** > **eWays** > **SAG** > **SAGOutboundeWay**. The **SAGOutboundeWay_1** OTD is added to the Selected OTDs field.
6. Click **Finish**. The Collaboration Editor (Java) appears in the right pane of the Enterprise Designer with the new **jcdSagIA** Collaboration.

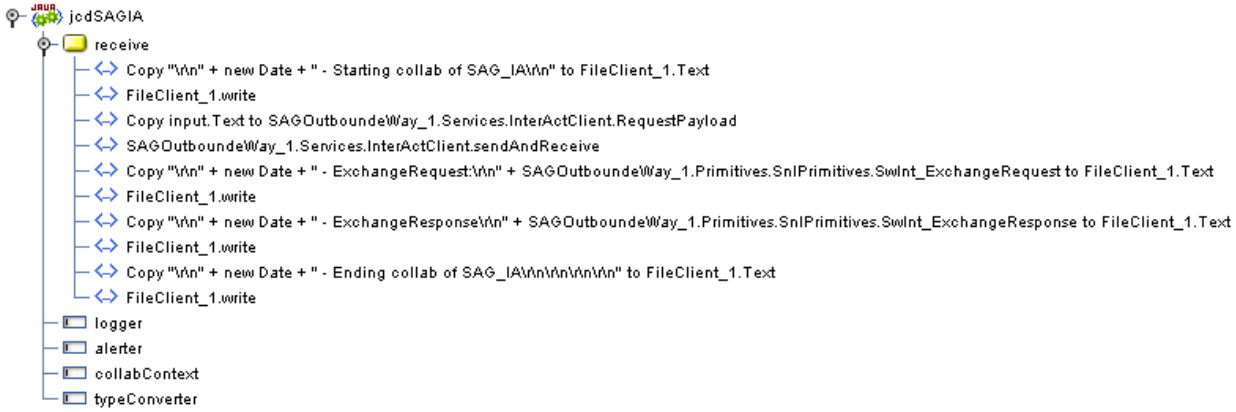
5.6.3 Using the Collaboration Editor (Java)

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

Creating the jcdSAGIA Business Rules

The **jcdSAGIA** Collaboration contains the Business Rules displayed in Figure 29.

Figure 29 jcdSAGIA Collaboration Business Rules



To create the **jcdSAGIA** Collaboration Business Rules using the Java Collaboration Editor's Business Rules Designer, do the following:

1. To open Collaboration Editor to the Sun **jcdSAGIA** Collaboration, double-click **jcdSAGIA** in the Project Explorer tree.
2. The **Copy "\r\n" + new Date + " - Starting collab of SAG_IA\r\n" to FileClient_1.Text** rule creates a "Starting collab of SAG_IA" comment and includes the starting time stamp.

   Create the **Copy "\r\n" + new Date + " - Starting collab of SAG_IA\r\n" to FileClient_1.Text** rule:

   A. From the Business Rules toolbar, click the **rule** icon to add a new rule under **receive** on the Business Rules tree.

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

   C. From the Class Browser, select **Date** as the class, **java.util** as the package, and **Date()** as the constructor. Click **Select**. The **Date** constructor box is added to the Business Rules Designer mapping canvas.

   D. From the Business Rules Designer's **Math** menu, select **Add**. The **Add** box appears.

   E. From the Business Rules Designer's **String** menu, select **Literal String**. The **String** Literal box appears. Enter **\r\n** as the Literal value.

   F. Map the **"\r\n"** output node of the **String** Literal box to the **Value1** input node of the **Add** box.

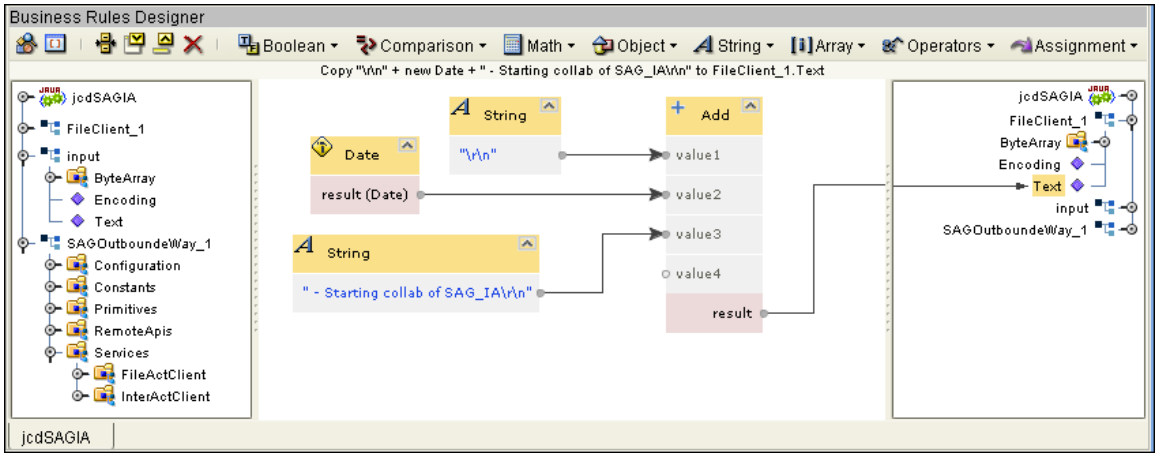
   G. Map the **result (Date)** output node of the **Date** constructor box to the **value2** input node of the **Add** box. To do this, click on the **result (Date)** output node of the **Date** constructor box, and drag your cursor to the input node of the **Add** box. A link now connects the two nodes.

H From the Business Rules Designer's **String** menu, select **Literal String**. The **String** Literal box appears. Enter **- Starting collab of SAG_IA\r\n** as the value.

I Map the **" - Starting collab of SAG_IA\r\n"** output node of the second **String** Literal box to the **Value3** input node of the **Add** box.

J Map the **result** output node of the **Add** box, to Text under FileClient_1 in the right pane of the Business Rules Designer (see Figure 30).

Figure 30 jcdSAGIA Collaboration Business Rules - Create Variable



3 The **FileClient_1.write** rule, used throughout this Collaboration, writes the processed data to the file.

Create the **FileClient_1.write** rule:

A From the Business Rules toolbar, click the **rule** icon to add a new rule.

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

C Select **write()** from the method selection window. The **write** method box appears.

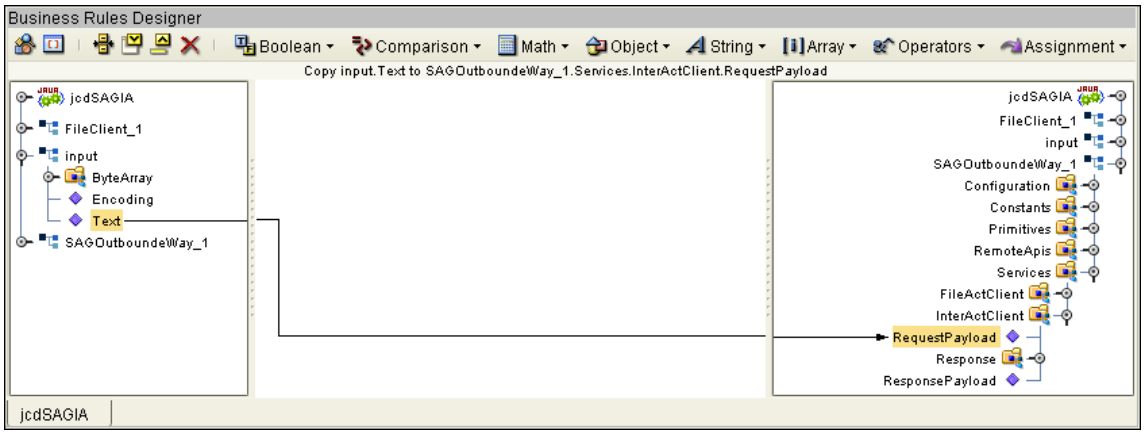
4 The **Copy input.Text to SAGOutboundeWay_1.Services.InterActClient.RequestPayload** rule adds the input text to the InterAct payload.

Create **Copy input.Text to SAGOutboundeWay_1.Services.InterActClient.RequestPayload** rule:

A From the Business Rules toolbar, click the **rule** icon to add a new rule.

B Map **Text** under **input** in the left pane of the Business Rules Designer, to **RequestPayload** under **SAGOutboundeWay_1 > Services > InterActClient** in the right pane of the Business Rules Designer. To do this, click on **Text** in the left pane of the Business Rules Designer, and drag your cursor to **RequestPayload** in the right pane. A link now connects the two nodes (see **Figure 31 on page 75**).

Figure 31 Java Collaboration Editor - jcdSAGIA Business Rules



5 The **SAGOutboundeWay_1.Services.InterActClient.sendAndReceive** rule uses the sendAndReceive method to send the payload

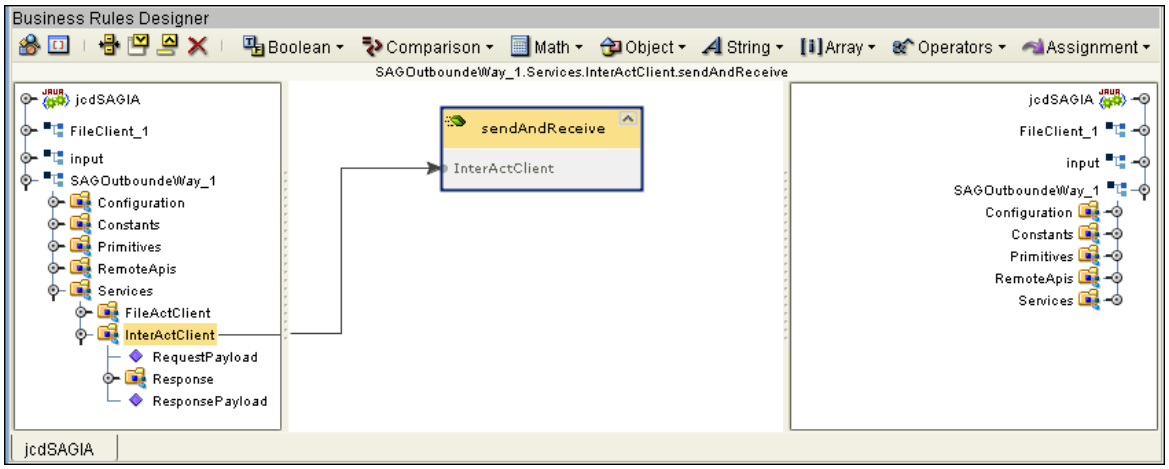
Create the **SAGOutboundeWay_1.Services.InterActClient.sendAndReceive** rule:

A From the Business Rules toolbar, click the **rule** icon to add a new rule to the Business Rules tree.

B From the left pane of the Business Rules Designer, right click **InterActClient**, and click **Select method to call** from the shortcut menu.

C From the method selection box, double-click **sendAndReceive()**. The **sendAndReceive** method box appears (see Figure 32).

Figure 32 Java Collaboration Editor - jcdSAGIA Business Rules



6 The **Copy "\r\n" + new Date + " - ExchangeRequest:\r\n" + SAGOutboundeWay_1.Primitives.SnlPrimitives.SwInt_ExchangeRequest to FileClient_1.Text** rule gets the Primitive value, "File Response," from the SAG server, and includes the time stamp.

Create the **Copy "\r\n" + new Date + " - ExchangeRequest:\r\n" + SAGOutboundeWay_1.Primitives.SnlPrimitives.SwInt_ExchangeRequest to FileClient_1.Text** rule:

A From the Business Rules toolbar, click the **rule** icon to add a new rule.

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

C From the Class Browser, select **Date** as the class, **java.util** as the package, and **Date()** as the constructor. Click **Select**. The Date constructor box is added to the Business Rules Designer mapping canvas.

D From the Business Rules Designer's **Math** menu, select **Add**. The **Add** box appears.

E From the Business Rules Designer's **String** menu, select **Literal String**. The **String** Literal box appears. Enter **\r\n** as the Literal value.

F Map the **"\r\n"** output node of the **String** Literal box to the **Value1** input node of the **Add** box.

G Map the **result (Date)** output node of the **Date** constructor box to the **value2** input node of the **Add** box.

H From the Business Rules Designer's **String** menu, select **Literal String**. The **String** Literal box appears. Enter **- ExchangeRequest:\r\n** as the Literal value.

I Map the **" - ExchangeRequest:\r\n"** output node of the second **String** Literal box to the **Value3** input node of the **Add** box.

J Map **SwInt_ExchangeRequest** under **Primitives > SnlPrimitives** in the left pane of the Business Rules Designer, to the **value4** input node of the **Add** box.

K Map the **result** output node of the **Add** box, to Text under FileClient_1 in the right pane of the Business Rules Designer.

7 Create the **FileClient_1.write** rule:

A Follow the directions provided in step 3 of this section to create another **FileClient_1.write** rule.

8 The **Copy "\r\n" + new Date + " - ExchangeResponse\r\n" + SAGOutboundeWay_1.Primitives.SnlPrimitives.SwInt_ExchangeResponse to FileClient_1.Text** rule gets the Primitive value, "ExchangeResponse," from the SAG server, and includes the time stamp.

Create the **Copy "\r\n" + new Date + " - ExchangeResponse\r\n" + SAGOutboundeWay_1.Primitives.SnlPrimitives.SwInt_ExchangeResponse to FileClient_1.Text** rule:

A From the Business Rules toolbar, click the **rule** icon to add a new rule.

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

C From the Class Browser, select **Date** as the class, **java.util** as the package, and **Date()** as the constructor. Click **Select**. The Date constructor box is added to the Business Rules Designer mapping canvas.

D From the Business Rules Designer's **Math** menu, select **Add**. The **Add** box appears.

E From the Business Rules Designer's **String** menu, select **Literal String**. The **String** Literal box appears. Enter **\r\n** as the Literal value.

F Map the **"\r\n"** output node of the **String** Literal box to the **Value1** input node of the **Add** box.

G Map the **result (Date)** output node of the **Date** constructor box to the **value2** input node of the **Add** box.

H From the Business Rules Designer's **String** menu, select **Literal String**. The **String** Literal box appears. Enter **- ExchangeResponse\r\n** as the Literal value.

I Map the **" - ExchangeResponse\r\n"** output node of the second **String** Literal box to the **Value3** input node of the **Add** box.

J Map **SwInt_ExchangeResponse** under **Primitives > SnlPrimitives** in the left pane of the Business Rules Designer, to the **value4** input node of the **Add** box.

K Map the **result** output node of the **Add** box, to Text under FileClient_1 in the right pane of the Business Rules Designer.

9 Create the **FileClient_1.write** rule:

A Follow the directions provided in step 3 of this section to create another **FileClient_1.write** rule.

10 The **Copy "\r\n" + new Date + " - Ending collab of SAG_IA\r\n\r\n\r\n\r\n" to FileClient_1.Text** rule creates the "Ending collab of SAG_IA" comment and includes the starting time and date.

Create the **Copy "\r\n" + new Date + " - Ending collab of SAG_IA\r\n\r\n\r\n\r\n" to FileClient_1.Text** rule:

A From the Business Rules toolbar, click the **rule** icon to add a new rule.

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

C From the Class Browser, select **Date** as the class, **java.util** as the package, and **Date()** as the constructor. Click **Select**. The Date constructor box is added to the Business Rules Designer mapping canvas.

D From the Business Rules Designer's **Math** menu, select **Add**. The **Add** box appears.

E From the Business Rules Designer's **String** menu, select **Literal String**. The **String** Literal box appears. Enter **\r\n** as the Literal value.

F Map the **"\r\n"** output node of the **String** Literal box to the **Value1** input node of the **Add** box.

G Map the **result (Date)** output node of the **Date** constructor box to the **value2** input node of the **Add** box.

H From the Business Rules Designer's **String** menu, select **Literal String**. The **String** Literal box appears. Enter **- Ending collab of SAG_IA\r\n\r\n\r\n\r\n** as the Literal value.

I Map the **" - Ending collab of SAG_IA\r\n\r\n\r\n\r\n"** output node of the **String** Literal box to the **Value3** input node of the **Add** box.

J Map the **result** output node of the **Add** box, to Text under FileClient_1 in the right pane of the Business Rules Designer (see Figure 21).

11 Create the **FileClient_1.write** rule:

A Follow the directions provided in step 3 of this section to create another **FileClient_1.write** rule.

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

13 Save your current changes to the repository.

For more information on how to create Business Rules using the Collaboration Editor see the *Sun SeeBeyond eGate™ Integrator User's Guide*.

5.6.4 Creating the Connectivity Map

To create the Project's **cmSAGIA** Connectivity Map, do the following:

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

2 The New Connectivity Map appears. From the Project Explorer tree, rename the Connectivity Map to **cmSAGIA**.

5.6.5 Generate the Connectivity Map Using the Connectivity Map Generator

The Connectivity Map Generator provides an alternative to manually populating and Binding the Connectivity Map components. The Connectivity Map Generator populates the Connectivity Map and binds the components using the information provided by your Java Collaboration Definition. To populate the **cmSAGIA** Connectivity Map using the Connectivity Map Generator, do the following.

1 From the Project Explorer tree, drag and drop the **jcdSAGIA** Collaboration onto the **cmSAGIA** Connectivity Map Editor canvas.

2 From the Connectivity Map Editor toolbar, click the **Connectivity Map Generator**. The Generator populates the Connectivity Map with the necessary components and binds the components according to your Collaboration Definition.

3 Save your changes to the Repository.

5.6.6 Creating an Environment

Environments include the external systems, Logical Hosts, integration servers and message servers used by a Project and contain the configuration information for these components. Environments are created using the Environment Editor.

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

2 Right-click the Repository and select **New Environment**. A new Environment is added to the Environment Explorer tree.

3 Rename the new Environment to **envSAGIA**.

4 Right-click **envSAGIA** and select **New > SAG External System**. Name the External System **esSAG**. Click **OK**. **esSAG** is added to the Environment Editor.

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

6 Right-click **envSAGIA** and select **New > Logical Host**. **LogicalHost1** is added to the Environment Editor.

7 From the Environment Explorer tree, right-click **LogicalHost1** and select **New > Sun SeeBeyond Integration Server**. A new Integration Server (**IntegrationSvr1**) is added to the Environment Explorer tree under LogicalHost1.

5.6.7 Configuring the eWays

The prjSagIA Project uses three eWays, each represented in the Connectivity Map as a node between an External Application and a Service.

Configuring the File eWay Properties

eWay properties are set in both the Connectivity Map and the Environment Explorer.

Modify the File eWay Connectivity Map Properties

1 Double-click the inbound **FileIn eWay**. The **Properties Editor** opens to the inbound File eWay properties. Modify the Inbound File eWay properties for your system, including the settings in Table 21, and click **OK**..

Table 21 Inbound File eWay Settings

Inbound eWay Connectivity Map Properties	
Input file name	input_SAG_IA.txt

2 In the same way, open and modify the outbound File eWay properties for your system, including the settings in Table 22, and click **OK**.

Table 22 Outbound File eWay Settings

Outbound eWay Connectivity Map Properties	
Output file name	outputSAGIA%d.dat

Modify the File eWay Environment Explorer Properties

1 From the **Environment Explorer** tree, right-click the File eWay External System (**esFile** in this sample), and select **Properties** from the shortcut menu. The Properties Editor appears.

2 Modify the File eWay Environment properties for your system, including the settings in **Table 23 on page 80**, and click **OK**.

Table 23 File eWay Environment Settings

File eWay Environment Properties	
Inbound File eWay > Parameter Settings - Set as directed, otherwise use the default settings	
Directory	An input directory (for example C:/temp
Outbound File eWay > Parameter Settings - Set as directed, otherwise use the default settings	
Directory	An output directory (for example C:/temp

Configuring the SWIFT AG eWay Properties

The SWIFT AG eWay properties are set in both the Connectivity Map and the Environment Explorer. For more information on the SWIFT AG eWay properties and the Properties Editor, see **"Configuring the eWay" on page 21**.

Modify the SWIFT AG eWay Connectivity Map Properties

1. From the **Connectivity Map**, double-click the **SWIFT AG eWay**. The Properties Editor opens to the SWIFT AG eWay properties.
2. Modify the SWIFT AG eWay Connectivity Map properties for your system, including the settings in Table 24, and click **OK**.

Table 24 SWIFT AG eWay Connectivity Map Properties

SWIFT AG eWay Connectivity Map Properties	
Envelope Settings - Set as directed, otherwise use the default settings	
Application Id	My_AI
Sender	*SAG Operator name*
Sender Auth	*The password for the SAG Operator*
InterAct Client - Set as directed, otherwise use the default settings	
User DN	*SwSec_UserDN. Size limit: 100 bytes. For example, cn=John,o=ptsauszz,o=swift*
Requestor DN	*SwInt_Requestor. Size limit: 100 bytes. For example, o=ptsauszz,o=swift*
Responder DN	*SwInt_Responder. Size limit: 100 bytes. For example, cn=management,o=swift,o=swift*
Service Name	*SwInt_Service. Size limit: 30 bytes. For example, swift.cte, swift.generic.fa!x, swift.generic.fast!x, etc.*
User Reference	*SwInt_RequestRef. Size limit: 30 bytes.*

Modify the SWIFT AG eWay Environment Explorer Properties

1. From the **Environment Explorer** tree, right-click the SWIFT AG eWay External System **(esSAG** in this sample), and select **Properties** from the shortcut menu. The Properties Editor appears.
2. Modify the SWIFT AG eWay Environment properties for your system, including the settings in Table 25, and click **OK**.

Table 25 SWIFT AG eWay Environment Properties

SWIFT AG eWay Environment Properties	
Transport - Set as directed, otherwise use the default settings	
Read From RA CFG File	sagta_ra.cfg
Host Name	*The Host Name*
Port Number	*The Port Number*

5.6.8 Configuring the Integration Server

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

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

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

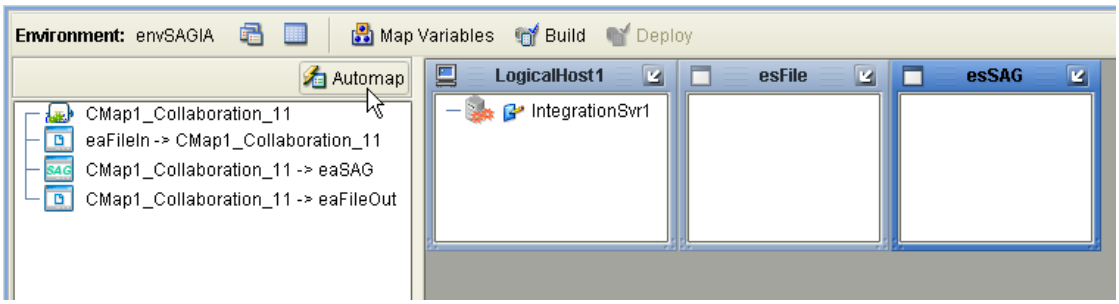
5.6.9 Creating the Deployment Profile

Deployment Profiles are specific instances of a Project in a particular Environment. A Deployment Profile is created using the Enterprise Designer's Deployment Editor.

To create a Deployment Profile, do the following:

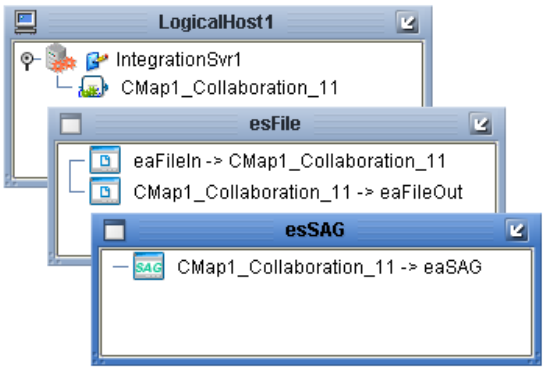
1. From the Enterprise Explorer's Project Explorer, right-click the Project and select **New** > **Deployment Profile**.
2. From the **Create Deployment Profile** dialog box, enter a name for the Deployment Profile (for this example, **dpSAGIA**). Select the appropriate Environment (**envSAGIA)** and click **OK**.
3. Click the **Auto Map** icon as displayed in Figure 33.

Figure 33 Deployment Profile - Auto Map



4 The Project's components are automatically mapped to their system windows as seen in Figure 34.

Figure 34 Deployment Profile



5 Click **Activate**. When activation succeeds, save the changes to the Repository.

5.6.10 Initializing your Remote API

Initialize (start) the Remote API instance on your JavaCAPS Integration Server (IS) host prior to starting the JavaCAPS Integration Server and deploying your Project. Running the JavaCAPS IS on top of the RA fulfills the required environment variables. For directions, see **Installing and Initializing the SWIFT AG Remote APIs** on page 18.

5.6.11 Creating and Starting the Domain

To deploy your Project, you must first create a domain. A domain is an instance of a Logical Host.

Create and Start the Domain

1 Navigate to your ***<JavaCAPS51>*\logicalhost** directory (where *<JavaCAPS51>* is the location of your Sun Java Composite Application Platform Suite installation.

2 Double-click the **domainmgr.bat** file. The **Domain Manager** appears.

3 If you have already created a domain, select your domain in the Domain Manager and click the **Start an Existing Domain** button. Once your domain is started, a green check mark indicates that the domain is running.

4 If there are no existing domains, a dialog box indicates that you can create a domain now. Click **Yes**. The **Create Domain** dialog box appears.

5 Make any necessary changes to the **Create Domain** dialog box and click **Create**. The new domain is added to the Domain Manager. Select the domain and click the **Start an Existing Domain** button. Once your domain is started, a green check mark indicates that the domain is running.

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

5.6.12 Building and Deploying the Project

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

Build the Project

1 From the Deployment Editor toolbar, click the **Build** icon.

2 If there are any validation errors, a **Validation Errors** pane will appear at the bottom of the Deployment Editor and displays information regarding the errors. Make any necessary corrections and click **Build** again.

3 After the Build has succeeded you are ready to deploy your Project.

Deploy the Project

1 From the Deployment Editor toolbar, click the **Deploy** icon. Click **Yes** when the **Deploy** prompt appears.

2 A message appears when the project is successfully deployed.

5.6.13 Running the Project

To run your deployed sample Project do the following

1 From your configured input directory, paste (or rename) the sample input file to trigger the eWay. The trigger input file, **input_SAG_IA.txt**, is picked up by the file eWay.

2 The processed output file is published to your output directory. Verify the output data. A sample output data file, **outputSAGIA1.dat** is included in the sample package download for comparison.

Appendix A

Sample prjSAGCert Project Overview

This appendix provides the Java code that is represented by the Business Rules of the SWIFT AG eWay sample Project's jcdSAGCert Collaboration.

What's in this Appendix:

A.1 The prjSAGCert Sample Project Collaboration

The prjSAGCert Sample Project demonstrates several different scenarios, and therefore provides a good example of how the logic for these scenarios to create.

jcdSAGCert Collaboration Definition Scenarios

The jcdSAGCert Java Collaboration Definition performs the following scenarios:

Scenario 1: InterAct Real Time swift.cte

Scenario 1.1: InterAct Real Time synchronous cte

Got InterAct ResponsePayload [<Response><ComRes><Result></Result><OutXmitTime></OutXmitTime><OutInt>0</OutInt></ComRes></Response>]

Scenario 1.2: InterAct Real Time asynchronous swift.cte

Scenario 1.2.1: InterAct Real Time asynchronous cte send 1st request...3(token)

Scenario 1.2.2: InterAct Real Time asynchronous cte send 2nd request...4(token)

Scenario 1.2.3: InterAct Real Time asynchronous cte receive 1st response...3

Scenario 1.2.4: InterAct Real Time asynchronous cte receive 2nd response...4

Scenario 1.3: InterAct Real Time cte signed request

Scenario 2: FileAct Real Time get file

Scenario 2.1: FileAct RealTime get file swift.cte

Scenario 2.2 (to be continued on 2.6): FileAct RealTime get 1 MB file non-blocking asynchronous

Scenario 2.3: FileAct RealTime synchronous get file

Scenario 2.4: FileAct RealTime get file, signed request and flagged urgent

Scenario 2.5: FileAct RealTime get file with Non-Repudiation

Scenario 2.6 (continued on 2.2): FileAct RealTime get of 1 MB file non-blocking asynchronous, and check transfer status

Scenario 3: FileAct RealTime put file

Scenario 3.1 (to be continued on 3.5): FileAct RealTime put 1 MB file non-blocking asynchronous

Scenario 3.2: FileAct RealTime put file blocking synchronous

Scenario 3.3: FileAct RealTime put file signed request

Scenario 3.4: FileAct RealTime put file Non-Repudiation

Scenario 3.5 (continued on 3.1): FileAct RealTime put 1 MB file non-blocking asynchronous, and check transfer status

Scenario 4: Store & Forward Queue Non-Recovery mode

Scenario 5: FileAct Delivery Notification Queue

Scenario 5.1: FileAct given Delivery Notification Queue 1st file

Scenario 5.2: FileAct given Delivery Notification Queue 2nd file

Scenario 5.3: FileAct given Delivery Notification Queue signed request 3rd file

Scenario 5.4: FileAct given Delivery Notification Queue with Non-Repudiation 4th file

Scenario 6: Fetch file

Wait a while for SwiftNet Store & Forward Queue access

Scenario 6.1: Fetch file 1st file

Scenario 6.2: Fetch file 2nd file

Scenario 6.3: Fetch file 3rd file

Scenario 6.4: Fetch file Non-Repudiation 4th file

Scenario 7: Store & Forward messages from Queue in Recovery Mode

Scenario 8: InterAct given Delivery Notification Queue

Scenario 8.1: InterAct given Delivery Notification Queue 1st file

Scenario 8.2: InterAct given Delivery Notification Queue 2nd file

Scenario 8.3: InterAct given Delivery Notification Queue signed request 3rd file

Scenario 9: Pull messages from Queue

Wait a while for SwiftNet Store & Forward Queue access

Scenario 9.1: Pull messages from Queue 1st file

Scenario 9.2: Pull messages from Queue 2nd file

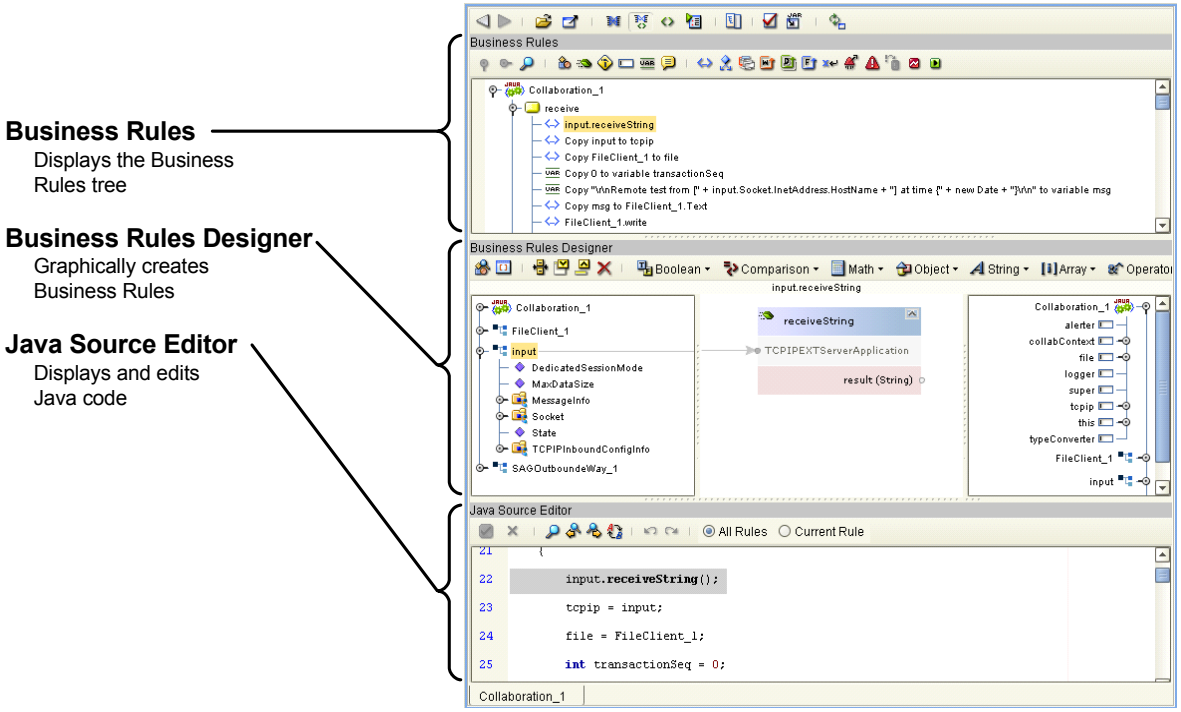
Scenario 9.3: Pull messages from Queue 3rd file

Scenario 10: Show more usages of other API/Primitive

The Java Collaboration Editor

The Enterprise Designer's Collaboration Editor (Java) allows you to create Business Rules using the Business Rules Designer, a graphical interface used to graphically add methods and map data paths. This creates the Java code that drives the Collaboration's business logic. You can see this Java code from the Collaboration Editor's Java Source Editor (see Figure 35).

Figure 35 Java Collaboration Editor



jcdSAGCert Collaboration Java Code

The jcdSAGCert sample Project's jcdSAGCert Collaboration contains the following Java code:

```
package prjSAGCert;


public class jcdSAGCert
{

    public com.stc.codegen.logger.Logger logger;

    public com.stc.codegen.alerter.Alerter alerter;

    public com.stc.codegen.util.CollaborationContext collabContext;

    public com.stc.codegen.util.TypeConverter typeConverter;

    private com.stc.connector.appconn.file.FileApplication file;

    public void receive( com.stc.connector.appconn.file.FileTextMessage input,
com.stc.connector.sagadapter.sagapi.SAGOutbound SAGOutboundeWay_1,
com.stc.connector.appconn.file.FileApplication FileClient_1 )
        throws Throwable
```

```
    {
        file = FileClient_1;
        int transactionSeq = 0;
        String msg = input.getText();
        msg = input.getText();
        FileClient_1.setText( msg );
        FileClient_1.write();
        track( "\r\n*************************************************" );
        track( "\r\n*  Start of remote test for SAG Certification   *" );
        track( "\r\n*************************************************" );
        track( "\r\n\r\nStart of collaboration from SAG_Cert" );
        java.util.Date startTime = new java.util.Date();
        java.text.SimpleDateFormat format = new java.text.SimpleDateFormat( "dd/MM/yyyy
HH:mm:ss.SSS" );
        String from = format.format( startTime );
       msg = "\r\n\r\n" + ++transactionSeq + ".  Scenario 1: SAG_Cert: RealTime/InterAct/swift.cte
...";
        track( msg );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 1.1: SAG_Cert: RT/IA/cte/Sync ...";
        track( msg );
        SAGOutboundeWay_1.getServices().getInterActClient().setRequestPayload( msg );
        SAGOutboundeWay_1.getServices().getInterActClient().sendAndReceive();
        track( " OK" );
        msg = "\r\n\r\n                  Got InterAct ResponsePayload [" +
SAGOutboundeWay_1.getServices().getInterActClient().getResponsePayload() + "]";
        track( msg );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 1.2: SAG_Cert: RT/IA/swift.cte/ASync
...";
        track( msg );
       msg = "\r\n\r\n" + ++transactionSeq + ".      Scenario 1.2.1: SAG_Cert: RT/IA/cte/ASync/send
1st req ...";
        track( msg );
        SAGOutboundeWay_1.getServices().getInterActClient().setRequestPayload( msg );
        long t1 = SAGOutboundeWay_1.getServices().getInterActClient().sendRequest();
        track( t1 + "(token) OK" );
       msg = "\r\n\r\n" + ++transactionSeq + ".      Scenario 1.2.2: SAG_Cert: RT/IA/cte/ASync/send
2nd req ...";
        track( msg );
        SAGOutboundeWay_1.getServices().getInterActClient().setRequestPayload( msg );
        long t2 = SAGOutboundeWay_1.getServices().getInterActClient().sendRequest();
        track( t2 + "(token) OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".      Scenario 1.2.3: SAG_Cert: RT/IA/cte/ASync/
receive 1st resp ...";
        track( msg );
        SAGOutboundeWay_1.getServices().getInterActClient().receiveResponse( t1 );
        track( t1 + " OK" );
       msg = "\r\n\r\n" + ++transactionSeq + ".      Scenario 1.2.4: SAG_Cert: RT/IA/cte/ASync/recv
2nd resp ...";
        track( msg );
        SAGOutboundeWay_1.getServices().getInterActClient().receiveResponse( t2 );
        track( t2 + " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 1.3: SAG_Cert: RT/IA/cte/Signed ...";
        track( msg );
        SAGOutboundeWay_1.getServices().getInterActClient().setRequestPayload( msg );
        SAGOutboundeWay_1.getConfiguration().getInterActClient().setSwInt_RequestCrypto(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getTRUE() );
        SAGOutboundeWay_1.getServices().getInterActClient().sendAndReceive();
        SAGOutboundeWay_1.getConfiguration().getInterActClient().setSwInt_RequestCrypto(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getFALSE() );

SAGOutboundeWay_1.getPrimitives().getSnlPrimitives().getSwInt_ExchangeRequest().getSwInt_Request()
.removeSwSec_Crypto();
        track( " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".  Scenario 2: SAG_Cert: RealTime/FileAct/Get ...";
        track( msg );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 2.1: SAG_Cert: RT/FA/Get/swift.cte
...";
        track( msg );

SAGOutboundeWay_1.getConfiguration().getFileActClient().getGetFile().setSw_TransferDescription(
msg );
        SAGOutboundeWay_1.getServices().getFileActClient().get();
        track( " OK" );
       msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 2.2 (to be continued on 2.6): SAG_Cert:
RT/FA/Get/generic.fa!x/Non-Blocking/ASync/1MB-file ...";
        track( msg );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setBlockFileTransfer(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getFALSE() );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setSwInt_Service(
"swift.generic.fa!x" );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setSwInt_Responder(
"o=ptsauszz,o=swift" );

SAGOutboundeWay_1.getConfiguration().getFileActClient().getGetFile().setSw_TransferDescription(
msg );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getGetFile().setSw_PhysicalName(
"c:\\temp\\SAG_Cert\\getDownload2FromFTA_1MB.txt" );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getGetFile().setSw_LogicalName(
"downloadFile2_1MB.txt" );
        String getTransferRef22 = SAGOutboundeWay_1.getServices().getFileActClient().get();
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setBlockFileTransfer(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getTRUE() );
```

```
        track( getTransferRef22 + " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 2.3: SAG_Cert: RT/FA/Get/generic.fa!x/
Sync ...";
        track( msg );

SAGOutboundeWay_1.getConfiguration().getFileActClient().getGetFile().setSw_TransferDescription(
msg );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getGetFile().setSw_PhysicalName(
"c:\\temp\\SAG_Cert\\getDownload3FromFTA.txt" );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getGetFile().setSw_LogicalName(
"downloadFile3.txt" );
        SAGOutboundeWay_1.getServices().getFileActClient().get();
        track( " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 2.4: SAG_Cert: RT/FA/Get/generic.fa!x/
Signed/Urgent ...";
        track( msg );

SAGOutboundeWay_1.getConfiguration().getFileActClient().getGetFile().setSw_TransferDescription(
msg );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setSwInt_RequestCrypto(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getTRUE() );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setSwInt_Priority(
SAGOutboundeWay_1.getConstants().getSwInt_Priority().getUrgent() );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getGetFile().setSw_PhysicalName(
"c:\\temp\\SAG_Cert\\getDownload4FromFTA.txt" );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getGetFile().setSw_LogicalName(
"downloadFile4.txt" );
        SAGOutboundeWay_1.getServices().getFileActClient().get();
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setSwInt_RequestCrypto(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getFALSE() );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setSwInt_Priority(
SAGOutboundeWay_1.getConstants().getSwInt_Priority().getNormal() );
        track( " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 2.5: SAG_Cert: RT/FA/Get/generic.fa!x/
Non-Rep ...";
        track( msg );

SAGOutboundeWay_1.getConfiguration().getFileActClient().getGetFile().setSw_TransferDescription(
msg );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setSwInt_NRIndicator(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getTRUE() );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getGetFile().setSw_PhysicalName(
"c:\\temp\\SAG_Cert\\getDownload5FromFTA.txt" );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getGetFile().setSw_LogicalName(
"downloadFile5.txt" );
        SAGOutboundeWay_1.getServices().getFileActClient().get();
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setSwInt_NRIndicator(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getFALSE() );
        track( " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 2.6 (continued on 2.2): SAG_Cert: RT/
FA/Get/generic.fa!x/Non-Blocking/ASync/1MB-file/CheckTransferStatus on " + getTransferRef22 + "
...";
        track( msg );
        for (int i = 0; i < 100; i++) {
            String status = SAGOutboundeWay_1.getServices().getFileActClient().checkFileStatus(
getTransferRef22 );
            msg = "\r\n\r\n          " + status + " - " +
SAGOutboundeWay_1.getPrimitives().getSnlPrimitives().getSw_GetFileStatusResponse().getSw_FileStatu
s().getSw_CurrentSize().getX__PCDATA__() + " of " +
SAGOutboundeWay_1.getPrimitives().getSnlPrimitives().getSw_GetFileStatusResponse().getSw_FileStatu
s().getSw_Size().getX__PCDATA__() + " (bytes)";
            track( msg );
            if
(SAGOutboundeWay_1.getConstants().getSw_TransferStatus().getCompleted().equalsIgnoreCase( status )
|| SAGOutboundeWay_1.getConstants().getSw_TransferStatus().getDuplicated().equalsIgnoreCase(
status ) || SAGOutboundeWay_1.getConstants().getSw_TransferStatus().getUnknown().equalsIgnoreCase(
status ) || SAGOutboundeWay_1.getConstants().getSw_TransferStatus().getFailed().equalsIgnoreCase(
status ) || SAGOutboundeWay_1.getConstants().getSw_TransferStatus().getRejected().equalsIgnoreCase(
status ) || SAGOutboundeWay_1.getConstants().getSw_TransferStatus().getAborted().equalsIgnoreCase(
status )) {
                track( " ... " );
                break;
            }
            Thread.sleep( 20000 );
        }
        track( " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".  Scenario 3: SAG_Cert: RealTime/FileAct/Put/
swift.generic.fa!x ...";
        track( msg );
       msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 3.1 (to be continued on 3.5): SAG_Cert:
RT/FA/Put/Non-Blocking/ASync/1MB-file ...";
        track( msg );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setBlockFileTransfer(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getFALSE() );

SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_TransferDescription(
msg );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_PhysicalName(
"C:\\temp\\SAG_Cert\\testFile1ToFTA_1MB.txt" );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_LogicalName(
"testFile1ToFTA_1MB" );
        String putTransferRef31 = SAGOutboundeWay_1.getServices().getFileActClient().put();
```

```
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setBlockFileTransfer(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getTRUE() );
        track( putTransferRef31 + " OK" );
       msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 3.2: SAG_Cert: RT/FA/Put/Blocking/Sync
...";
        track( msg );

SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_TransferDescription(
msg );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_PhysicalName(
"C:\\temp\\SAG_Cert\\testFile2ToFTA.txt" );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_LogicalName(
"testFile2ToFTA" );
        SAGOutboundeWay_1.getServices().getFileActClient().put();
        track( " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 3.3: SAG_Cert: RT/FA/Put/Signed ...";
        track( msg );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setSwInt_RequestCrypto(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getTRUE() );

SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_TransferDescription(
msg );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_PhysicalName(
"C:\\temp\\SAG_Cert\\testFile3ToFTA.txt" );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_LogicalName(
"testFile3ToFTA" );
        String transferRef31 = SAGOutboundeWay_1.getServices().getFileActClient().put();
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setSwInt_RequestCrypto(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getFALSE() );
        track( " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 3.4: SAG_Cert: RT/FA/Put/Non-Rep ...";
        track( msg );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setSwInt_NRIndicator(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getTRUE() );

SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_TransferDescription(
msg );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_PhysicalName(
"C:\\temp\\SAG_Cert\\testFile4ToFTA.txt" );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_LogicalName(
"testFile4ToFTA" );
        SAGOutboundeWay_1.getServices().getFileActClient().put();
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setSwInt_NRIndicator(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getFALSE() );
        track( " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 3.5 (continued on 3.1): SAG_Cert: RT/
FA/Put/Non-Blocking/ASync/1MB-file/CheckTransferStatus on " + putTransferRef31 + " ...";
        track( msg );
        for (int i = 0; i < 100; i++) {
            String status = SAGOutboundeWay_1.getServices().getFileActClient().checkFileStatus(
putTransferRef31 );
            msg = "\r\n\r\n         " + status + " - " +
SAGOutboundeWay_1.getPrimitives().getSnlPrimitives().getSw_GetFileStatusResponse().getSw_FileStatu
s().getSw_CurrentSize().getX__PCDATA__() + " of " +
SAGOutboundeWay_1.getPrimitives().getSnlPrimitives().getSw_GetFileStatusResponse().getSw_FileStatu
s().getSw_Size().getX__PCDATA__() + " (bytes)";
            track( msg );
            if
(SAGOutboundeWay_1.getConstants().getSw_TransferStatus().getCompleted().equalsIgnoreCase( status )
|| SAGOutboundeWay_1.getConstants().getSw_TransferStatus().getDuplicated().equalsIgnoreCase(
status ) || SAGOutboundeWay_1.getConstants().getSw_TransferStatus().getUnknown().equalsIgnoreCase(
status ) || SAGOutboundeWay_1.getConstants().getSw_TransferStatus().getFailed().equalsIgnoreCase(
status ) || SAGOutboundeWay_1.getConstants().getSw_TransferStatus().getRejected().equalsIgnoreCase(
status ) || SAGOutboundeWay_1.getConstants().getSw_TransferStatus().getAborted().equalsIgnoreCase(
status )) {
                track( " ... " );
                break;
            }
            Thread.sleep( 20000 );
        }
        track( " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".  Scenario 4: SAG_Cert: SnF/Queue/Non-RecoveryMode
...";
        track( msg );
        SAGOutboundeWay_1.getServices().getFileActClient().snfAcquire();
        SAGOutboundeWay_1.getServices().getFileActClient().snfRelease();
        track( " OK" );
       msg = "\r\n\r\n" + ++transactionSeq + ".  Scenario 5: SAG_Cert: FileAct/NotifQueue/Delivery
...";
        track( msg );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 5.1: SAG_Cert: FA/Delivery/1st file
...";
        track( msg );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setSwInt_Service(
"swift.generic.fast!x" );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setSwInt_NotifQueue(
"ptsauszz_generic!x" );

SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_TransferDescription(
msg );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_PhysicalName(
"C:\\temp\\SAG_Cert\\testFile1ToSnF.txt" );
```

```
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_LogicalName(
"testFile1ToSnF" );
        SAGOutboundeWay_1.getServices().getFileActClient().put();
        track( " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 5.2: SAG_Cert: FA/Delivery/2nd file
...";
        track( msg );

SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_TransferDescription(
msg );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_PhysicalName(
"C:\\temp\\SAG_Cert\\testFile2ToSnF.txt" );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_LogicalName(
"testFile2ToSnF" );
        SAGOutboundeWay_1.getServices().getFileActClient().put();
        track( " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 5.3: SAG_Cert: FA/Delivery/Signed/3rd
file ...";
        track( msg );

SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_TransferDescription(
msg );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setSwInt_RequestCrypto(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getTRUE() );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_PhysicalName(
"C:\\temp\\SAG_Cert\\testFile3ToSnF.txt" );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_LogicalName(
"testFile3ToSnF" );
        SAGOutboundeWay_1.getServices().getFileActClient().put();
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setSwInt_RequestCrypto(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getFALSE() );
        track( " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 5.4: SAG_Cert: FA/Delivery/
NonRepudiation/4th file ...";
        track( msg );

SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_TransferDescription(
msg );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_PhysicalName(
"C:\\temp\\SAG_Cert\\testFile4ToSnF.txt" );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().getPutFile().setSw_LogicalName(
"testFile4ToSnF" );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setSwInt_NRIndicator(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getTRUE() );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setSwInt_RequestCrypto(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getTRUE() );
        SAGOutboundeWay_1.getServices().getFileActClient().put();
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setSwInt_RequestCrypto(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getFALSE() );
        SAGOutboundeWay_1.getConfiguration().getFileActClient().setSwInt_NRIndicator(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getFALSE() );
        track( " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".  Scenario 6: SAG_Cert: FetchFile ...";
        track( msg );
        msg = "\r\n\r\n       Wait a while for SwiftNet SnF Queue access...";
        track( msg );
        Thread.sleep( 10000 );
        track( " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 6.1: SAG_Cert: FetchFile/1st file ...";
        track( msg );
        SAGOutboundeWay_1.getServices().getFileActClient().snfAcquire();
        SAGOutboundeWay_1.getServices().getFileActClient().snfPull();

SAGOutboundeWay_1.getConfiguration().getFileActClient().getSnFFetchFile().setSw_PhysicalName(
"C:\\temp\\SAG_Cert\\fetch1FromSnF.txt" );
        SAGOutboundeWay_1.getServices().getFileActClient().snfFetch();
        track( " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 6.2: SAG_Cert: FetchFile/2nd file ...";
        track( msg );
        SAGOutboundeWay_1.getServices().getFileActClient().snfPull();

SAGOutboundeWay_1.getConfiguration().getFileActClient().getSnFFetchFile().setSw_PhysicalName(
"C:\\temp\\SAG_Cert\\fetch2FromSnF.txt" );
        SAGOutboundeWay_1.getServices().getFileActClient().snfFetch();
        SAGOutboundeWay_1.getServices().getFileActClient().snfAck();
        SAGOutboundeWay_1.getServices().getFileActClient().snfRelease();
        track( " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 6.3: SAG_Cert: FetchFile/3rd file ...";
        track( msg );
        SAGOutboundeWay_1.getServices().getFileActClient().snfAcquire();
        SAGOutboundeWay_1.getServices().getFileActClient().snfPull();

SAGOutboundeWay_1.getConfiguration().getFileActClient().getSnFFetchFile().setSw_PhysicalName(
"C:\\temp\\SAG_Cert\\fetch3FromSnF.txt" );
        SAGOutboundeWay_1.getServices().getFileActClient().snfFetch();
        SAGOutboundeWay_1.getServices().getFileActClient().snfAck();
        SAGOutboundeWay_1.getServices().getFileActClient().snfRelease();
        track( " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 6.4: SAG_Cert: FetchFile/
NonRepudiation/4th file ...";
        track( msg );
        SAGOutboundeWay_1.getServices().getFileActClient().snfAcquire();
```

```
        SAGOutboundeWay_1.getServices().getFileActClient().snfPull();

SAGOutboundeWay_1.getConfiguration().getFileActClient().getSnFFetchFile().setSw_PhysicalName(
"C:\\temp\\SAG_Cert\\fetch4FromSnF.txt" );
        SAGOutboundeWay_1.getServices().getFileActClient().snfFetch();

SAGOutboundeWay_1.getPrimitives().getSnlPrimitives().getSw_ExchangeSnFRequest().getSw_SnFRequest()
.getSw_SnFRequestControl().getSwInt_NRIndicator().setX__PCDATA__(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getTRUE() );
        SAGOutboundeWay_1.getServices().getFileActClient().snfAck();

SAGOutboundeWay_1.getPrimitives().getSnlPrimitives().getSw_ExchangeSnFRequest().getSw_SnFRequest()
.getSw_SnFRequestControl().getSwInt_NRIndicator().setX__PCDATA__(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getFALSE() );
        SAGOutboundeWay_1.getServices().getFileActClient().snfRelease();
        track( " OK" );
       msg = "\r\n\r\n" + ++transactionSeq + ".  Scenario 7: SAG_Cert: SnF/Queue/RecoveryMode ...";
        track( msg );
        SAGOutboundeWay_1.getServices().getInterActClient().snfAcquire();
        SAGOutboundeWay_1.getServices().getInterActClient().snfRelease();
        track( " OK" );
      msg = "\r\n\r\n" + ++transactionSeq + ".  Scenario 8: SAG_Cert: InterAct/NotifQueue/Delivery
...";
        track( msg );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 8.1: SAG_Cert: IA/Delivery/1st ...";
        track( msg );
        SAGOutboundeWay_1.getConfiguration().getInterActClient().setSwInt_Service(
"swift.generic.iast!x" );
        SAGOutboundeWay_1.getConfiguration().getInterActClient().setSwInt_Requestor(
"ou=management,o=ptsauszz,o=swift" );
        SAGOutboundeWay_1.getConfiguration().getInterActClient().setSwInt_Responder(
"ou=management,o=ptsauszz,o=swift" );
        SAGOutboundeWay_1.getConfiguration().getInterActClient().setSwInt_NotifQueue(
"ptsauszz_generic!x" );
        SAGOutboundeWay_1.getServices().getInterActClient().setRequestPayload( msg );
        SAGOutboundeWay_1.getServices().getInterActClient().sendAndReceive();
        track( " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 8.2: SAG_Cert: IA/Delivery/2nd ...";
        track( msg );
        SAGOutboundeWay_1.getServices().getInterActClient().setRequestPayload( msg );
        SAGOutboundeWay_1.getServices().getInterActClient().sendAndReceive();
        track( " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 8.3: SAG_Cert: IA/Delivery/Signed/3rd
...";
        track( msg );
        SAGOutboundeWay_1.getConfiguration().getInterActClient().setSwInt_RequestCrypto(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getTRUE() );
        SAGOutboundeWay_1.getServices().getInterActClient().setRequestPayload( msg );
        SAGOutboundeWay_1.getServices().getInterActClient().sendAndReceive();
        SAGOutboundeWay_1.getConfiguration().getInterActClient().setSwInt_RequestCrypto(
SAGOutboundeWay_1.getConstants().getSw_Boolean().getFALSE() );

SAGOutboundeWay_1.getPrimitives().getSnlPrimitives().getSwInt_ExchangeRequest().getSwInt_Request()
.removeSwSec_Crypto();
        track( " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".  Scenario 9: SAG_Cert: PullQueue ...";
        track( msg );
        msg = "\r\n\r\n       Wait a while for SwiftNet SnF Queue access...";
        track( msg );
        Thread.sleep( 10000 );
        track( " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 9.1: SAG_Cert: PullQueue/1st ...";
        track( msg );
        SAGOutboundeWay_1.getServices().getInterActClient().snfAcquire();
        SAGOutboundeWay_1.getServices().getInterActClient().snfPull();
        track( " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 9.2: SAG_Cert: PullQueue/2nd ...";
        track( msg );
        SAGOutboundeWay_1.getServices().getInterActClient().snfPull();
        SAGOutboundeWay_1.getServices().getInterActClient().snfAck();
        SAGOutboundeWay_1.getServices().getInterActClient().snfRelease();
        track( " OK" );
        msg = "\r\n\r\n" + ++transactionSeq + ".    Scenario 9.3: SAG_Cert: PullQueue/3rd ...";
        track( msg );
        SAGOutboundeWay_1.getServices().getInterActClient().snfAcquire();
        SAGOutboundeWay_1.getServices().getInterActClient().snfPull();
        SAGOutboundeWay_1.getServices().getInterActClient().snfAck();
        SAGOutboundeWay_1.getServices().getInterActClient().snfRelease();
        track( " OK" );
        String to = "";
        msg = "\r\n\r\n" + ++transactionSeq + ".  Scenario 10: SAG_Cert: Show more usages of other
API/Primitive (example: Process Sag:Primitive ReadEventLog: starting time {" + startTime + "}) ...
";
        track( msg );
        String readLog = "<Sag:PrimitiveRequest>" + "<Sag:PrimitiveName>ReadEventLog</
Sag:PrimitiveName>" + "<Sag:PrimitiveRelease>SAG 4.0</Sag:PrimitiveRelease>" +
"<Sag:ReadEventLogRequest>" + "<Sag:EventLogSelect>" + "<Sag:FromLogSysTime>" + from + "</
Sag:FromLogSysTime>" + "<Sag:FromLogSequence></Sag:FromLogSequence>" + "<Sag:ToLogSysTime>" + to +
"</Sag:ToLogSysTime>" + "<Sag:ToLogSequence></Sag:ToLogSequence>" + "<Sag:LogCorrelationId></
Sag:LogCorrelationId>" + "<Sag:EventSeverity></Sag:EventSeverity>" + "<Sag:EventClass></
Sag:EventClass>" + "</Sag:EventLogSelect>" + "<Sag:StartEventLogRecord>" + "<Sag:LogSequence></
```

```
Sag:LogSequence>" + "</Sag:StartEventLogRecord>" + "<Sag:NumberOfRecord></Sag:NumberOfRecord>" +
"</Sag:ReadEventLogRequest>" + "</Sag:PrimitiveRequest>";
        SAGOutboundeWay_1.getRemoteApis().getClient().getRequest().getEnvelope().setMsgFormat(
SAGOutboundeWay_1.getConstants().getEnvelopeMsgFormat().getSagPrimitive() );
        SAGOutboundeWay_1.getRemoteApis().getClient().getRequest().setLetter( readLog );
        SAGOutboundeWay_1.getRemoteApis().getClient().call();
        track( " OK" );
        String log = SAGOutboundeWay_1.getRemoteApis().getClient().getResponse().getLetter();
        int idx = log.length() > 8192 ? log.length() - 8192 : 0;
        msg = "\r\n\r\n                 Got Event Log for this set of tests\r\n [...... " +
log.substring( idx );
        track( msg );
        msg = "\r\n\r\n\r\n  Total number of request/response test interactions ... " +
transactionSeq;
        track( msg );
        track( "\r\n\r\n\r\nEnd of collaboration from SAG_Cert\r\n\r\n\r\n\r\n" );
        track( "\r\n************************************************" );
        track( "\r\n*    End of remote test for SAG Certification   *" );
        track( "\r\n************************************************\r\n\r\n\r\n\r\n" );
    }

    private void track( String msg )
        throws Exception
    {
        logger.info( msg );
        msg = msg + " at time {" + new java.util.Date() + "}";
        file.setText( msg );
        ;
        file.write();
    }

}
```

Index

M

N

O

P

Q

R

S

T

U

W