SeeBeyond ICAN Suite

SWIFT ADK eWay Intelligent Adapter User's Guide

Release 5.0

Version 20040806171856.

Contents

Chapter 1

Introducing the SWIFT ADK eWay

This guide explains how to install, use, and operate the SeeBeyond® Integrated Composite Application Network Suite™ (ICAN) SWIFT ADK eWay Intelligent Adapter, referred to as the SWIFT ADK eWay throughout this guide.

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

Chapter Topics

1.1 SWIFT Overview

This section provides an overview of Society for Worldwide Interbank Financial Telecommunication (SWIFT) and the SWIFT ADK eWay.

Introduction to SWIFT

SWIFT is a bank-owned cooperative which supplies secure payment event transfer, matching, and other services to owner/member banks and other financial organizations (including brokers, securities deposit and clearing organizations, and stock exchanges) via its SWIFT Transport Network (STN). The types of events processed by SWIFT include:

- **Payments**: Clearing and settlements between member banks.
- **Securities**: Clearing and settlements and cross border electronic trade confirmations.
- **Forex, money markets and derivatives**: Confirmation of trades, marketing and reporting facilities.
- **Trade finance**: Documenting credits and collections.

The SWIFT ADK eWay provides secure messaging services (both receiving and transmitting) between SWIFT financial institutions. The SWIFT ADK eWay is designed specifically to interface with the SWIFTAlliance, and enables the SeeBeyond eGate and eInsight system to exchange data with SWIFTAlliance by providing:

- **Automated integration** of securities events in the new securities standards (events MTxxx) which is based on the ISO15022 Data Dictionary.
- **Translation** of incoming events received from SWIFT into the format required by existing applications.
- **Security**, by enabling Secure Sockets Layer (SSL) between the SEWS executable and eGate.

The SWIFT ADK eWay uses the SWIFTAlliance Developer Toolkit (ADK), which is a library of APIs that can call services provided by SWIFTAlliance servers. For more information about ADK, see the *SWIFT ADK Reference Guide*.

1.2 SWIFT ADK eWay Overview

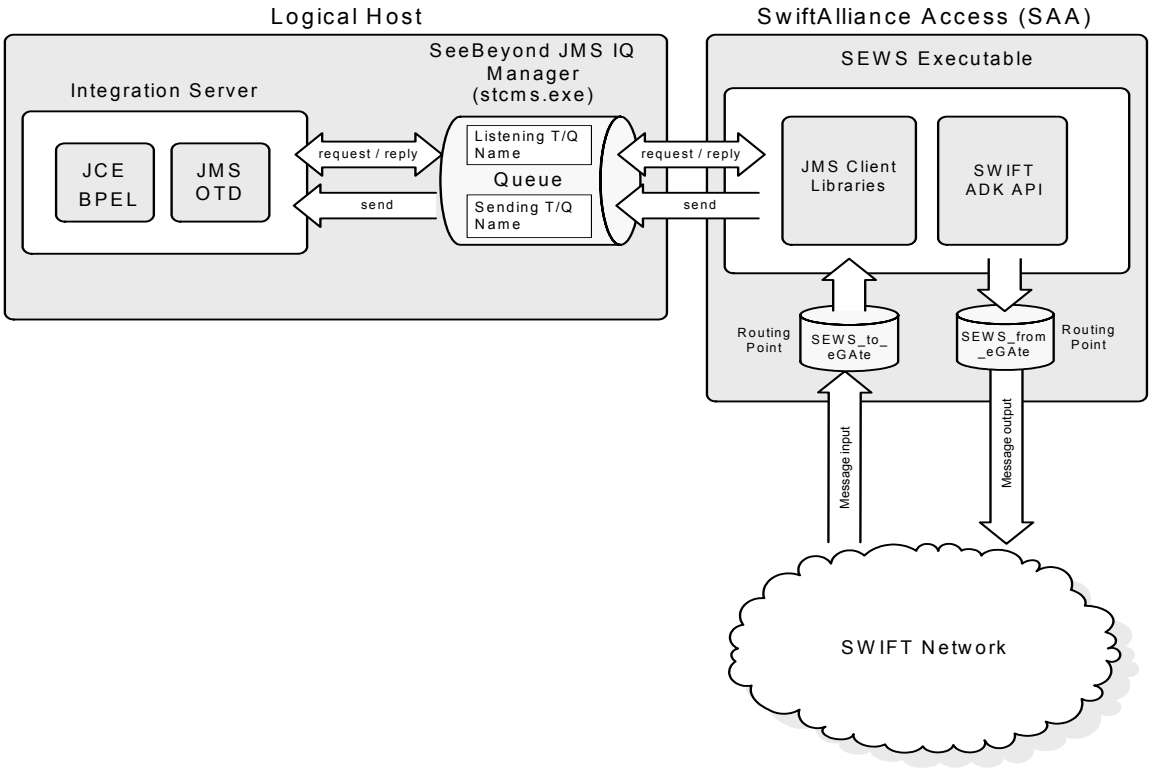
The SWIFT ADK eWay functions as a component within the SWIFTAlliance Access (SAA), operating in conjunction with the SWIFT ADK API to provide direct access to messages that pass between the SeeBeyond JMS IQ Manager and the SWIFT Network.

The SEWS Executable (SEWS.exe) component is linked into SAA through two routing points, one for incoming messages (SEWS_from_egate) and the other for outgoing messages (SEWS_to_egate).

The message remains in the routing point until the SEWS component forwards it to the JMS IQ Manager, or SAA delivers it to the network. The connection between eGate/eInsight and SWIFT can be monitored in the SWIFTAlliance log.

Both the eGate API Kit and the Swift ADK API transfer message data from the SAA to a Queue managed by the SeeBeyond JMS IQ Manager within the SeeBeyond Logical Host. JMS messages stemming from the JMS IQ Manager are either consumed or delivered transactionally. These transmissions occur between the JMS OTD, which resides in the Integration Server, and the Java Collaboration Definitions or BPEL business processes, which functions to transform inbound and outbound message data. The SWIFT ADK eWay supports both synchronous (request/reply) and asynchronous (send only) communication between these components.

Figure 1 SWIFT ADK eWay Message Flow Overview



1.2.1 Types of SWIFT ADK eWay Message Flows

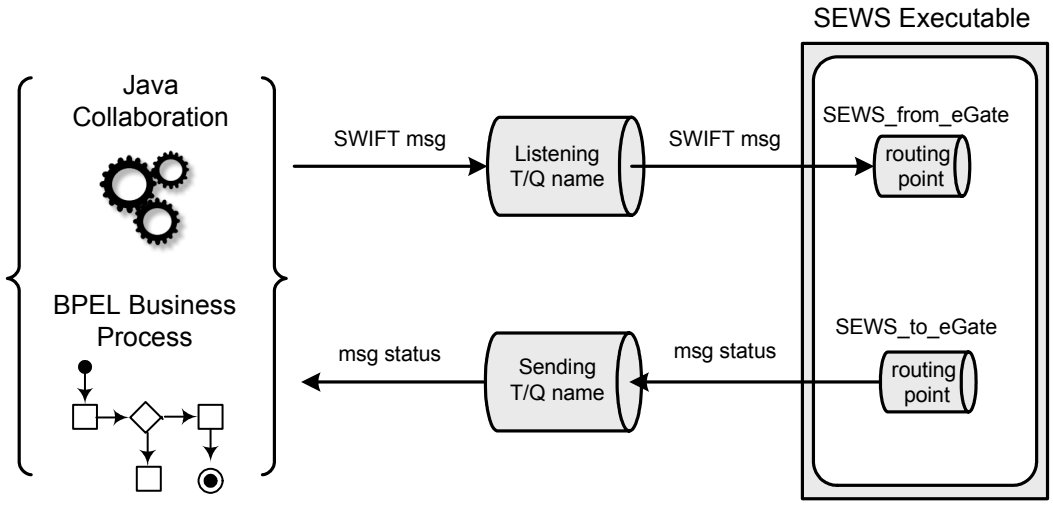
Message that pass between SEWS and eGate/eInsight are classified within the following three message flows:

- Publish (outbound asynchronous)
- Request/Reply (outbound synchronous)
- Subscribe (inbound asynchronous)

Publish (outbound asynchronous)

Outbound messages that are published asynchronously to SEWS are sent to the **SEWS_from_eGate** routing point without waiting to know the status or result of the request. We call the *send* method on the JMS OTD within a Java Collaboration or BPEL Business Process. This method does not return any value. The JMS message is sent to the JMS queue/topic (as defined by the Listening T/Q name). The same message is then retrieved by SEWS. Based on the message's "Header" property value, the SEWS component will either "PUT" or "LIST". The status/result of these actions can not be returned to the Java Collaboration because the JMS OTD *send* method is asynchronous. The result of "**PUT**" or "**LIST**" enters eGate through a JMS queue/topic (as defined by the Sending T/Q name). See Figure 2 for an illustration of the message transmission flow.

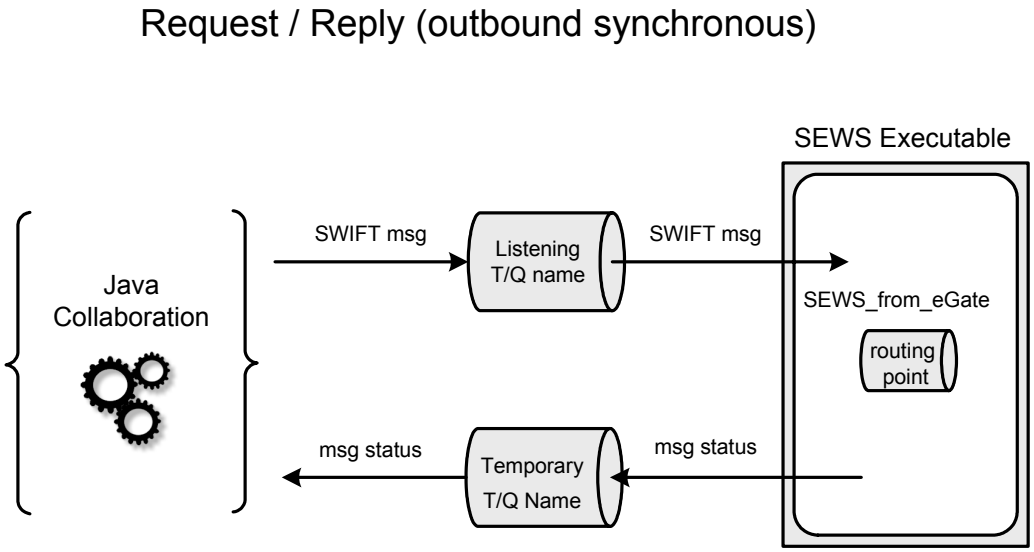
Figure 2 Publish (outbound asynchronous) Message Transmission Method



Request / Reply (outbound synchronous)

When publishing outbound messages synchronously to SEWS, the message is sent to the **SEWS_from_eGate** routing point, then the Java Collaboration waits for the status or result of the request before sending another request. The synchronous nature is achieved by calling the JMS OTD method *requestReply*. This method is different from *send* in terms of its blocking/waiting. By calling this method in a Java Collaboration, a JMS message is sent to the JMS queue/topic (as defined by the Listening T/Q name). Also, the JMS OTD creates a hidden temporary JMS queue/topic for SEWS to reply to. Blocking/waiting happens when the JMS OTD becomes an immediate subscriber of the temporary queue/topic. In parallel, the SEWS component retrieves the message from the JMS queue/topic. SEW then processes the message by action defined in the JMS "Header" property. The result of the "PUT/LIST" action is sent as a JMS message by SEWS on the hidden temporary JMS queue/topic. The *requestReply* call unblocks and returns the resulting JMS message to a Java Collaboration. See Figure 3 for an illustration of the message transmission flow.

Figure 3 Request / Reply (outbound synchronous) Message Transmission Method



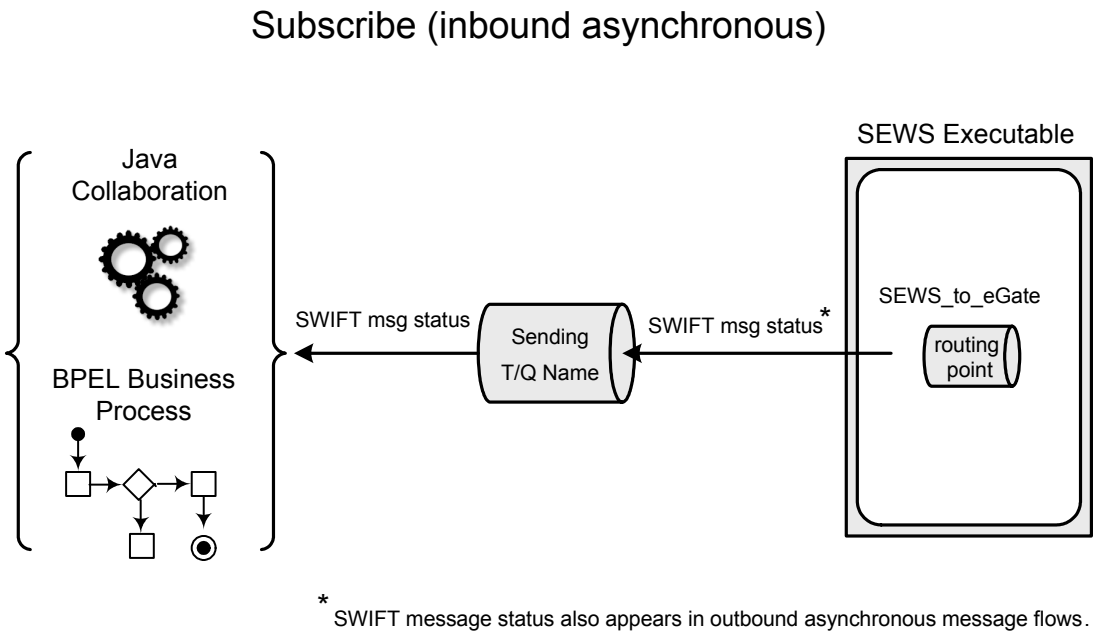
Subscribe (inbound asynchronous)

Messages from SEWS always arrive at eGate/eInsight asynchronously on the JMS queue/topic (as defined by the Sending T/Q name). There can be three types of messages:

- The real SWIFT message from the SEWS_to_eGate routing point. This message leaves the routing point and enters eGate/eInsight as a JMS text message.
- The status message, if you publish to SEWS asynchronously.
- A heart beat message. This message implies that there is no available messages from the SEWS_to_eGate routing point.

See Figure 4 for an illustration of the message transmission flow.

Figure 4 Subscribe (inbound asynchronous) Message Transmission Method



Chapter 2

Installing the SWIFT ADK eWay

This chapter describes the requirements and procedures for installing the SWIFT ADK eWay. Procedures for implementing sample projects, are described in **Chapter 6**.

Chapter Topics

2.1 Supported Operating Systems

The SWIFT ADK eWay is available for the following operating systems:

- Windows 2000
- IBM AIX 5.2
- Sun Solaris 9

2.2 System Requirements

To use the SWIFT ADK eWay, you need the following:

- An eGate Logical Host.
- A TCP/IP network connection to SWIFTAlliance.
- Sufficient free disk space to accommodate eWay files:
  - Approximately 200 KB on Windows systems
  - Approximately 820 KB on Solaris system
  - Approximately 500 KB on AIX systems

- Additional free disk space on the SWIFTAlliance host for the SEWS component (see additional information under **External System Requirements** on page 13):
  - Approximately 95 KB of disk space on Windows systems
  - Approximately 1.7 MB of disk space on Solaris systems
  - Approximately 780 KB of disk space on AIX systems

Note: *Additional disk space is required to process and queue the data that this eWay processes; the amount necessary varies, based on the type and size of the data being processed, and any external applications performing the processing.*

Logical Host Requirements

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

Environment Configuration

No changes are required to the Logical Host's operating environment to support this eWay.

2.3 External System Requirements

- The SWIFT ADK eWay supports SWIFTAlliance version 5.5. The SWIFT ADK eWay also requires a SWIFT ADK eWay run-time license for SWIFTAlliance version 5.5.

Note: *The ADK (Alliance Developer Kit) API protocol is supported only by the SWIFTAlliance Access product family. It is* ***not*** *supported by SWIFTAlliance Entry. Customers using SWIFTAlliance Entry can send and receive SWIFT messages through the SeeBeyond Batch eWay, by appropriately configuring the AFT (Automated File Transfer) interface in SWIFTAlliance Entry.*

External Configuration Requirements

- The SEWS component must be installed and configured (see **Chapter 4**)

Note: *The SEWS component must be installed on the same platform as the SWIFTAlliance server.*

- Before installing SEWS into SWIFTAlliance on UNIX, the root user must set up the correct environment.

Two routing points, **SEWS_to_egate** and **SEWS_from_egate** are installed when the **SEWS ADK** component is installed. The ways in which these routing points are used, and messages are routed to and from them, are independent of SEWS and depend on the application being used. For information on how to configure routing points, see the SWIFT *System Management Guide*.

2.4 Before Installing the eWay

Open and review the **Readme.txt** file for any additional information or requirements prior to installation. The readme.txt is located on the installation CD-ROM.

2.5 eWay Installation Procedures

During the ICAN Suite installation process, the Enterprise Manager, a Web-based application, is used to select and upload eWay and add-on files (**.sar** files) from the ICAN installation CD-ROM to the Repository.

The eGate installation process includes the following operations:

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

Follow the instructions for installing the ICAN Suite found in the "*SeeBeyond ICAN Suite Installation Guide*", and include the following steps:

1. On the Enterprise Manager under the **ADMIN** tab, select the **SwiftADKeWay.sar** (to install the Swift ADK eWay) file to upload.
2. On the Enterprise Manager under the **Admin** tab, select the **FileeWay.sar** (to install the File eWay) file to upload.
3. On the Enterprise Manager under the Admin tab, select the **SwiftADKeWayDocs.sar** (to install the documentation and sample projects) file to upload.
4. On the Enterprise Manager under the **DOWNLOADS** tab, click the link for the SEWS component that matches the system running SAA. Files available for download include:
   - **SEWS component for SAA5.5 Win32** – downloads the SEWS_Win32.zip file
   - **SEWS component for SAA5.5 Aix52** – downloads the SEWS_AIX52.tar file
   - **SEWS component for SAA5.5 Solaris 9** – downloads the SEWS_SOL9.tar file

   You must save the file the following local directory folder:

   ```
   <cd>:\SETUP\ADDONS\EWSWIFTADK\SEWS\WIN32\
   ```

   These are SWIFT ADK medium files that are accessed directly from the SAA install GUI, (see **Installing and Setting up the SEWS eWay Component** on page 20).
5. On the Enterprise Manager under the **DOCUMENTATION** tab, click the sample project link. For the sample project, it is recommended that you extract the file to

another location prior to importing it using the Enterprise Explorer's Import Project tool.

For additional information on how to use eGate, please see the "*eGate Integrator Tutorial*".

2.6 After Installation

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

Chapter 3

Configuring the SWIFT ADK eWay

This chapter explains how to set the properties for the SWIFT ADK eWay.

Chapter Topics

3.1 SWIFT ADK eWay Configuration

The configuration settings within eGate for the SWIFT ADK eWay only apply to the SeeBeyond JMS IQ Manager. The SWIFT ADK eWay contains no eWay configuration properties since the SEWS component resides as a component within the SWIFTAlliance Access, which communicates directly with Topics of Queues managed by the SeeBeyond JMS IQ Manager.

To configure the SWIFT ADK eWay

1. From the Enterprise Designer, click the **Environment Explorer** tab located at the bottom left pane.
2. Create a new project **Environment**.
3. Create all the systems required to deploy your project, including:
   - External systems
   - Logical Host
   - SeeBeyond Integration Server
   - SeeBeyond JMS IQ Manager
4. Right-click **JMS IQ Manager** and select **Properties** from the list box.
5. The eWay **Properties dialog box** appears, see **Figure 5 on page 17**. You can use this dialog box to modify the **SeeBeyond JMS IQ Manager** properties necessary for communication with the **SEWS Executable** in the **SWIFTAlliance Access**.
6. Click **OK** then **Save All** to save your changes.

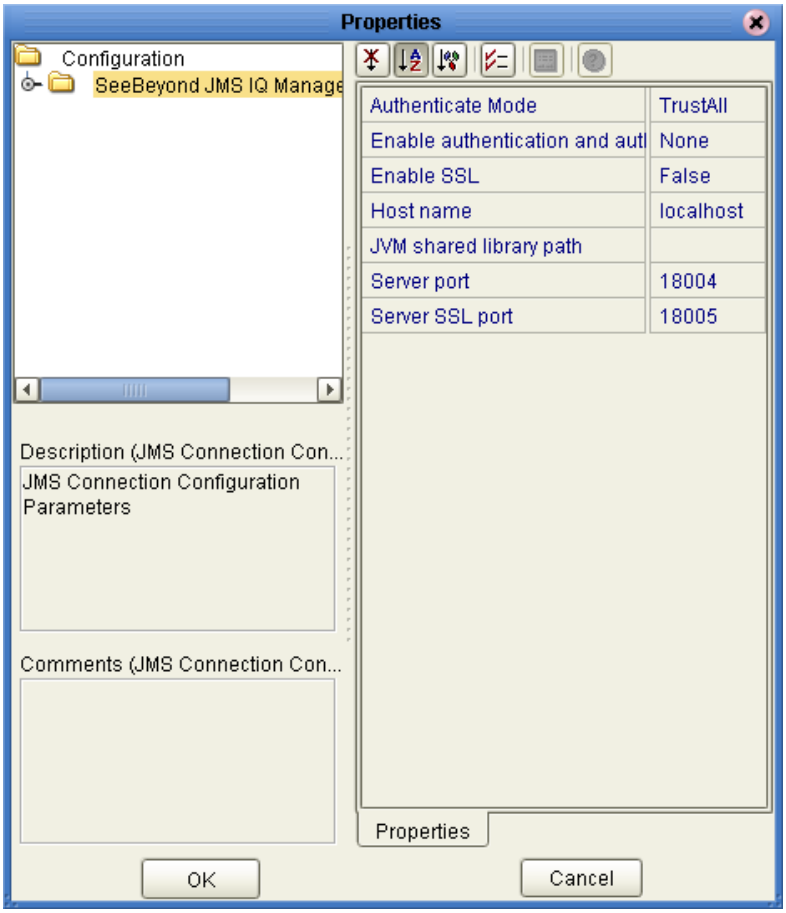
3.2 Configuring the SeeBeyond JMS IQ Manager

This chapter describes how to configure the SeeBeyond JMS IQ Manager properties from the Environment Explorer tab.

Properties you can configure on the dialog box include:

- Enable authentication and authorization
- Enable SSL
- Host Name
- Server Port
- Server SSL Port

Figure 5 SeeBeyond JMS IQ Manager Properties Dialog Box



Enable authentication and authorization

Description

Turns the authentication and authorization security service on or off. If **File** is selected, then the Sun Java System, AD and OpenLDAP access to the SeeBeyond JMS IQ Manager is only granted if a valid user id and password are supplied. If **File** is selected for File-based aa service, **SunJava** is selected for Sun Java System, **AD** is selected for Active Directory Service, **OpenLDAP** is selected for OpenLDAP Directory Service.

Required Values

Change the default value to **File**.

Note: *Previous versions of eGate use the value **True** to require user id and password access through the SeeBeyond JMS IQ Manager. You can access the JMS configuration properties by double-clicking the JMS icon on the connectivity map. Security settings are found under: **Configuration** > **JMS Client** > **Basic** > **Security**. For additional information on these settings, refer to the eGate Integrator JMS Reference Guide.*

Enable SSL

Description

Turns on or off Secure Sockets Layer (SSL). TCP/IP connections from the client to the JMS IQ Manager are secured using SSL when this parameter is set to **True**, but you must use the Server SSL port to take advantage of it.

Required Values

The default value is **False**.

Host Name

Description

Lists the host name of the machine running the SeeBeyond JMS IQ Manager.

Required Values

The default value is **localhost**.

Server Port

Description

Specifies the TCP/IP port number that the SeeBeyond JMS IQ Manager listens on. Each SeeBeyond JMS IQ Manager must have a unique port number per Logical Host. Change this setting to an available port if you add additional SeeBeyond JMS IQ Managers.

Required Values

The value displayed in the Server Port field is different for each version of eGate installed.

Server SSL Port

Description

Specifies the TCP/IP port number that the SeeBeyond JMS IQ Manager listens on for Secure Socket Listener (SSL) connections. This port number is enabled when you change the Enable SSL configuration property to "**True**".

Required Values

The value displayed in the Server SSL Port field is different for each version of eGate installed.

Chapter 4

Installing and Setting up the SEWS eWay Component

This chapter describes procedures for installing the SEWS component within the SWIFTAlliance Access, and for customizing the SEWS component to operate with your system.

This Chapter Includes:

4.1 Overview

Prior to installation you must:

- Install the SEWS component into the SWIFTAlliance server
- Make sure all SWIFTAlliance servers are not running
- Click **Cancel** if the Add-ons installer starts up automatically

You must properly configure the SEWS component to connect SWIFTAlliance to the SeeBeyond JMS IQ Manager.

4.2 Installing the SEWS Component on Windows

To install the SEWS component on a Windows system

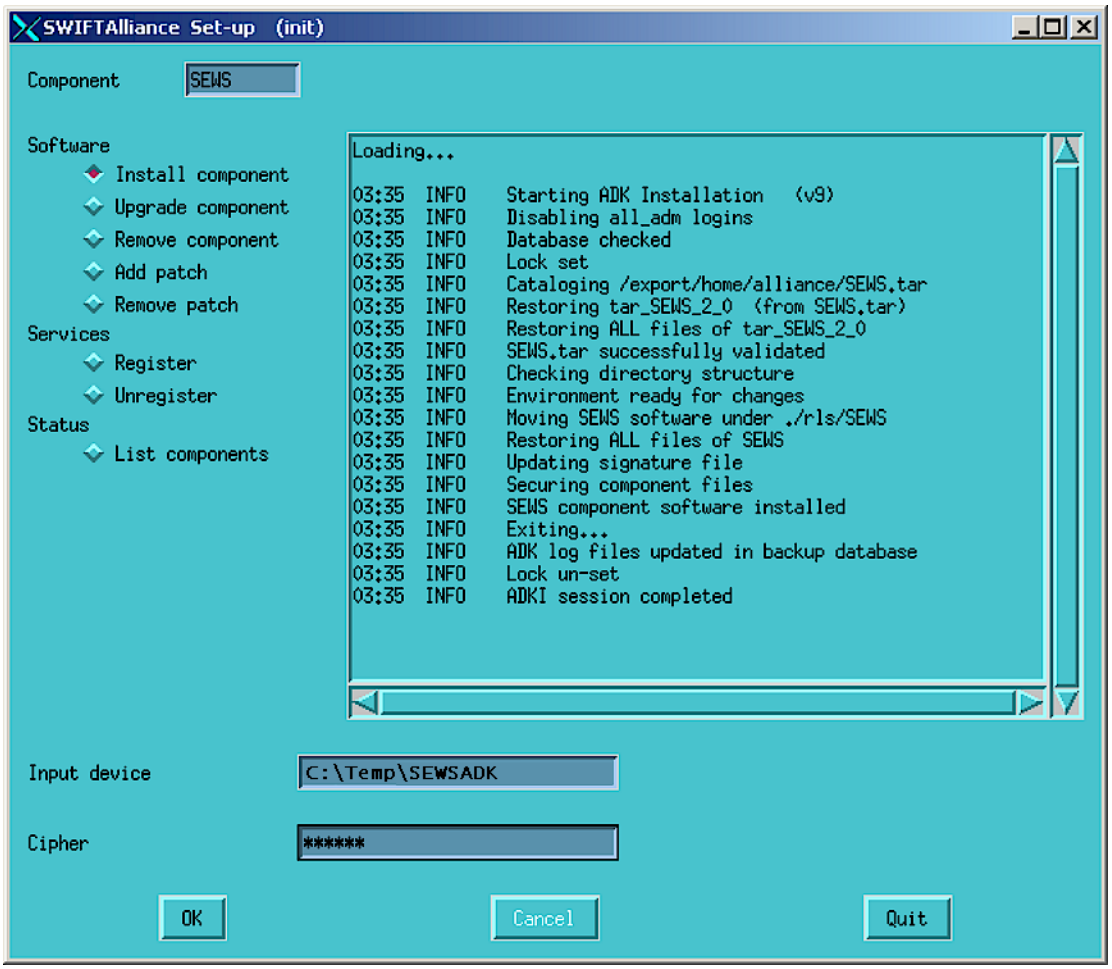
1 Unzip the SEWS component for the SAA5.5_Win32 to a local folder:

```
<cd>:\temp
```

2 Log into SWIFTAlliance ADK as the user account under which it was installed.

3 Start the SWIFTAlliance ADK setup program by locating and running the **adk_install.exe** on SAA 5.5. The SWIFTAlliance Set-up window appears.

Figure 6 SWIFTAlliance Set-up (Install component)



4 In the Set-up dialog, install SEWS as follows:

A In the Component field, type SEWS.

B In the Software group, select the **Install component** option (or **Upgrade component**, if a previous version of SEWS is installed)

C In the **Input device** field, type in the path to the folder extracted in step 1:

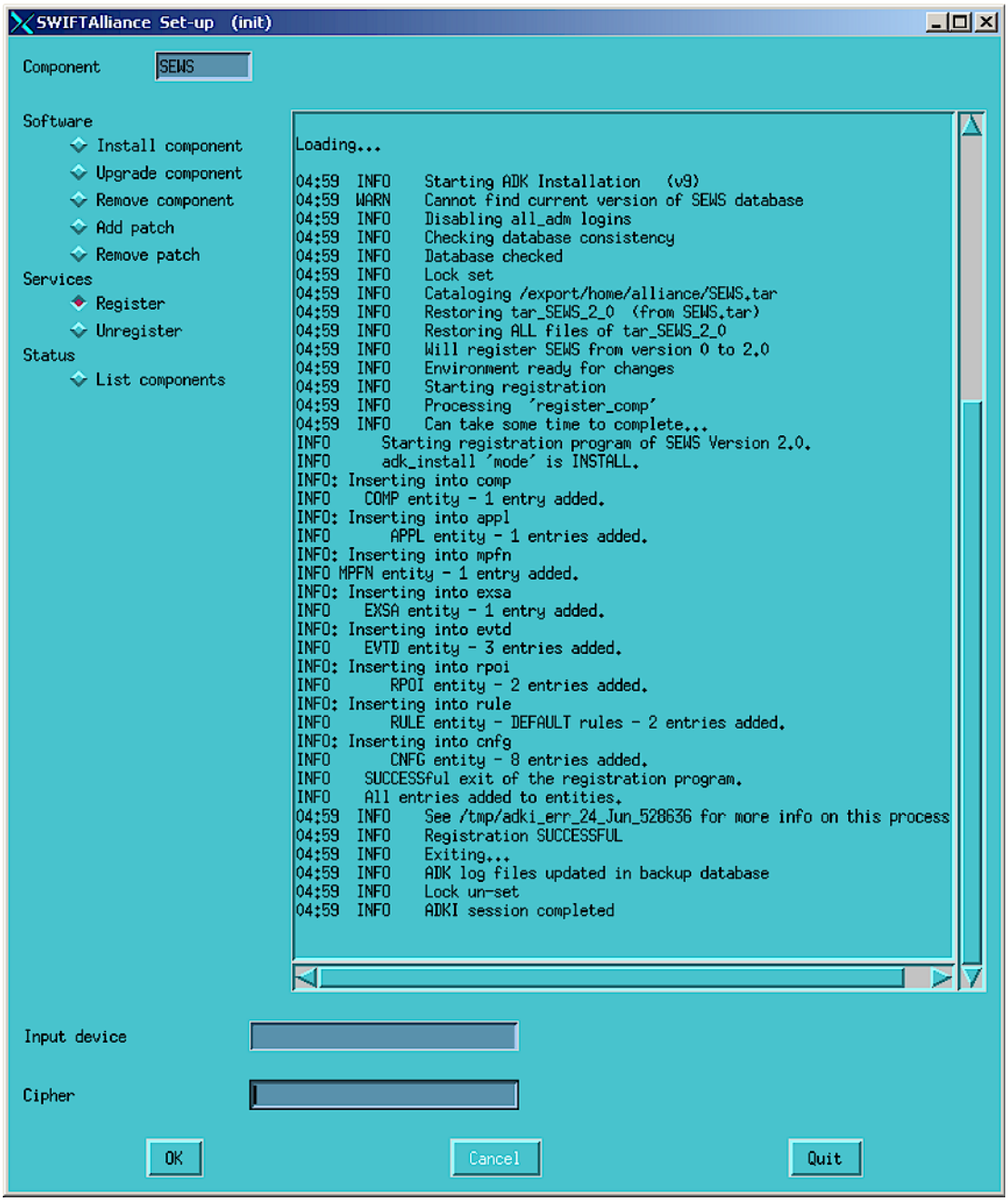
```
<drive>\<temp directory>
```

D In the Cipher field, enter "**seebeyond**"

E Click **OK**. Progress messages display in the dialog box. SEWS is successfully installed when you see the following message:

```
ADKI session completed
```

Figure 7 SWIFTAlliance Set-up (Register)



5 If you are preforming a first-time installation, register SEWS as follows:

A In the **Set-up** dialog **Software** group, clear the **Install component** option by selecting **Register**.

B Clear the **Input device** and **Cipher** fields.

C Click **OK**. Progress messages are displayed in the message area of the dialog. SEWS is successfully registered when you see the following message:

```
ADKI session completed
```

D Click **Quit** to close the Set-up window.

4.3 Installing the SEWS Component on UNIX

To install the SEWS component on a UNIX system

1 Log in as root, running under /bin/ksh.

2 Save the SEWS component for **SAA5.5_AIX52** or **SAA5.5_Solaris_9** to:

```
/export/home/alliance/
```

3 Set up the correct environment by sourcing the following script (including the period and space at the beginning):

```
. /usr/swa/alliance_init -s
```

4 Set up the **LIBPATH** (for AIX 5.2) or the **LD_LIBRARY_PATH** (for Solaris 9) to:

```
LIBPATH=/usr/alliance/rls/common/lib/AIX

LD_LIBRARY_PATH=/usr/alliance/rls/common/lib/SunOS
```

5 Set the display variable.

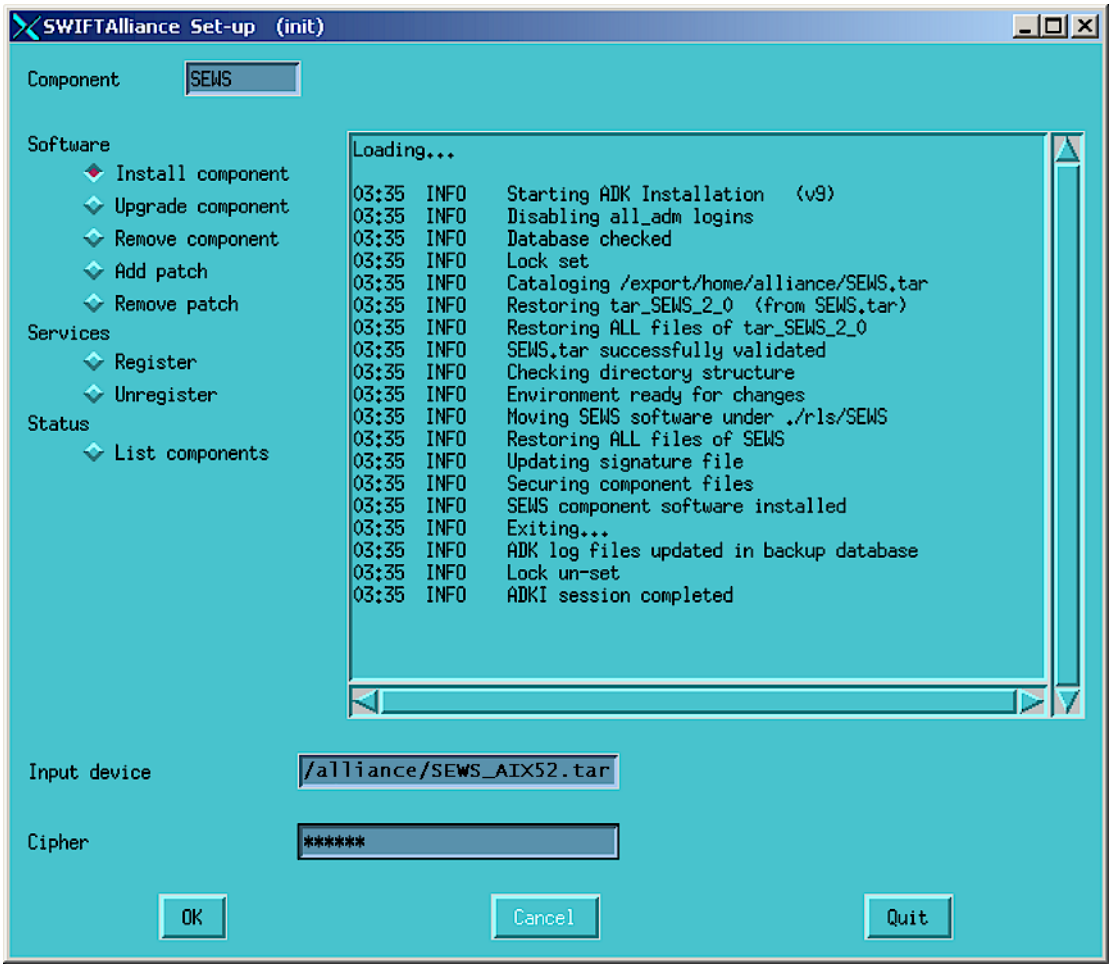
6 Start the SWIFTAlliance ADK setup program by typing the following command:

```
$ (ALLIANCE) /INA/bin/$(ARCH)/adk_install:
```

Note: *On Solaris,* **$(ARCH)** *is set to* **'SunOS'***; on AIX 5.2,* **$(ARCH)** *is set to* **'AIX'***.*

The **SWIFTAlliance Set-up** window appears.

Figure 8 SWIFTAlliance Set-up (Install component)



7 In the Set-up dialog, install SEWS as follows:

A In the **Component** field, type SEWS.

B In the **Software** group, select the **Install component** option (or **Upgrade component**, if a previous version of SEWS is installed).

C In the **Input device** field, type in the fully-qualified path on the installation CD-ROM, such as:

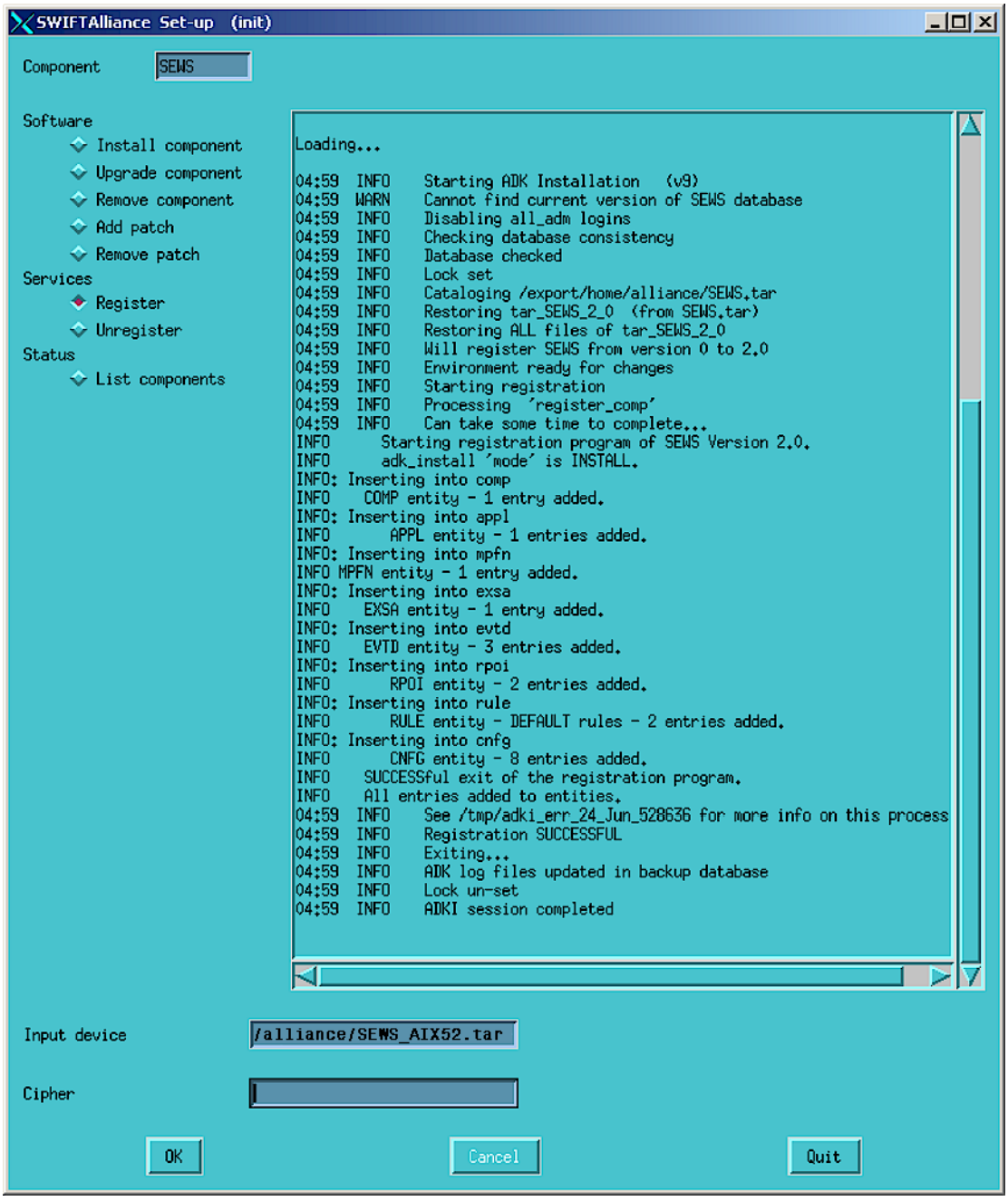
```
<cd>:/export/home/alliance/SEWS_AIX52.tar
```

D In the Cipher field, enter "**seebeyond**"

E Click **OK**. Progress messages display in the dialog box. SEWS is successfully installed when you see the following message:

```
ADKI session completed
```

Figure 9 SWIFTAlliance Set-up (Register)



8 Register SEWS as follows if you are performing a first-time installation:

A In the **Set-up** dialog **Software** group, clear the **Install component** option.

B In the **Services** group, select **Register**.

C Clear the **Input device** and **Cipher** fields.

D Click **OK**. Progress messages are displayed in the message area of the dialog. SEWS is successfully registered when you see the following message:

```
ADKI session completed
```

E Click **Quit** to close the Set-up window.

4.4 Configuring SEWS

To configure SEWS, first set your SWIFTAlliance servers to **housekeeping** mode, then set up the following item to match your system and your schema:

- Network Parameters

Note: *The values entered for the SEWS parameters must exactly match the corresponding eWay configuration parameters.*

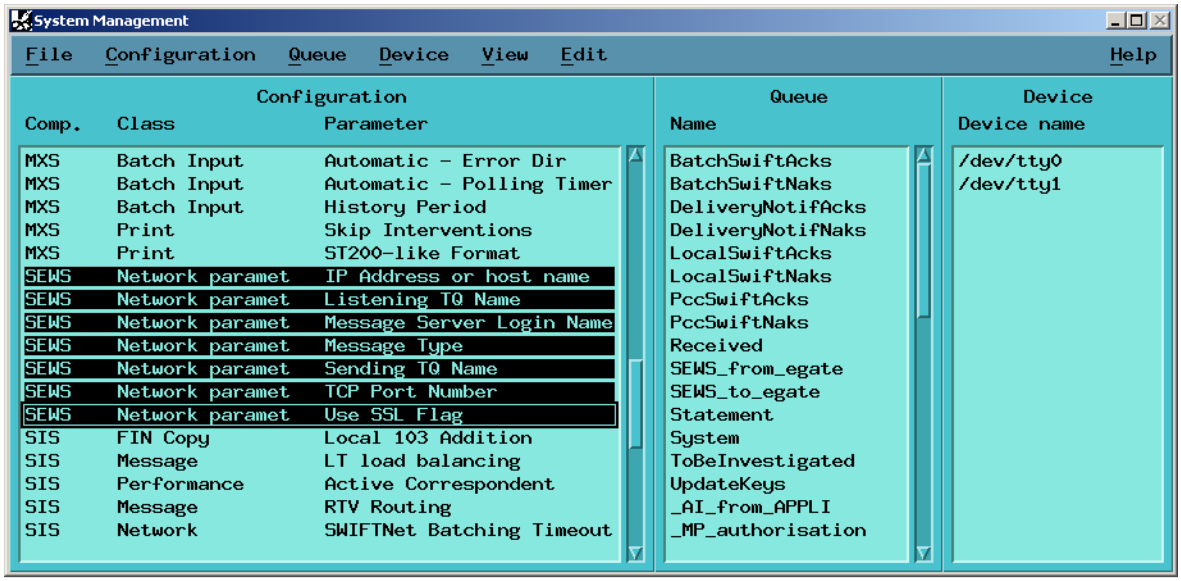
4.4.1 System Management Window

The System Management window lists the seven parameters that require configuration. Each parameter must contain the proper value or the SEWS executable will attempt to initiate a JMS connection to a nonexistent JMS server.

To open the System Management window:

1 Click the **System Management** icon on the **SWIFTAlliance Access Control** window. The System Management window appears.

Figure 10 Systems Management Window



4.4.2 Network Parameters

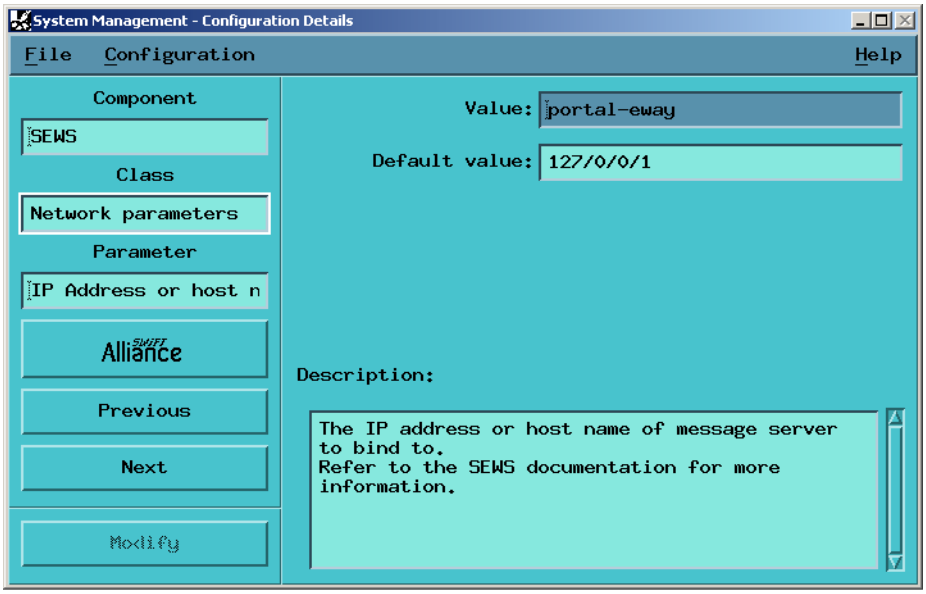
The Systems Management window of SWIFTAlliance allows you to configure seven *non-secret* parameters (see **Figure 10 on page 26**) of the SEWS component, including:

- **IP Address or host name**
- **Listening TQ Name**
- **Message Server Login Name**
- **Message Type**
- **Sending TQ name**
- **TCP Port Number**
- **Use SSL Flag**

IP Address or host name

1 Double-click the **IP Address or host name** line in the System Management Window, to open the configuration details for that parameter, (see Figure 11 below).

Figure 11 Systems Management – IP Address



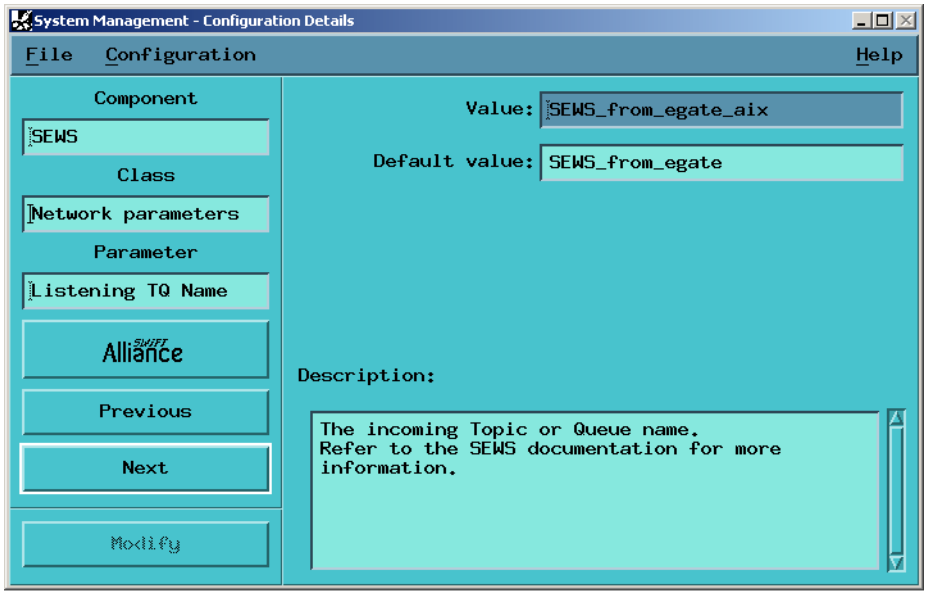
2 Enter the IP Address or host name where your message server runs.

Note: *Review the SWIFT documentation for the required IP Address format.*

Listening TQ Name

3 Double-click the **Listening TQ Name** line to open the configuration details for that parameter, (see Figure 12 below).

Figure 12 Systems Management – Listening TQ Name



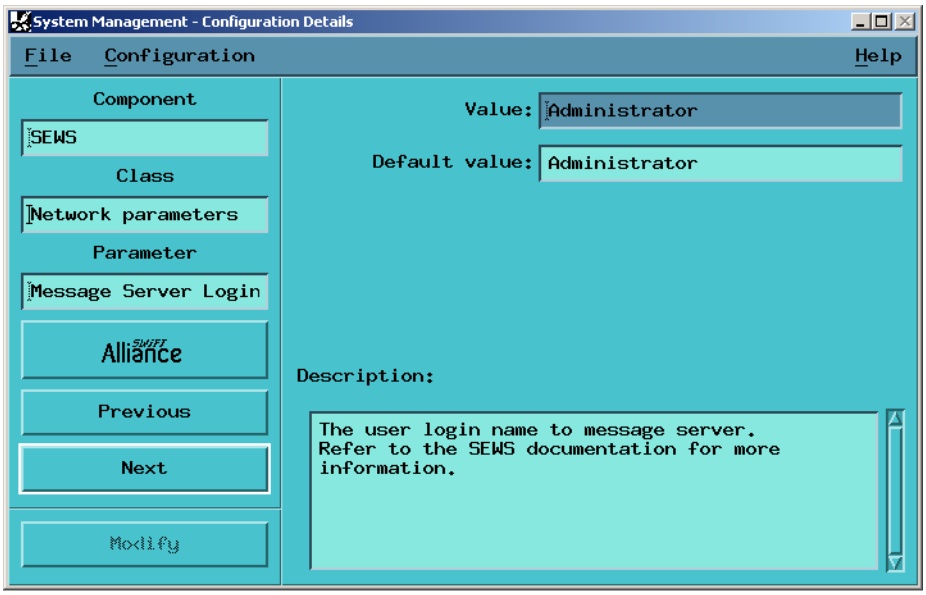
4 Enter the name of the JMS Topic or Queue in the **SeeBeyond Enterprise Designer – Connectivity Map Editor** that is designated for sending messages to SEWS.

The SEWS executable listens for available messages and inserts the SWIFT message to the **SEWS_from_egate** routing point. You can provide any name for the JMS queue or topic in eGate, but not for the routing point which is registered with SWIFT. We recommend providing similar naming conventions to distinguish between incoming and outgoing messages.

Message Server Login

5 Double-click the **Message Server Login** line in the System Management Window, to open the configuration details for that parameter, (see Figure 13 below).

Figure 13 Systems Management – Message Server Login



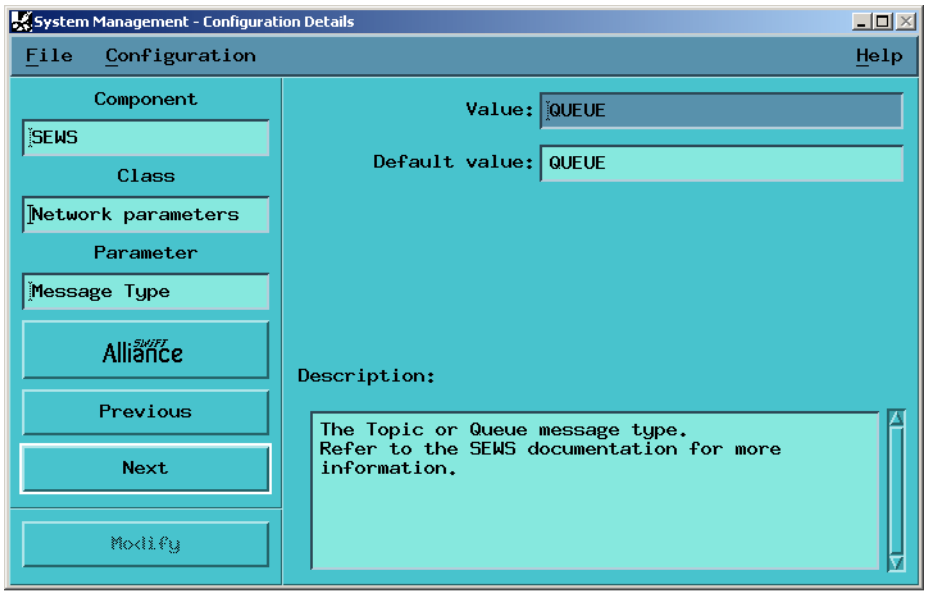
6 Enter the **Message Server Login** name to communicate with the JMS server.

This parameter only takes affect if you enable the JMS message server to authenticate the client. The example in Figure 13 uses the default value.

Message Type

7 Double-click the **Message Type** line in the System Management Window, to open the configuration details for that parameter, (see Figure 13 below).

Figure 14 Systems Management – Message Type

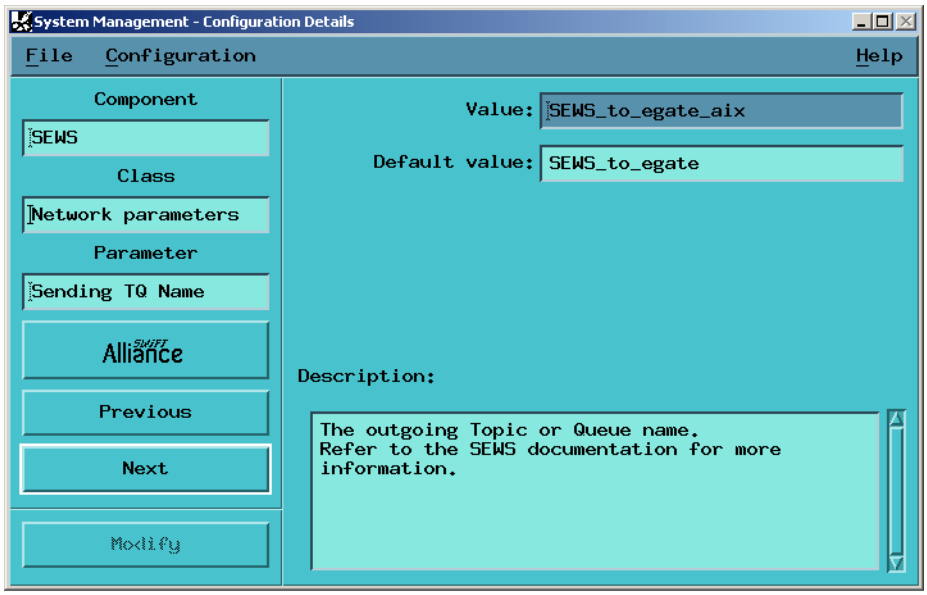


8 Enter **QUEUE** or **TOPIC** in upper case characters. Entering any other character causes the value to default to queue mode communication with the JMS message server.

Sending TQ Name

9 Double-click the **Sending TQ Name** line in the System Management Window, to open the configuration details for that parameter, (see Figure 13 below).

Figure 15 Systems Management – Sending TQ Name



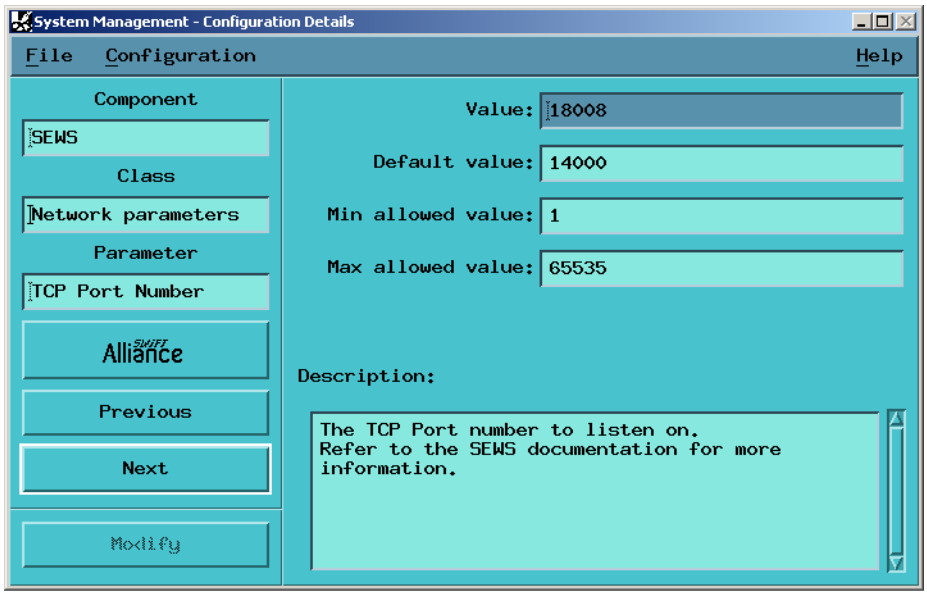
10 Enter the name of the JMS **Topic** or **Queue** in the **Connectivity Map Editor** that is designated for receiving messages from SEWS.

The SEWS Executable uses the JMS Topic or Queue name to subscribe to a message from the routing point, such as "**SEWS_to_egate**", and mashals the ADK message into a SWIFT message. The SEWS Executable then converts the SWIFT message to a text message and delivers it to the SeeBeyond IQ Manager. You also need to subscribe to the JMS Queue or Topic in the eGate Collaboration.

TCP Port Number

11 Double-click the **TCP Port Number** line in the System Management Window, to open the configuration details for that parameter, (see Figure 13 below).

Figure 16 Systems Management – TCP Port Number

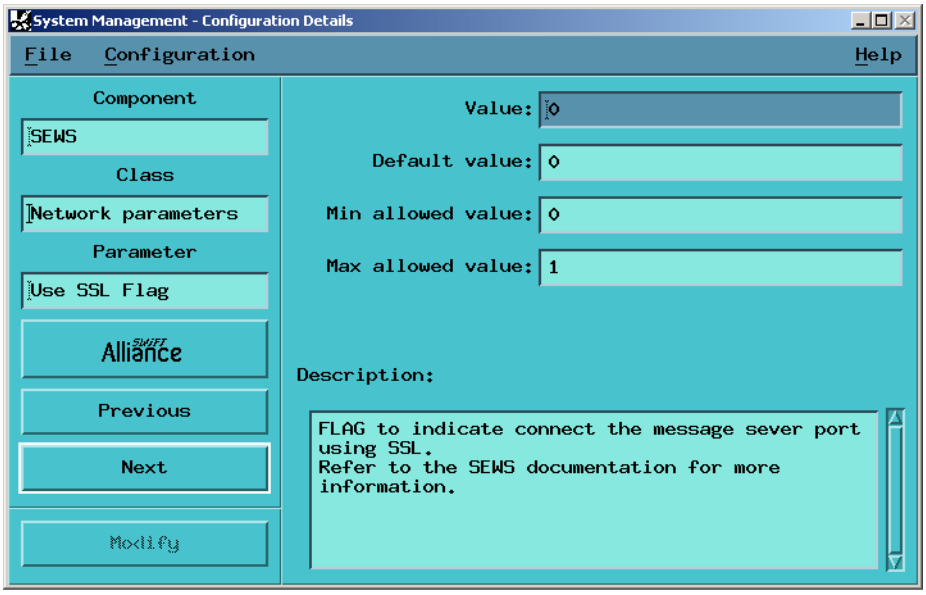


12 Enter the port number the SeeBeyond IQ Manager uses to communicate to the JMS message client. You can get this value from the **JMS IQ Manager Properties** window in the Environment Explorer.

Use SSL Flag

13 Double-click the **Use SSL Flag** line in the System Management Window, to open the configuration details for that parameter, (see Figure 13 below).

Figure 17 Systems Management – Use SSL Flag

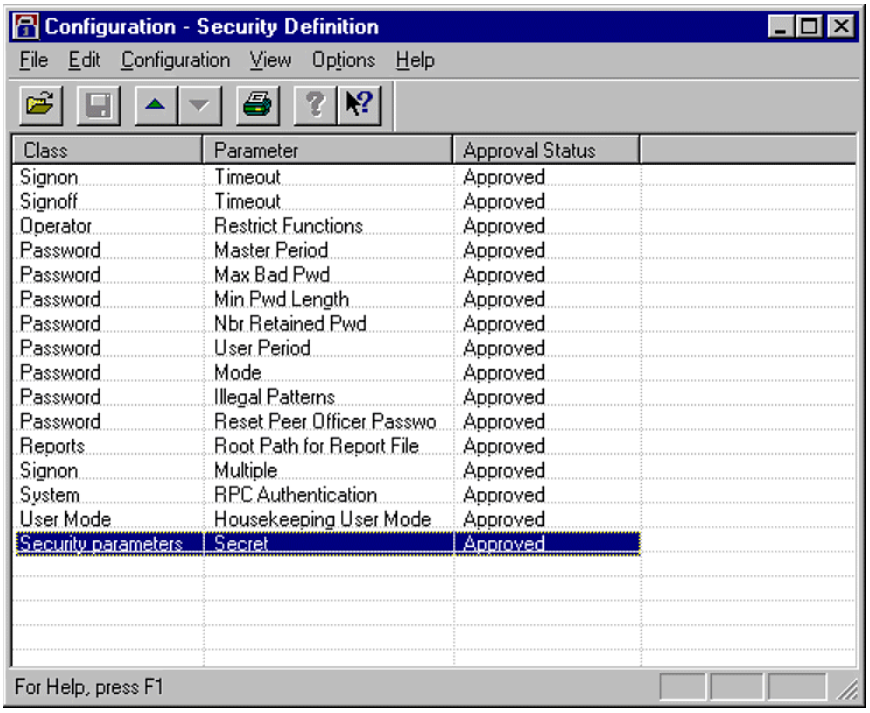


14 Enter an integer value of "**1**" if you want the SEWS executable to communicate with the JMS message server using Secured Sockets Layer (SSL) protocol. Otherwise, set the value to "**0**". If you set the value to "**1**", then the TCP Port Number in the previous section should be the SSL port used in the **SeeBeyond JMS IQ Manager's Properties dialog box**, located on the Environment Explorer.

4.4.3 Security Definition Window

The Security Definition window can only be run by security officers **LSO** and **RSO** (which stands for Left and Right Security Office, respectively). Any changes made by any security officer to any of the security definitions must be approved by both security officers before they can become effective.

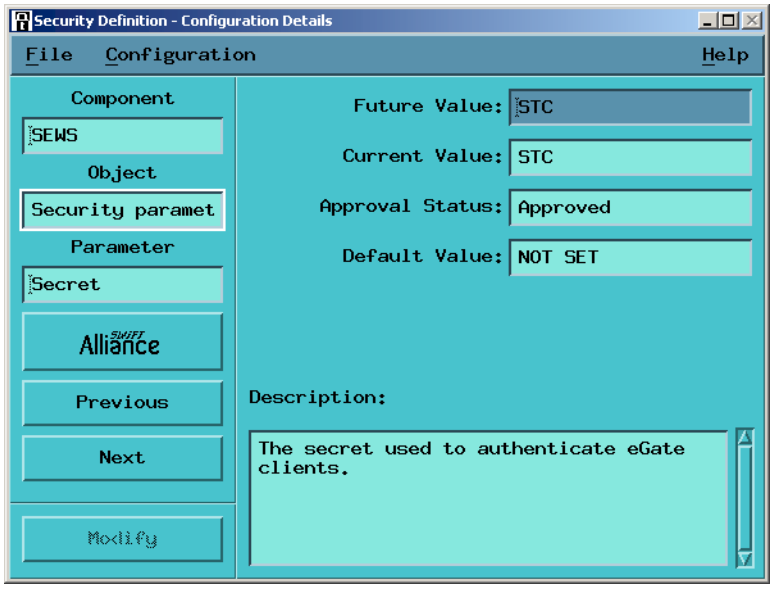
Figure 18 Security Definitions window



Secret

Enter the value that will be used to connect to eGate. You must first login as LSO to enter this value, then login as RSO to approve the value.

Figure 19 Security Definition window

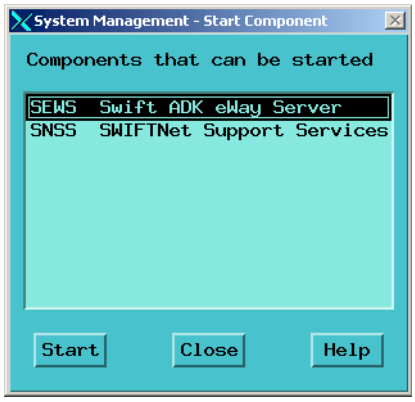


4.5 Starting the SEWS Swift ADK eWay Component

To start the SEWS component

1. Set your SWIFTAlliance servers to **operational** mode.
2. Select **Start Component** from the **File** menu in the System Management window. The Start Component window appears.

Figure 20 System Management – Star Component



3. Select **SEWS Swift ADK eWay Server** from the component list and click **Start**.

4.6 Setting Up a Test Environment (Optional)

If you do not have an existing SWIFT environment suitable for testing, you can set up a simple loopback configuration for testing purposes, following installation of the SEWS component. When running this test environment, any eGate-outbound events sent to the SEWS component will be routed back as eGate-inbound events, using the **SEWS_from_egate** and **SEWS_to_egate** routing points.

To create this test environment, you must set up the following items to match your system and your schema.

1. Queues for the routing points.
2. The routing points themselves.
3. Logical terminals.
4. TCP/IP parameters.
5. Security options.

Remember to set your SWIFTAlliance servers to **housekeeping** mode. All procedures except those related to security options require only Operator privileges.

Note: *When you create a new security, both the LSO and RSO must log in and **approve** it before you can **activate** it.*

4.6.1 Queues

First, you need to set up the queue for the routing points:

1. Launch the Systems Management window, invoke the **View** drop-down menu, and select **Queue**.
2. Open the routing point **SEWS_from_egate**, and select the Routing tab.
3. Under *Valid routing targets*, move **SEWS_to_egate** from **Available** to **Selected**.
4. **Modify** and **Save** this configuration.

4.6.2 Routing Points

Second, you need to configure the routing points to loop the messages from *inbound* back to *outbound*:

1. Launch the Routing window.
2. View the schemas to make sure your test schema is separate from any working schema. If not, create a dummy schema and move it to the **Selected** box under the **Used in Routing Schema(s)** field.

3 With the test schema active, open the routing point **SEWS_from_egate** and create a new routing rule, setting the following parameters to the indicated values.

Parameter	Value
Condition On	Function Result and Message
Function Result	Accept
Message	Creating_mpfn = 'SEWS_mpf'
Action On	source
Action	Route To (on the source sub-panel)
Route To	SEWS_to_egate
Append Intervention	No Intervention
Unit	Keep Current

4 Click **Validate**, then **Save** the configuration.

5 View the **Routing Rule** window and click **Add** or **Modify** to save the routing rule settings.

4.7 Authentication

The SWIFT ADK eWay only supports password based authentication. The SEWS component authenticates itself to the eGate IQ Manager by providing **Message Server Login** (see **Message Server Login** on page 29) and **Secret** (see **Secret** on page 34). For additional security, we recommend enabling SSL. The SEWS component always trusts the eGate IQ Manager. Server authorization (IQ Manager authenticating itself to SEWS) is currently not supported.

Chapter 5

Using JMS OTD with the SEWS Component

This chapter describes the properties used to establish message connections between the JMS OTD and the SEWS Component.

This Chapter Includes:

5.1 Publish to SEWS Asynchronously

This section describes the steps required to set up JMS properties that are required to publish to SEWS asynchronously (publish – outbound asynchronous). This component supports two types of messages:

- PUT
- LIST

For additional information on how these message types are implemented in a typical outbound asynchronous message flow, see **Publish (outbound asynchronous) Message Transmission Method** on page 9.

5.1.1 Setting the JMS Header property as "PUT"

To insert a SWIFT ADK eWay message using a Java Collaboration or BPEL business process, you must:

1 Create a text message that invokes the JMS OTD's *createTestMessage* method. In a Java Collaboration, this method appears as:

```
com.stc.connectors.jms.Message requestTextMsg =
JMS_1.createTextMessage();
```

2 Call the *storeUserProperty* method on the requestTextMsg object to set up the following JMS properties:

- **storeUserProperty("Header", "PUT")**, the "PUT" means the SEWS will put the message into routing point SEWS_from_egate.
- **storeUserProperty("type", "Swift")**, means the message type is Swift.
- **storeUserProperty("validation", "none")**, the level of message validation you want SAA to do it before insert to routing point. SAA supports these possible validation levels:
  - none
  - minimum
  - intermediate
  - maximum

  If you misspell the validation property or do not set validation property, the SEWS takes the default validation level: "maximum".
- **storeUserProperty("recourse_action","complete")**. The possible choices are:
  - **complete**: means the message will be complete if validation ok
  - **modify**: this means the message will be modified due to validation failure.
- **storeUserProperty("duplicate","true")**, you only set this user property when you know the message you want to insert into RP: SEWS_from_egate is a duplicated message.
- Set the text pay load on this JMS OTD object.

5.1.2 Setting the JMS Header property as "LIST"

Another "Header" action is "**LIST**", which allows you to get a list of messages on the routing point. The following JMS properties apply:

- **storeUserProperty("Header", "LIST")**, the "LIST" means the SEWS will list the messages on the RP.
- **storeUserProperty("state", "reserved")**, it will only list "reserved" messages, choices are:
  - all
  - reserved
  - unreserved

  If you do not set "state" property, the default state is "unreserved".
- **storeUserProperty("rp_name", "SEWS_to_egate"),** which RP you want to list, choices are:
  - SEWS_to_egate
  - SEWS_from_egate.

Publishing to SEWS asynchronously requires calling the *send* method on this OTD. If you use eGate 5.0.1 – 5.0.3, the JMS OTD does not have a *requestReply* method, so you can only call *send*. The result of "**PUT**" or "**LIST**" will come in eGate through the JMS queue/topic (as defined in the Sending T/Q name).

5.2 Publish to SEWS Synchronously

This section describes the steps required to set up JMS properties that are required to publish to SEWS synchronously (Request / Reply – outbound synchronous).

For additional information on how these message types are implemented in a typical outbound synchronous message flow, see **Request / Reply (outbound synchronous) Message Transmission Method** on page 10.

This feature is only supported in eGate 5.0.4 with the addition of a new method in the JMS OTD.

Everything is the same as in "Publish to SEWS asynchronously", except for the last step. Instead of calling the *send* method, you call *requestReply(requestTextMsg);*.

There will be a return result from calling *requestReply*, the return result is a com.stc.connectors.jms.Message type object. You should call *retrieveUserPropertyList* on the reply object to get all the user properties. Also you should call *retrieveStringFromMessage* to retrieve the reply message body.

If your request is "**Header=PUT**", then the reply message is a status report of "**PUT**". The reply message has the following JMS properties:

- **s_umid,** the value of this JMS property is the s_umid of new swift message we add to RP: SEWS_from_egate.
- **JMS_ProducerID,** the value is always ICANSEWSLSTN.
- **isOK,** the value if "TRUE" if you insert successfully, otherwise "FALSE".
- **times,** the value a series of timestamp the "PUT" request take to put a message.

If "**isOK=FLASE**", e.g. the message has the wrong format. The JMS property will list the cause of the failure, it will has the following properties:

- code, value may be "**SEWS_FAILED_TO_ADD**". Other possible codes are:
  - SEWS_UNKNOWN_COMMAND
  - SEWS_UNKNOWN_STATE
  - SEWS_FAILED_TO_ROUTE
  - SEWS_FAILED_TO_RESERVE
  - SEWS_UNKNOWN_MSG_TYPE
  - SEWS_NO_MSG_TYPE_SUPPLIED
  - SEWS_INVALID_RP.
- **JMS_ProducerID**, same as previous.
- **isOK**, "FALSE"
- **reason**, value is "ADK_FAILURE

If your request is "**Header=LIST**", then the reply message is a list of messages' **s_umids** The reply message has JMS properties:

- **isOK,** "TRUE" or "FALSE"
- **JMS_ProducerID,** the value is always ICANSEWSLSTN
- **code,** value may be "SEWS_FAILED_TO_LIST". Other possible codes are:
  - SEWS_UNKNOWN_COMMAND
  - SEWS_UNKNOWN_STATE
  - SEWS_FAILED_TO_ROUTE
  - SEWS_FAILED_TO_RESERVE
  - SEWS_UNKNOWN_MSG_TYPE
  - SEWS_NO_MSG_TYPE_SUPPLIED
  - SEWS_INVALID_RP.

  You get code only isOK=FALSE.

The reply message body will have a list of **s_umids** separated by space. In case there is no message, then the reply message body is a empty string, i.e. "".

5.3 Subscribe Asynchronously

This section describes the steps required to set up JMS properties that are required to subscribe to messages asynchronously (Subscribe – inbound asynchronous).

For additional information on how these message types are implemented in a typical inbound asynchronous message flow, see **Subscribe (inbound asynchronous) Message Transmission Method** on page 11.

These messages are delivered to eGate asynchronously from SEWS. There can be three types of messages:

1. The real swift message from RP: SEWS_to_egate, those messages are real SWIFT message and coming into egate as a JMS text message. The SWIFT message is contained in the JMS message body. It has following JMS message properties
   - **mesg_s_umid:** s_umid of the message
   - **inst_type:** always "ADK_INST_TYPE_ORIGINAL", we only support routing original message
   - **inst_num,** message instance number, normally 0 because we only support routing original message.
   - **times,** a series of timing data
   - **mesg_sub_format,** either "ADK_INPUT" or "ADK_OUTPUT"
   - **JMS_ProducerID**
   - **inst_s_umid,** the instance s_umid, normally same as mesg_s_umid.
   - **isOK,** always "TRUE"
   - **type,** can be "Swift" or "Telex"
   - **text_s_umid,** the message text body s_umid, normally same as mesg_s_umid
   - **s_umid,** the message body s_umid, normally same as mesg_s_umid

   The text payload of the JMS message will have the real Swift message without any string padding.
2. The status message, if you use "Publish to SEWS asynchronously", the message is a JMS message, has same format as we discussed in "Publish to SEWS synchronously", except the JMS_ProducerID has new value: ICANFROMSEWS.
3. A heart beat message saying there is no message to get from RP: SEWS_to_egate. It has JMS properties:
   - **code,** "SEWS_NO_MESSAGE"
   - **JMS_ProducerID,** "ICANFROMSEWS"
   - i**sOK,** "FALSE". It is always FALSE because no message on RP: SEWS_to_egate, you should also look at "code" property for "SEWS_NO_MESSAGE" to filter out such benign heart beat message.
   - **rp_name,** always "SEWS_to_egate"

Chapter 6

Locating, Importing, and Using the Sample Projects

This chapter describes how to use the sample projects included in the installation CD-ROM package.

Chapter Topics Include:

6.1 Sample Projects Overview

Sample projects are designed to provide an overview of the basic functionality of the SWIFT ADK eWay by demonstrating how to pass information between the SWIFTAlliance Access and eGate.

Sample Projects Include:

SWIFT_ADK_Sample_JCE – uses Collaboration Definitions (Java) to either listen for, or send messages to SWIFT.

The following components are found in the sample project:

- Connectivity Map (**cm_SWIFT_ADK**)
- File External Applications (**FileIN** and **FileOUT**)
- Java based Collaboration Definitions (**jce_ListenFromSwift** and **jce_SendToSwift**)
- Queues (**SEWS_from_egate** and **SEWS_to_egate**)

SWIFT_ADK_Sample_BPEL – uses BPEL based business processes that are designed to either listen for, or send messages to SWIFT.

The following components are found in the sample project:

- Connectivity Map (**cm_SWIFT_ADK**)
- File External Applications (**FileIN** and **FileOUT**)
- BPEL based business processes (**bp_ListenFromSwift** and **bp_SendToSwift**)
- DTD based OTD (**UserPropertyMsg_UserPropertyMsg**)
- Queues (**SEWS_from_egate** and **SEWS_to_egate**)

6.2 Locating and Importing the Sample Projects

The eWay sample projects are included in the **SWIFTADKeWayDocs.sar**. This file is uploaded separately from the **SWIFTADKeWay.sar** file during installation. For more information, refer to **"Installing the SWIFT ADK eWay" on page 12**.

Once you have uploaded the **SWIFTADKeWayDocs.sar** to the Repository, you can begin downloading the sample Projects from the **Documentation** tab on Enterprise Manager, to a folder of your choosing.

Before using the sample project, you must first import it into the SeeBeyond Enterprise Designer using the Enterprise Designer Project Import utility.

To Import the Sample Project:

1. From the Enterprise Designer's Project Explorer pane, right-click the Repository and select **Import**.
2. In the **Import Manager** window, browse to the directory that contains the sample Project zip file.
3. Select the sample file and then click **Open**.
4. Click the **Import** button. If the import was successful, then click the **OK** button on the **Import Status** window.

6.3 Running the Sample Projects

Steps required to run the sample projects include:

- Configure the Properties
- Creating the Environment Profile
- Deploying the Project
- Running the Sample

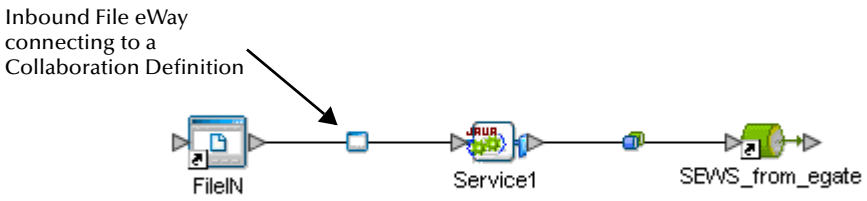
6.3.1 Setting the Properties

Sample projects use both an inbound and an outbound File eWay to deliver or receive sample data. The following information describes how to configure the File eWays from the Connectivity Map.

To Configure the File eWays:

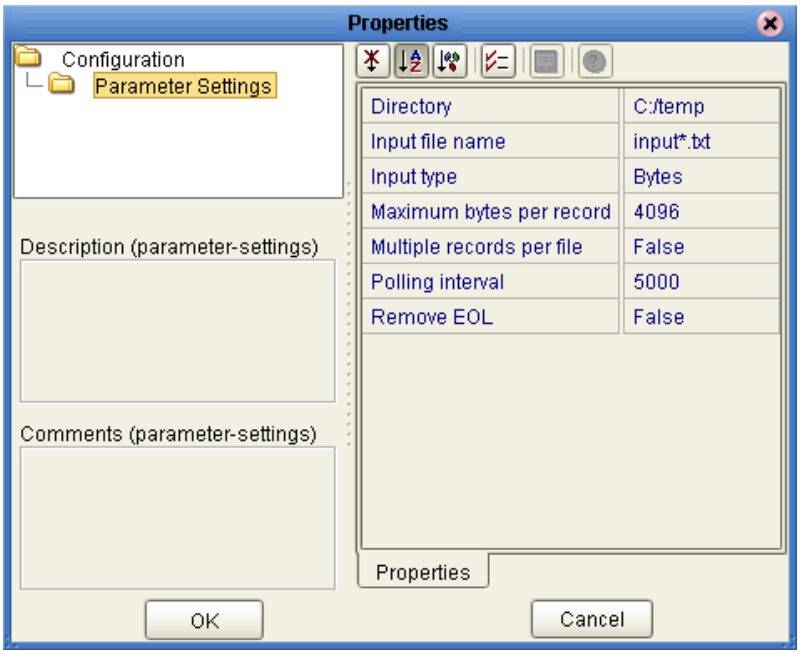
1 On the Connectivity Map, double-click the **Inbound File eWay** (denoted as the eWay that delivers data to the Collaboration Definition or BPEL Business Process).

Figure 21 Connectivity Map – Inbound File eWay



2 The **Properties** window for the Inbound File eWay opens. Modify any parameter settings necessary for your system, including the **Directory** and **Input file name** so they match the location and name of the sample data file, see **Data Used in the Sample Project** on page 51.

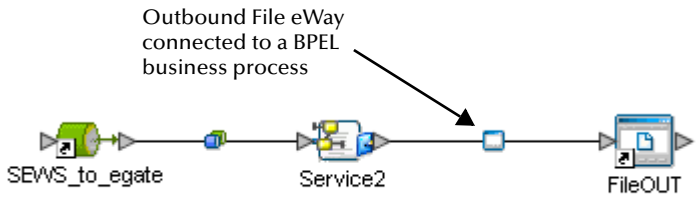
Figure 22 Properties Dialog Box – Inbound File eWay



3 Click **OK** to close the **Properties** window.

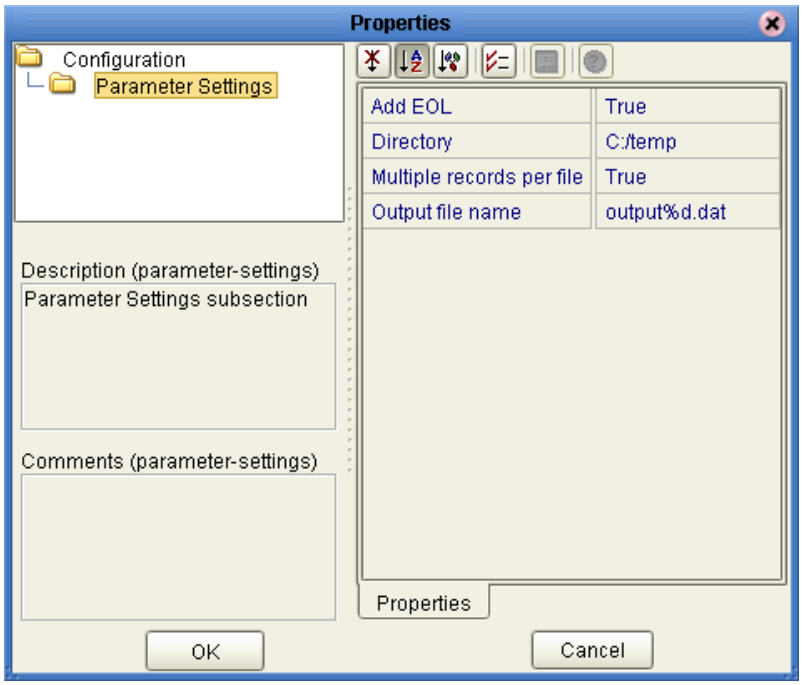
4 On the Connectivity Map, double-click the **Outbound File eWay** (denoted as the eWay that receives data to the Collaboration Definition or BPEL Business Process).

Figure 23 Connectivity Map – Outbound File eWay



5 The **Properties** window for the Outbound File eWay opens. Modify any parameter settings necessary for your system, including the **Output file name** and the target **Directory** where you want the resulting data to appear.

Figure 24 Properties Dialog Box – Outbound File eWay



6 Click **OK** to close the Properties window.

6.3.2. Creating the Environment Profile

An eGate Environment represents the physical system required to implement a Project. A typical Environment contains several components, including Logical Hosts, Integration Servers, JMS IQ Managers, and External Systems. Environments are created using the Enterprise Designer's Environment Explorer.

To Create a New Environment for the Sample Projects:

1 On the Environment Explorer, right-click the **Repository** icon and select **New Environment**. This creates a new Environment that you can rename to match the sample project you want to deploy.

2 Right-click the new Environment and add the following components:

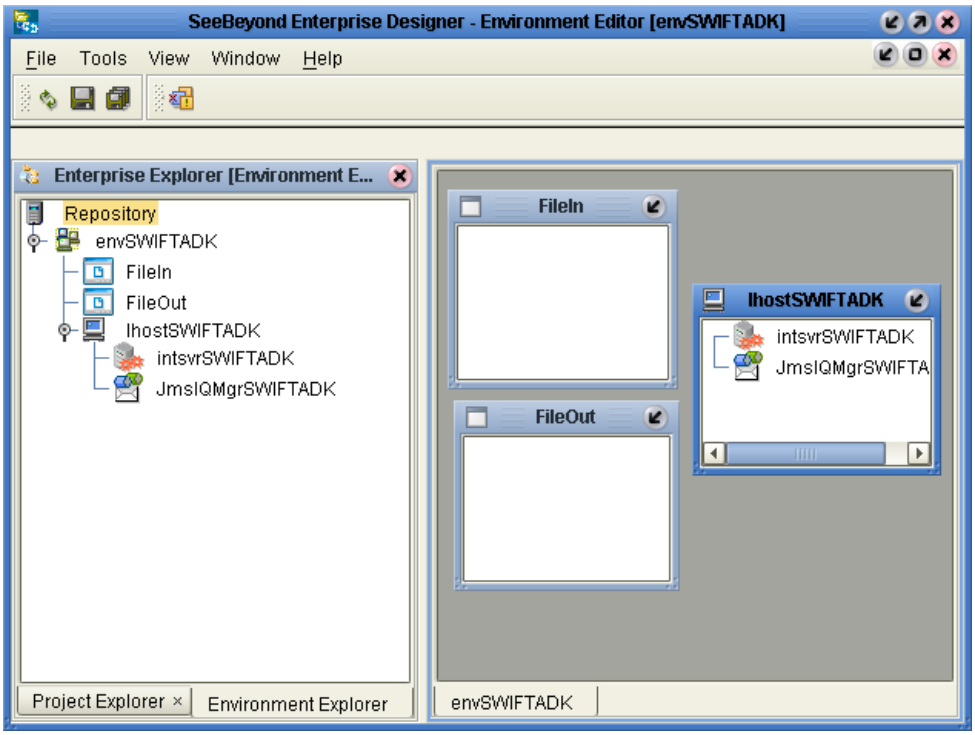
- New Logical Host
- New File External System (as an outbound eWay)
- New File External System (as an inbound eWay)

3 Right-click the new Logical Host and add the following components:

- New SeeBeyond Integration Server
- New SeeBeyond JMS IQ Manager

Figure 25 on page 47 shows an example of these Environment containers in the Environment Editor.

Figure 25 Environment Editor – Environment Containers



6.3.3 Deploying the Project

Deployment Profile maps the ICAN components with the Environment hosts (hosts) that support them. A project may have one or more Deployment Profiles, but each project's active Deployment Profile must reside within a separate Environment.

To create a Deployment Profile:

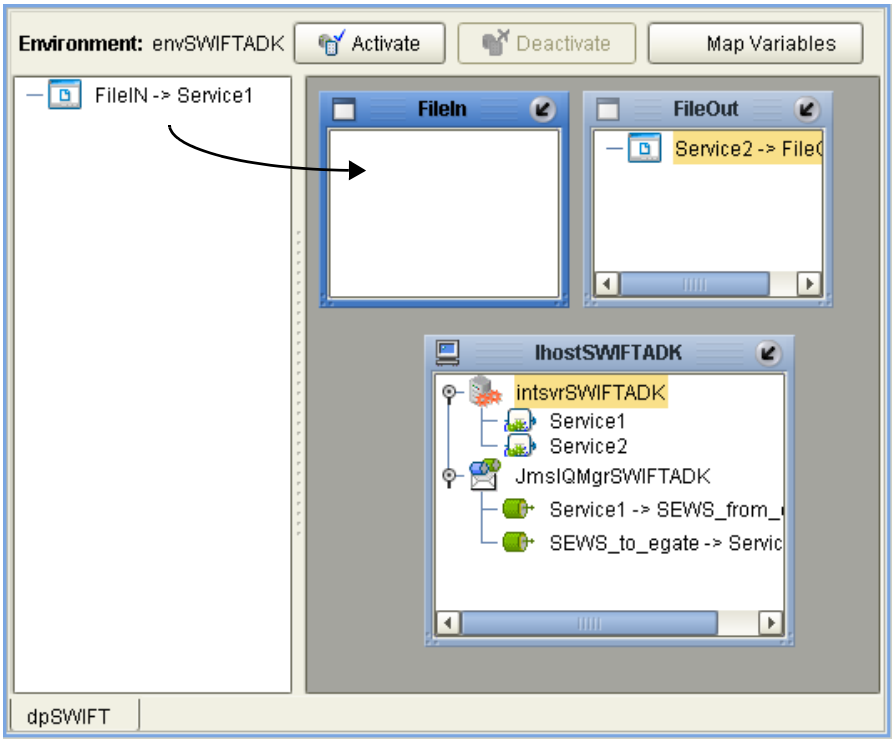
1 Right-click the sample project and select **New** > **Deployment Profile**.

2 In the **Create Deployment Profile** window, enter a name for the component and select the applicable **Environment** from the drop-down list. Click **OK** to close the window.

3 From the Deployment Profile pane, drag-and-drop all the ICAN icons from the left pane, to the applicable Environment containers on the right pane (See **Figure 26 on page 48**).

4 Click **Activate** to activate your **Deployment Profile**.

Figure 26 Dragging ICAN Icons to the Environment Containers



6.3.4. Running the Sample

Running a sample project requires starting the "Bootstrap" process. The Bootstrap is a process that launches the Integration Servers and Message Servers for one Logical Host. The Bootstrap process is started and stopped with the following scripts:

```
/ican50/logicalhost/bootstrap/bin/bootstrap.bat
```

```
/ican50/logicalhost/bootstrap/bin/shutdown.bat
```

When the Bootstrap is started it locates the Respository via the command line parameters or with the user-defined configuration file, **logical-host.properties**, located in:

```
/ican50/logicalhost/bootstrap/config
```

Bootstrap Invocation Parameters

The following table lists the parameters required to invoke the bootstrap. You can enter these parameters in the **logical-host.properties** file, or create a command line that runs these parameters in a separate **.bat** file.

Table 1 Bootstrap Invocation Parameters

Parameter	Description	Example
-r	Repository URL	http://localhost:12000/Repository
-e	Environment	Environment1
-l	Logical Host	LogicalHost1
-i	User Name	Administrator
-p	Password	STC

After completing the process, the Output file in the target directory—that was previously configured in the Outbound File eWay—contains all records retrieved from the database in text format.

6.4 Using the Sample Project in eInsight

This section describes in greater detail how the **SWIFT_ADK_Sample_BPEL** sample project works with the ICAN Suite's eInsight Business Process Manager and the Web Services interface. This section *does not* explain how to create a project using an eInsight business process. For these instructions, refer to the *"eInsight Business Process Manager User's Guide"*.

Before running this sample project, you must:

- Import the sample Project, (see **"Locating and Importing the Sample Projects" on page 44**)
- Configure the properties, (see **"Setting the Properties" on page 45**)
- Create an Environment for the sample project, (see **"Creating the Environment Profile" on page 46**)
- Create a Deployment Profile, (see **"Deploying the Project" on page 47**)

6.4.1 eInsight Engine and eGate Components

You can deploy an eGate component as an Activity in an eInsight Business Process. Once you have associated the desired component with an Activity, the eInsight engine can invoke it using a Web Services interface. Examples of eGate components that can interface with eInsight in this way are:

- Java Messaging Service (JMS)
- Object Type Definitions (OTDs)
- An eWay
- Collaborations

Using the eGate Enterprise Designer and eInsight, you can add an Activity to a Business Process, then associate that Activity—such as an eWay—with an eGate component. When eInsight runs the Business Process, it automatically invokes that component via its Web Services interface.

6.4.2 The SWIFT_ADK_Sample_BPEL Sample Project

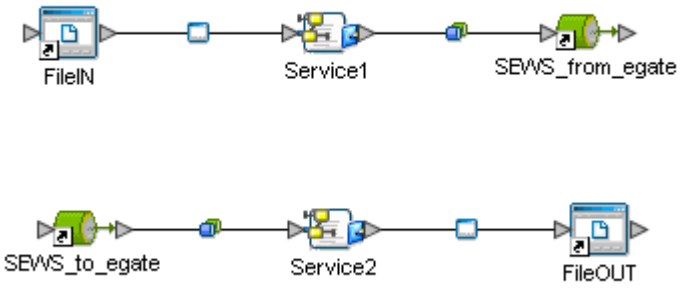
The **SWIFT_ADK_Sample_BPEL** sample Project uses two BPEL business processes to send data to, and receive data from the SWIFTAlliance Access. The flow of data between eGate and SWIFT are represented on the sample project **Connectivity Map**.

Sample Project Connectivity Map – cm_SWIFT_ADK

A Connectivity Map provides a canvas for assembling and configuring a Project's components. The icons on the Connectivity Map Editor tool bar represent the components used to populate the Connectivity Map canvas. When linked together, the components define the Connectivity Map for your Project.

The **cm_SWIFT_ADK** Connectivity Map contains two components. The first component displays the flow of data from eGate to SWIFT, and uses a BPEL business process called **bp_SendToSWIFT**, located in **Service1**. The second component displays the flow of data from SWIFT to eGate and uses a BPEL business process called **bp_ListenFromSWIFT**, located in **Service2**. Both of these Connectivity Map components are seen below in Figure 27.

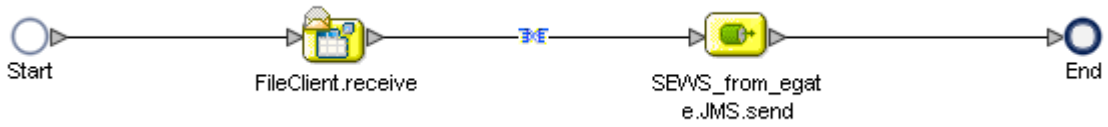
Figure 27 Components on the cm_SWIFT_ADK Connectivity Map



bp_SendToSWIFT BPEL Business Process

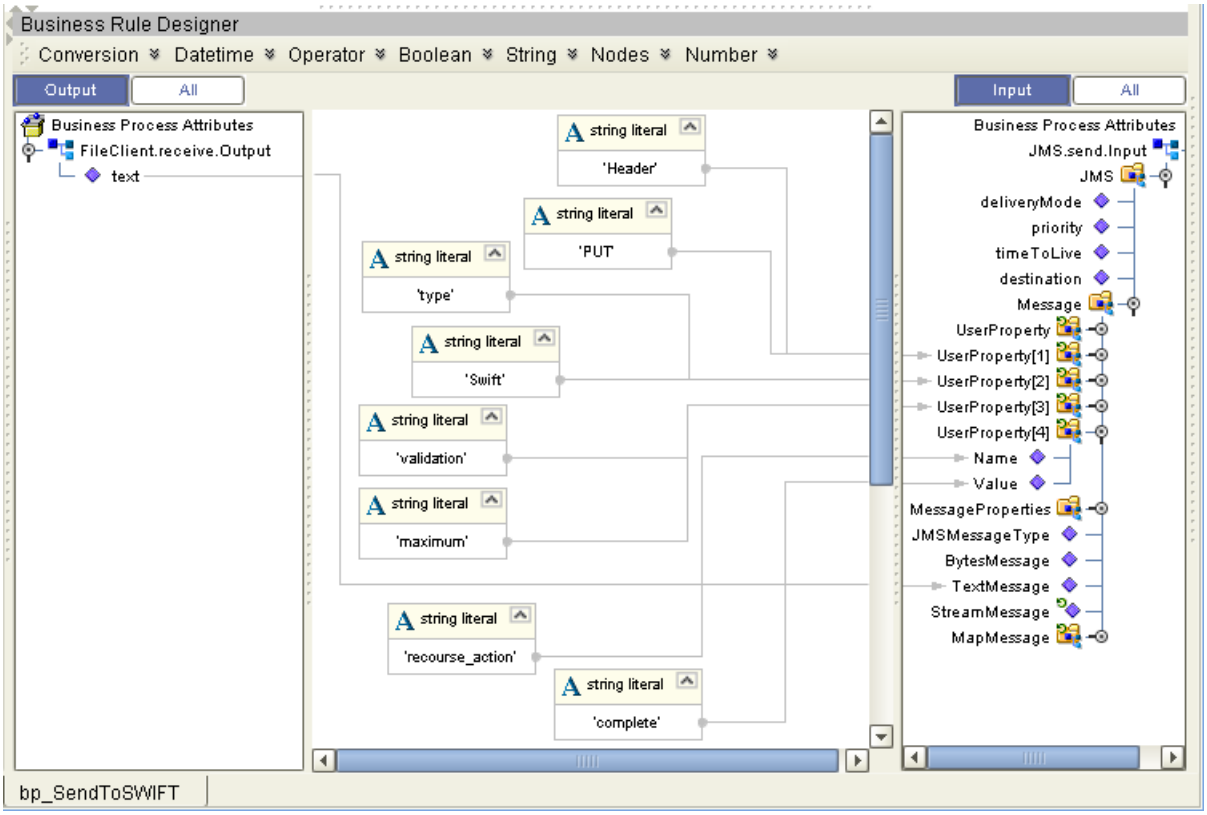
The **bp_SendToSWIFT** BPEL business process is designed to pass data from a sample text file, using the **FileClient.receive** web service, to the Message Properties OTD, using the **SEWS_from.JMS.send**.

Figure 28 bp_SendToSWIFT BPEL Business Process



In addition to passing through the entire contents of the text file, The **SWIFT ADK API** also requires four string literals that the business process passes in as message properties, as seen in Figure 29.

Figure 29 Business Rules in bp_SendToSWIFT



Data Used in the Sample Project

The data used for the inbound samples (data sent to SWIFT) is received from a text file included in the sample project called **Sample_SWIFT_Msg.txt**, see Figure 30 below.

Figure 30 Sample Data sent to SWIFTAlliance Access

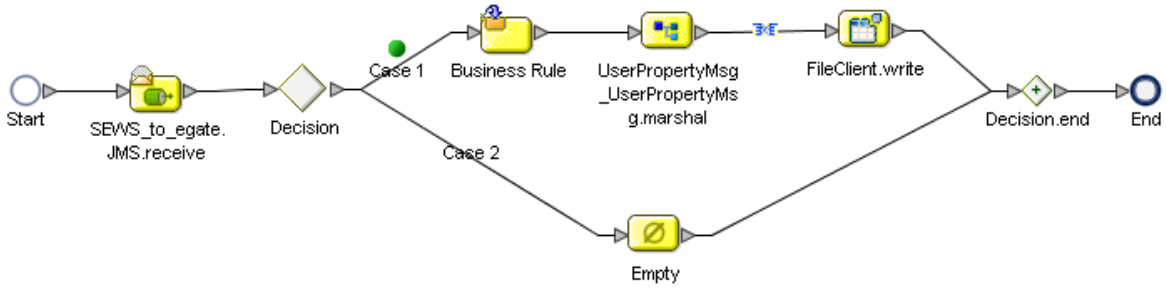
```
{1:F01PTSAUSZZAXXX1515293255}{2:I502050002070lPTSAUSZZ80A082645987402
510130500S}{3:{108:32516200}}{4:
:16R:GENL
:20C::SEME//UESS420701B02216
:23G:NEWM
```

```
:98C::PREP//20020701152928
:22F::TRTR//TRAD
:16R:LINK
:20C::MAST//FUNDS502
:16S:LINK
:16R:LINK
:20C::PREV//FUNDS502.1
:16S:LINK
:16S:GENL
:16R:ORDRDET
:94B::TRAD//EXCH/XPARFRPPBIC
:22H::BUSE//SELL
:22F::TRCN//SOLI
:22F::CAOP//CASH
:22F::TOOR//MAKT
:22F::TILI//GTCA
:22H::PAYM//APMT
:98A::EXPI//20040731
:11A::FXIS//EUR
:16R:TRADPRTY
:95Q::SELL//DBZEFRPPXXX
:16S:TRADPRTY
:36B::ORDR//UNIT/100,
:19A::ORDR//EUR1000,
:35B:ISIN FR0000127771
ACT. VIVENDI UNIVERSAL
/TS/EX/FRA
:16R:FIA
:22F::FORM//BEAR
:16S:FIA
:16S:ORDRDET
:16R:SETDET
:22F::SETR//TRAD
:16R:AMT
:19A::EXEC//EUR123,
:16S:AMT
:16R:AMT
:19A::CHAR//EUR123,
:16S:AMT
:16R:AMT
:19A::LOCO//EUR123,
:16S:AMT
:16S:SETDET
-}
```

bp_ListenFromSWIFT BPEL Business Process

The **bp_ListenFromSWIFT** BPEL business process is designed to listen for messages from SWIFT. Messages received by the business process first go through a Decision Activity to determine if the data is either valid or invalid. Valid data gets transformed and marshalled using a DTD based OTD before being written out to a text file. Invalid data passes to an empty Activity, see Figure 31 below.

Figure 31 bp_ListenFromSWIFT BPEL Business Process



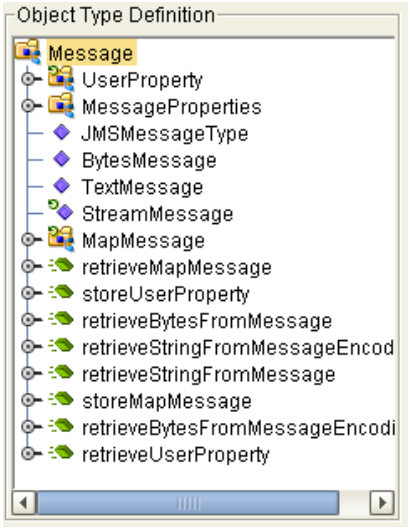
The following BPEL based Activities are used in the business process:

- **SEWS_to_egate.JMS.receive**
- **Branching Decision**
- **Business Rule**
- **UserPropertyMsg_UserPropertyMsg.marshal**

SEWS_to_egate.JMS.receive

The **SEWS_to_egate.JMS.receive** contains a JMS receive OTD, that uses JMS much like a web service to pull data into a queue. Figure 32 below lists the methods used.

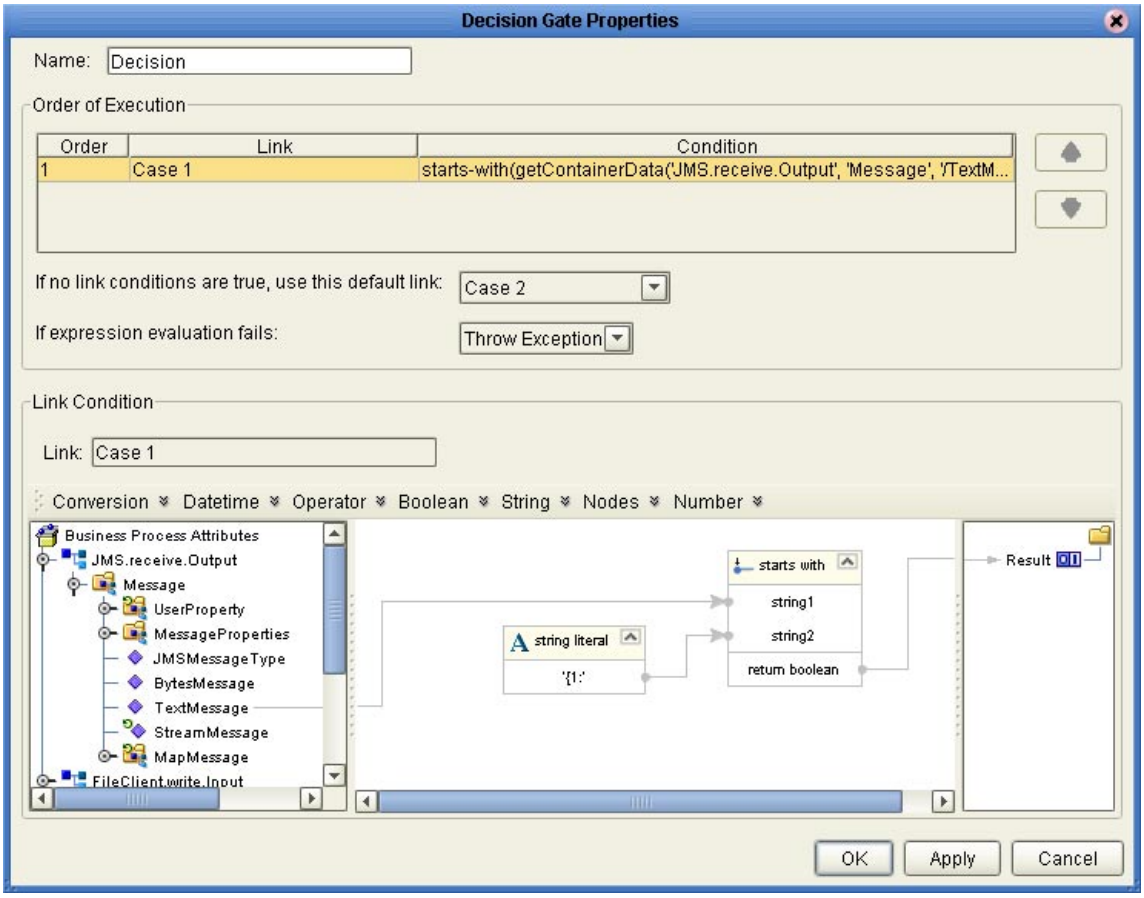
Figure 32 SEWS_to_egate.JMS.receive – Receive OTD



Branching Decision

The Branching Decision (Case1) is used to create a condition that looks for text in a receiving file that starts with "**{1:**". Text that matches this criteria passes onto a **Business Rule** Activity. Text that fails to match this criteria pass into an **Empty** Activity, see Figure 33.

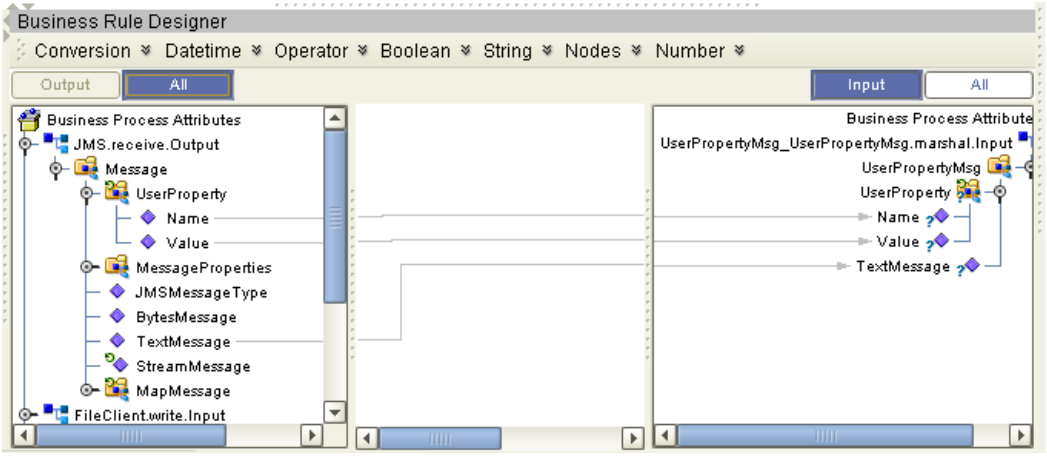
Figure 33 Branching Decision – Case1



Business Rule

The Business Rule is an Activity that passes the text message and the **Name** and **Value** user properties into the **UserPropertyMsg_UserPropertyMsg**, see Figure 34.

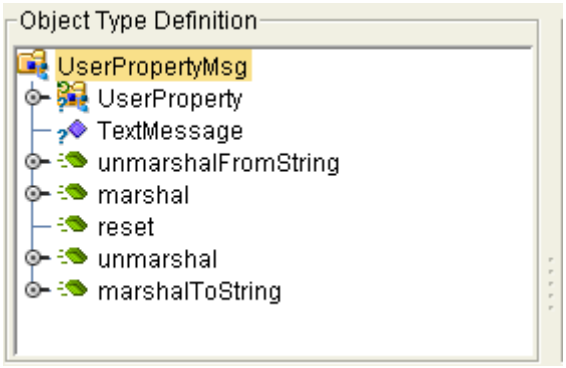
Figure 34 Business Rule Activity



UserPropertyMsg_UserPropertyMsg.marshal

The **UserPropertyMsg_UserPropertyMsg.marshal** is a DTD OTD based Activity that marshals the contents of the message before passing it onto the **FileClient.write** Activity. Figure 35 below lists the methods used by this OTD.

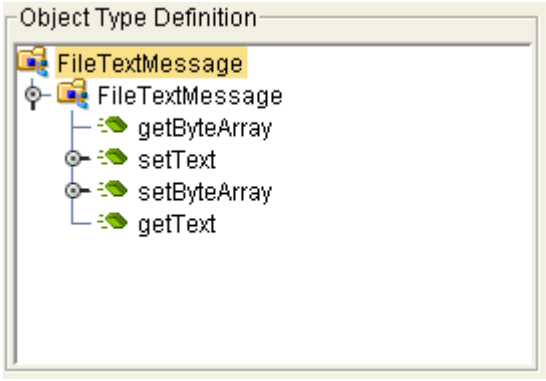
Figure 35 DTD OTD methods



FileClient.write

The **FileClient.write** is an Activity based on the File eWay's web service OTD, see Figure 36.

Figure 36 File eWay Write Methods



6.5 Using the Sample Project in eGate

This section describes in greater detail how the **SWIFT_ADK_Sample_JCE** sample project works with Java based Collaboration Definitions created with the Collaboration Editor. This section *does not* explain how to create Collaborations in eGate. For these instructions, refer to the *"eGate Integrator User's Guide"*.

Before running a sample Project, you must:

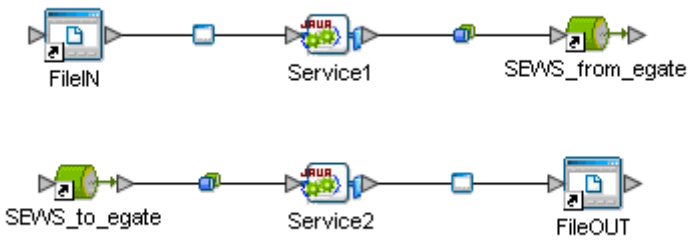
- Import the sample Project, (see **"Locating and Importing the Sample Projects" on page 44**)
- Configure the eWay properties, (see **"Setting the Properties" on page 45**)
- Create an Environment for the sample project, (see **"Creating the Environment Profile" on page 46**)
- Create a Deployment Profile, (see **"Deploying the Project" on page 47**)

Sample Project Connectivity Map – cm_SWIFT_ADK

A Connectivity Map provides a canvas for assembling and configuring a Project's components. The icons on the Connectivity Map Editor tool bar represent the components used to populate the Connectivity Map canvas. When linked together, the components define the Connectivity Map for your Project.

The **cm_SWIFT_ADK** Connectivity Map contains two components. The first component displays the flow of data from eGate to SWIFT, and uses a Java based Collaboration Definition called **jce_SendToSWIFT**, located in **Service1**. The second component displays the flow of data from SWIFT to eGate and uses a Java based Collaboration Definition called **jce_ListenFromSWIFT**, located in **Service2**. Both of these Connectivity Map components are seen below in Figure 27.

Figure 37 Components on the cm_SWIFT_ADK Connectivity Map



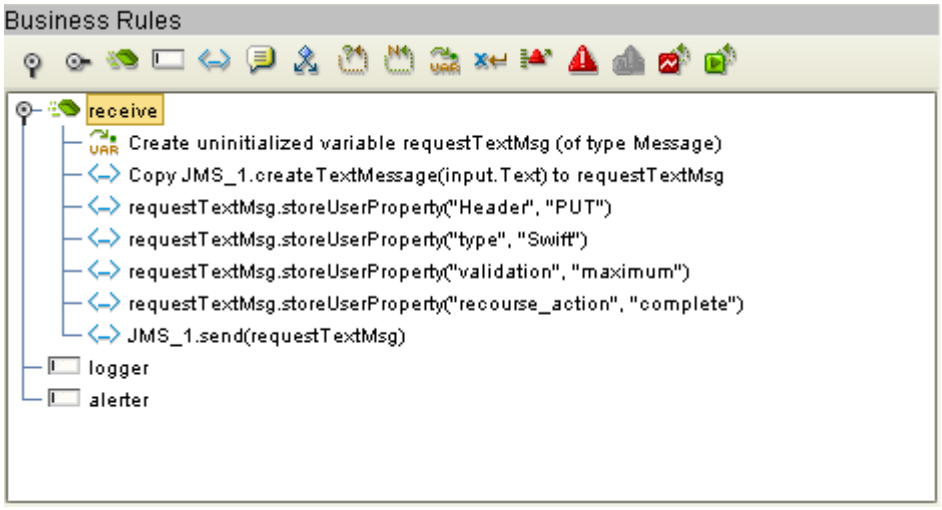
jce_SendToSWIFT Collaboration Definition

The **jce_SendToSWIFT** Collaboration Definition is designed to pass data from a sample text file, using the **FileClient.receive** web service, to the Message Properties OTD, using the **SEWS_from.JMS.send**. The following four user properties are also included with the text file using the **storeUserProperty()** method:

- **storeUserProperty[1]:** "Header", "PUT"
- **storeUserProperty[1]:** "type", "Swift"
- **storeUserProperty[1]:** "validation", "maximum"
- **storeUserProperty[1]:** "recourse_action", "complete"

Figure 38 below lists the business rules used to create the **jce_SendToSWIFT** Collaboration Definition.

Figure 38 jce_SendToSWIFT Collaboration Definition Business Rules

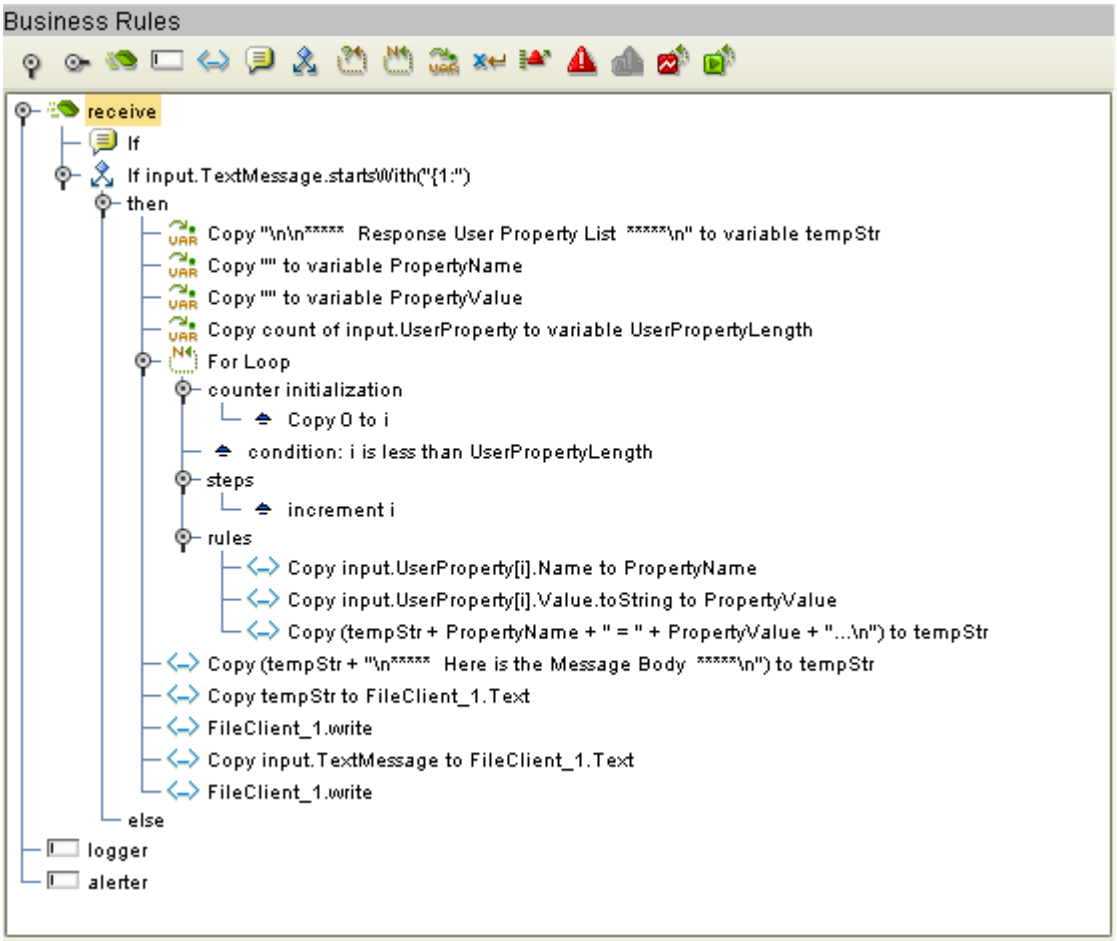


jce_ListenFromSWIFT Collaboration Definition

The **bp_ListenFromSWIFT** Collaboration Definition is designed to listen for messages from SWIFT. When a messages arrive into the JMS Queue, the Collaboration Definition first checks to see if the text of the message starts with "**{1:**". If the message is valid, then **Name** and **Value** parameters are passed into variables that are concatenated together and written out as a text message.

Figure 39 below lists the business rules used to create the jce_ListenFromSWIFT Collaboration Definition.

Figure 39 jce_ListenFromSWIFT Collaboration Definition Business Rules



6.6 SWIFT ADK eWay Error Messages

The following error messages can be returned:

Content	Meaning
“No messages to get”	The requested message does not exist.
“Unknown state”	The state given in the request was not recognized.
“No s_umid supplied”	The ACK/NAK request cannot succeed without a s_umid.
“No s_umid to ACK”	There is no outstanding message requiring acknowledgment.
“No s_umid to NAK”	There is no outstanding message requiring acknowledgment.
“Incorrect s_umid, pending s_umid supplied”	The s_umid given in the request is not the s_umid requiring acknowledgment. This response contains two additional arguments. ▪ s_umid: The s_umid of the message that is pending acknowledgment. ▪ instance: The instance number of that message.
“Failed to route message: ...”	After the message was added to the routing point, it couldn't be “routed on” in SWIFT terminology. The reason for this failure is included in the response Content.
“Failed to reserve message: ...“	The message specified could not be reserved. This response contains additional arguments on why the request failed. ▪ s_umid: The s_umid of the message that could not be acknowledged. ▪ instance: The instance number of the message that could not be acknowledged.
“Failed to get message: ...“	SEWS was not able to retrieve the message. The response content provides text that explains the reason for the error.

Content	Meaning
“Failed to get message, then failed to unreserve it: ...“	SEWS reserved the message, but could not retrieve it, and then could not unreserve the same message when recovering. Includes text further detailing the problem in the content.
“Failed to count/list instances ...“	SWIFTAlliance couldn't perform the operation. The human readable text - Content - includes the ADK error string that details the problem.
“Failed to add message: ...“	The message could not be added to the SWIFT routing point. The response content provides additional detail on why the procedure failed. When this error occurs, there are two additional arguments in the response. ▪ offset: The character offset into the text message, that caused the error. ▪ reason: Text giving further details about the error.
“Unknown message type”	The type argument contained an unexpected value. The response also contains a type argument: ▪ type: The type value given in the initial request that was not understood.
“Message type not supplied”	No type argument was supplied with the initial GET request.
“Pending ACK for another message”	No more messages can be retrieved until the last message is acknowledged. Responses with this code include another argument “pending”, which lists the s_umid of the pending message.
“Invalid routing point name supplied”	The routing point name given in the request was not recognized.
“No authorization”	Authentication was not possible.
“Too many clients connected”	SEWS cannot take more than one client per direction.

Content	Meaning
"Cannot translate SWIFT message to ADK"	The SWIFT message supplied in the request cannot be understood by SEWS. Perhaps there is a Signature error in the message. More exact details can be found in the SWIFTAlliance logs.
"Cannot translate TELEX message to ADK"	The Telex message supplied in the request cannot be understood by SEWS. Perhaps there is a Signature error in the message. More exact details can be found in the SWIFTAlliance logs.

Index

T

U