webMethods SWIFTNet Module Installation and User's Guide

Version 6.0.1 SP1

August 2006

Document ID: ESTD-SWIFTNET-IUG-601-SP1-20121011

Contents

About This Guide

This guide describes how to install, configure, and use the webMethods SWIFTNet Module, which supports synchronous communication of SWIFT messages and files between client and server applications.

To use this guide effectively, you should:

- Have a basic knowledge of SWIFT and SWIFT terminology. For more information, go to http://www.swift.com.
- Have installed all necessary SWIFT software. You must work with SWIFT to determine the appropriate software needs for your company.
- Have installed the webMethods Integration Server and the webMethods Trading Networks (server side and console side) software. For more information about installing the non-SWIFT components, see the *webMethods Installation Guide*.
- Be familiar with the webMethods Integration Server, the Server Administrator, and the webMethods Developer and understand the concepts and procedures described in the *webMethods Integration Server Administrator's Guide* and the *webMethods Developer User's Guide*.
- Be familiar with Trading Networks Console and understand the concepts and procedures described in the various webMethods Trading Networks guides.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Italic	Identifies variable information that you must supply or change based on your specific situation or environment. Identifies terms the first time they are defined in text. Also identifies service input and output variables.
Narrow font	Identifies storage locations for services on the webMethods Integration Server using the convention *folder.subfolder:service*.
`Typewriter font`	Identifies characters and values that you must type exactly or messages that the system displays on the console.
UPPERCASE	Identifies keyboard keys. Keys that you must press simultaneously are joined with the "+" symbol.

Convention	Description
\	Directory paths use the "\" directory delimiter unless the subject is UNIX-specific.
[ ]	Optional keywords or values are enclosed in [ ]. Do not type the [ ] symbols in your own code.

Additional Information

The webMethods Advantage Web site at http://advantage.webmethods.com provides you with important sources of information about your webMethods Integration Platform:

- **Troubleshooting Information.** The webMethods Knowledge Base provides troubleshooting information for various webMethods components.
- **Documentation Feedback.** To provide documentation feedback to webMethods, complete the Documentation Feedback Form on the webMethods Bookshelf.
- **Additional Documentation.** All of the webMethods documentation is available on the webMethods Bookshelf.

CHAPTER 1

Concepts

What Is SWIFTNet?

SWIFTNet is SWIFT's advanced IP-based messaging solution, which provides an alternate method of transferring information to SWIFT. It consists of a portfolio of products and services enabling the secure and reliable communication of financial information and transactional data.

SWIFTNet Link offers SWIFT users a single-window access to all SWIFTNet services. Its functionality includes transport, formatting, security and service management. Messages and files exchanged between parties consist of SWIFTNetLink (SNL) *primitives*. An SNL primitive is a pair of XML documents that is passed between an application program and the SNL software on your SWIFTAlliance Gateway (SAG) server. Each primitive document pair represents the invocation of a SWIFTNet processing function and includes an 'in' document and a corresponding 'out' document. These documents are either 1) the input for a function call to SWIFTNet, or 2) the SWIFTNet output of a function call.

SWIFTNet includes three messaging services:

- SWIFTNet InterAct. Allows the interactive (real-time) and store-and-forward exchange of messages between parties. It provides secure communication facilities for transferring messages. As the foundational messaging service of SWIFTNet, the InterAct Services include not only the InterAct Request/Response primitives, but also all of the supporting SWIFTNet infrastructure.
- SWIFTNet FileAct. Allows the exchange of files in an automated way, supporting both interactive (real-time) and store-and-forward modes. It provides secure communication facilities for transferring files. FileAct is oriented toward transferring data larger than the InterAct payload can accommodate.
- SWIFTNet Browse. Enables secure browser-based access to service providers' web servers. This service provides direct access to the secure messaging features of SWIFTNet InterAct and SWIFTNet FileAct. (This set of services is not currently supported by the webMethods SWIFTNet Module.)

For more information about SWIFT and SWIFTNet, see the documentation provided by SWIFT or go to http://www.swift.com.

What Is the webMethods SWIFTNet Module?

The webMethods SWIFTNet Module supports communication of SWIFT messages and files between client and server modules. A client module is the one that sends a request and receives a response. A server module is the one that receives a request and sends a response.

Important! Because client and server modules cannot co-exist in the same process, if you want to use both the client and server module services, you must install the webMethods SWIFTNet Module on two different Integration Servers. You must use only the WmSWIFTNetClient packages on one of the Integration Servers and only the WmSWIFTNetServer packages on the other Integration Server.

The SWIFTNet Module provides client-side and server-side support for the following messaging services and capabilities:

- InterAct Services
- FileAct Services

Both InterAct and FileAct services can work either in real-time mode or in store-and-forward mode. In real-time mode, both the requestor and the responder must be online at the same time. And in store-and-forward mode, the requestor and the responder need not be online at the same time.

SNL has only two functions: SwCall() and SwCallback(). SwCall() is used by the client module to access server module through the SWIFTNet. SwCallback() is used by server module to respond to clients through SWIFTNet.

As mentioned earlier, InterAct and FileAct services are implemented as a set of SNL primitives that are exchanged between the client or server module program and the SNL software on your SAG server. Along with its packages, the SWIFTNet Module provides two DLLs, WmSWIFTNetClient.dll and WmSWIFTNetServer.dll, that invoke the functionality of the SNL libraries to transfer the SNL primitives between the client and server modules.

For more information about the packages in this module, see "webMethods SWIFTNet Module Packages" on page 13.

Client Module Functionality

The WmSWIFTNetClient.dll invokes functionality for a client module, which sends a request to and receives a response from a server module in real-time or store-and-forward mode. When using the module with a client module, you can do the following:

- Send an InterAct request message and receive a response in real-time or store-and-forward mode.
- Put a file using FileAct service in real-time or store-and-forward mode.

- Get a file using FileAct service in real-time mode only.
- Pull messages from a queue in store-and-forward mode only.
- Fetch a file from SnF queue in store-and-forward mode only.

Server Module Functionality

The WmSWIFTNetServer.dll invokes functionality for a server module, which receives a request from and sends a response to a client module. When using the module with a server module, you can do the following:

- Receive an InterAct request message and send a response in real-time or store-and-forward mode.
- Accept a put file request from the client module in real-time mode only.
- Accept a get file request from the client module in real-time mode only.
- Receive the pushed messages from the SnF queue in store-and-forward mode only.

For more information about the architecture of the module, see "webMethods SWIFTNet Module Architecture" on page 14.

SNL Request and Response Primitives Support

The webMethods SWIFTNet Module supports all of the SNL request and response primitives involved in communication between the client module, the server module, and SWIFTNet. For a complete list of the supported primitives, see "Services and the SNL Request and Response Primitives" on page 60.

webMethods SWIFTNet Module Packages

The webMethods SWIFTNet Module contains the following packages, which contain webMethods services and related files, that you install on the webMethods Integration Server.

Important! Because client and server modules cannot co-exist in the same process, if you want to use both the client and server module services, you must install the webMethods SWIFTNet Module on two different Integration Servers. You must use only the WmSWIFTNetClient packages on one of the Integration Servers and only the WmSWIFTNetServer packages on the other Integration Server.

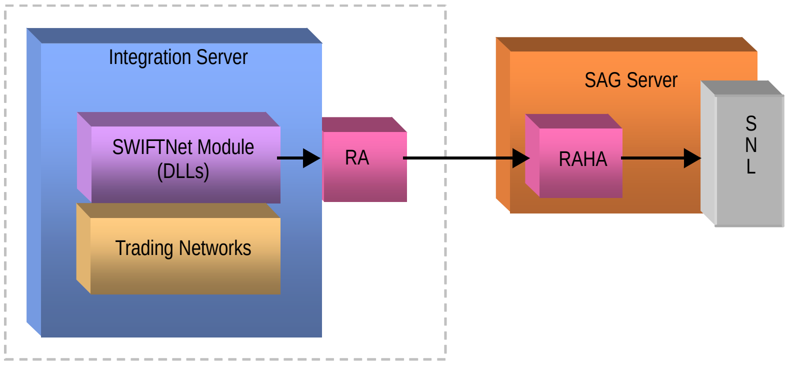
Package	Description
WmSWIFTNetClient	Contains the FileAct and InterAct services that support SWIFTNet Module client-side functionality.
WmSWIFTNetClientSample	Contains sample services and implementation of the SWIFTNet Module client-side functionality.
WmSWIFTNetServer	Contains the FileAct and InterAct services that support SWIFTNet Module server-side functionality.
WmSWIFTNetServerSample	Contains sample services and implementation of the SWIFTNet Module server-side functionality.

For detailed information about the contents of the these packages, see Chapter 5, “webMethods SWIFTNet Module Services” and Appendix A, “Samples” on page 81.

webMethods SWIFTNet Module Architecture

The following diagram illustrates the components involved in transferring messages and files using the webMethods SWIFTNet Module. See the table below the diagram for additional information.

SWIFTNet Module Architecture



As indicated in the diagram, the components listed in the following table comprise and support the SWIFTNet Module.

Component	Description
Integration Server	webMethods Integration Server is the underlying server of the webMethods Fabric. You use the web-based user interface, Server Administrator, to manage, configure, and administer all aspects of the Integration Server, such as users, security, packages, and services. For more information, see the *webMethods Integration Server Administrator's Guide*.
Trading Networks	webMethods Trading Networks enables your enterprise to link with other financial institutions and marketplaces to form a business-to-business trading network. In this instance, Trading Networks enables the webMethods SWIFTNet Module to exchange messages and files with your SWIFTAlliance Gateway (SAG) server. For more information about Trading Networks, see the *webMethods Trading Networks—Getting Started with Trading Networks* and the *webMethods Trading Networks—Building Your Trading Network* guides.

Component	Description
RA	The Remote API (RA) client enables the SWIFTNet Module to communicate with your SAG server and SNL through your Remote API Host Adapter (RAHA). You must install an RA client on the same machine as the Integration Server. To obtain an RA client, contact SWIFT.
RAHA	Your RAHA enables your SAG server to exchange messages and files with the Remote API (RA) client on your Integration Server. You must install an Remote API Host Adapter (RAHA) on the same machine as the SAG server. To obtain an RAHA, contact SWIFT.
SAG Server/SNL	Your SWIFTAlliance Gateway (SAG) server, on which you install your SWIFTNetLink (SNL) software, must be configured to exchange messages and files with SWIFTNet. You also will use this configuration information to configure the SWIFTNet Module and your RA client.

SWIFTNet Module Real-time Mode

Real-time InterAct message services are typically used when the receiver and the sender are online at the same time of message or file transmission. When real-time mode is used, the response comes from the server module at the Responder's site, which interprets the message sent.

Real-time InterAct

InterAct Services assure secure communication of Request/Response business messages between application-level clients and servers on SWIFTNet. It is cost-effective and ideal for online enquiry or reporting systems.

The sequence of the InterAct Request/Response session is as follows:

1. The Requestor's client module sends a Request.
2. The client Request is passed to SWIFTNet network, which processes the Request and sends it to the Responder's server module.
3. The Responder's server module receives the Request and sends the Response.
4. SWIFTNet processes the Response received from the Responder's server module and sends it to the Requestor's client module.
5. The Requestor's client module receives the Response.

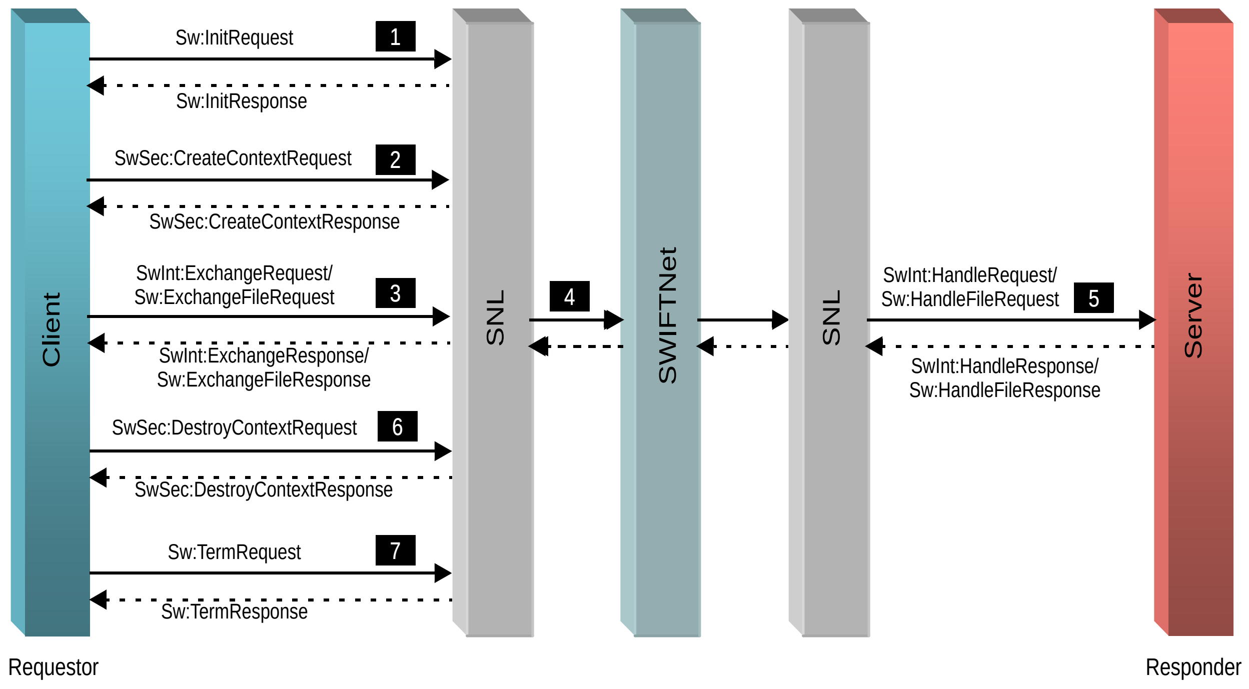
Real-time FileAct

Real-time FileAct Services offer a secure transfer of financial files between organizations on SWIFTNet. XML based FileAct primitives are used to transfer the files and maintain the status of the file transfers. FileAct services provide the following functionality:

- **Put File.** To send a file to another SWIFTNet user.
- **Get File.** To receive a file from another SWIFTNet user.
- **Subscribe to Transfer Events.** To receive progressive transfer status on an event-by-event basis.
- **Receive Transfer Events.** To respond to the terms of a subscription that is set up by the Subscribe Event primitive at the sending or receiving side of a transfer.

The following diagram illustrates the real-time InterAct/FileAct service. See the table below the diagram for additional information.

Real-time InterAct / FileAct Services



Step	Description
1	The Requestor's client sends the Sw:InitRequest primitive to initialize the SNL client process.
2	The Requestor's client makes a SwCall() with SwSec:CreateContextRequest as primitive to initialize the security context.
3	If it is an InterAct service, the client makes a request using the SwInt:ExchangeRequest primitive. If this is a FileAct service, the client makes a request using the Sw:ExchangeFileRequest primitive.
4	The Requestor's client side SNL passes the InterAct or FileAct request to the Responder's server side SNL via SWIFTNet.
5	The Responder's server side SNL extracts the request from SWIFTNet and invokes the server through SwCallback() using the SwInt:HandleRequest/Sw:HandleFileRequest as primitive. The Responder's server sends a response back to the client.
6	The client destroys the created security context.
7	The client triggers the termination with the SNL.

SWIFTNet Module Store and Forward Mode

In store-and-forward (SnF) mode, the messages and files are stored centrally within SWIFTNet in a queue. These messages and files are delivered from the queue to the receiver at a later time. Therefore, the Requestor and the Responder need not be online at the same time. A notification is sent to the Requestor in the event of non-delivery.

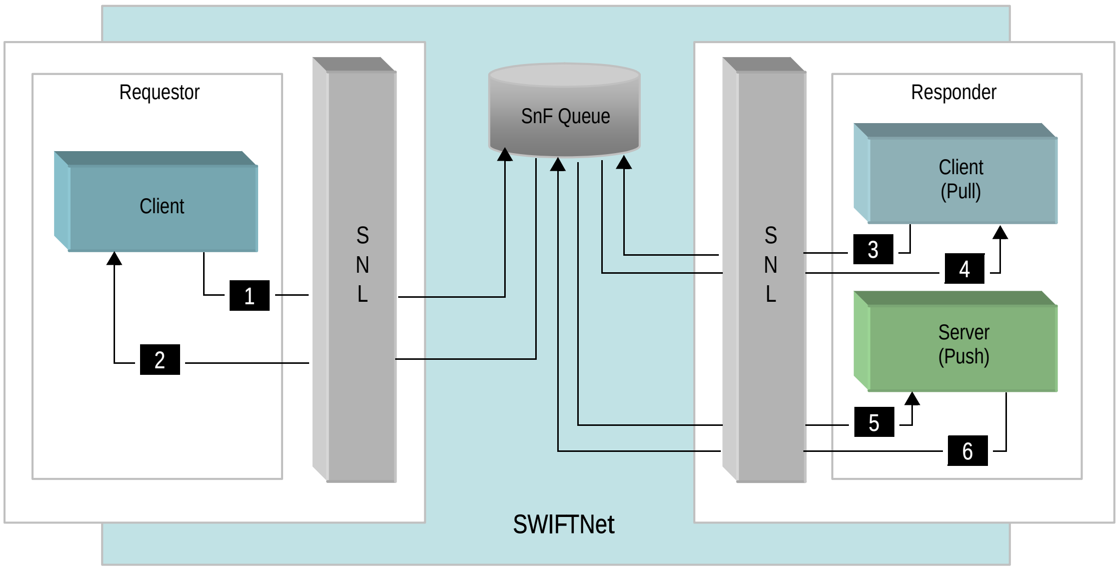
The SnF queues contain the messages and files that were sent by the Requestor to be delivered to the specified Responder. These queues also contain the delivery notifications generated by SWIFTNet SnF.

Messages and files can be routed into queues with the same flexibility as a message that is routed to a server process when real-time mode is used. Message Reception Registry function (MRR) specifies the message routing details. Queues are defined and configured by the Responder's organization. The Requestor decides in which queue a message or file will be put after it is sent by the Responder. The Responder will not know in which queue the message will be put.

In store-and-forward mode, the response comes from the SWIFTNet SnF queue and does not contain any feedback from the Responder. When real-time mode is used, the response comes from the server module on the Responder's side, which interprets the message sent.

Only the messages or files that are flagged for store-and-forward delivery mode are put in in a queue. Flagging can be done within the RequestControl for store-and-forward delivery mode for SWIFTNet InterAct and for SWIFTNet FileAct.

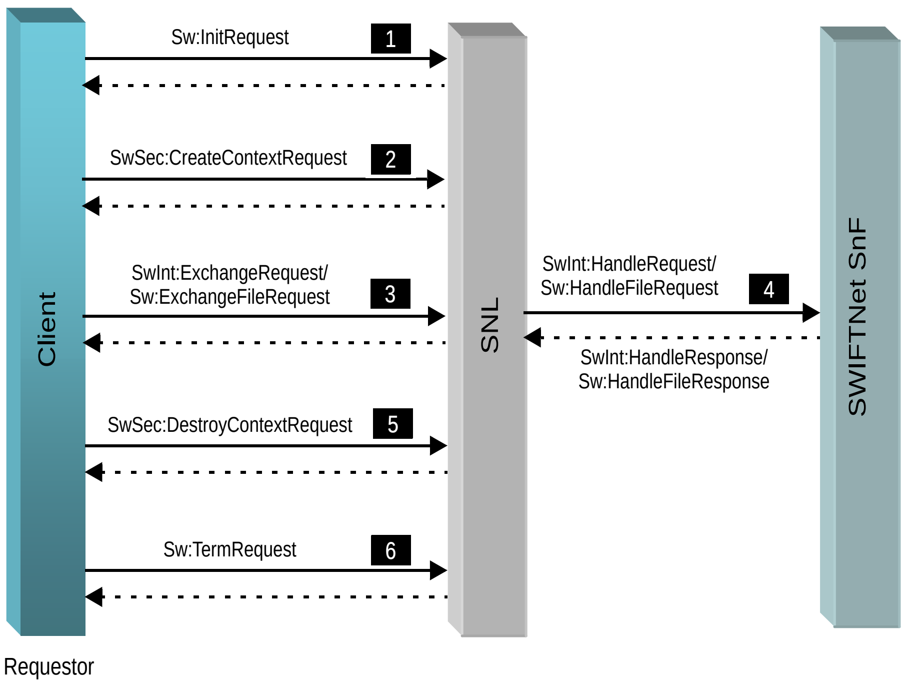
SWIFTNet Store and Forward Overview



Steps	Description
1 & 2	Requestor’s client sends messages or files to the SnF queue. The SnF queue stores the messages or files received, and sends a response to the Requestor’s client.
3 & 4	Responder’s client acquires the SnF queue in pull mode to pull the messages, and pulls the messages from the SnF queue.
5 & 6	Responder’s client acquires the SnF queue in push mode. The Responder’s server receives the pushed messages from the SnF queue and sends an acknowledgement.

The following diagram illustrates the store-and-forward flow on the Requestor’s side for an InterAct send or FileAct put session.

Requestor's Side Store-and-Forward Flow (InterAct send or put session)



Store and Forward InterAct

Store-and-forward InterAct services are used to send and receive messages when the Sender and the Receiver are not online at the same time. The Sender must indicate that the message is to be stored using SnF, and also indicate which queue will be used for storing any delivery notifications that will be generated by SWIFTNet SnF. If the file delivery fails, the failed delivery notification indicating the reason why the file was not delivered will be put in the queue of the Sender, as specified in the RequestControl.

Client processes on the Requestor's side initiate Requests and related functions. They pass a SWIFTNet primitive parameter to SNL representing the function to be performed.

A typical InterAct exchange request looks like this.

```
<?xml version="1.0"?>
<SwInt:ExchangeRequest>
  <SwSec:AuthorisationContext>
    <SwSec:UserDN>cn=abc,o=xxxx,o=swift</SwSec:UserDN>
  </SwSec:AuthorisationContext>
  <SwInt:Request>
    <SwInt:RequestControl>
      <SwInt:RequestCrypto>TRUE</SwInt:RequestCrypto>
      <SwInt:DeliveryCtrl>
        <SwInt:DeliveryMode>SnF</SwInt:DeliveryMode>
        <SwInt:NotifQueue>xxxx_generic!x</SwInt:NotifQueue>
        <Sw:DeliveryNotif>TRUE</Sw:DeliveryNotif>
      </SwInt:DeliveryCtrl>
    </SwInt:RequestControl>
    <SwInt:RequestHeader>
      <SwInt:Requestor>o=xxxx, o=swift</SwInt:Requestor>
      <SwInt:Responder>o=xxxx, o=swift</SwInt:Responder>
      <SwInt:Service>swift.generic.iast!x</SwInt:Service>
    </SwInt:RequestHeader>
    <SwInt:RequestPayload>This is for SnF Queue</SwInt:RequestPayload>
    <SwSec:Crypto>
      <SwSec:CryptoControl>
        <SwSec:MemberRef>RequestPayload</SwSec:MemberRef>
        <SwSec:SignDN>cn=abc,o=xxxx,o=swift</SwSec:SignDN>
      </SwSec:CryptoControl>
    </SwSec:Crypto>
  </SwInt:Request>
</SwInt:ExchangeRequest>
```

If the instruction to trigger store-and-forward is not contained in the SwInt:DeliveryCtrl element for an SnF service request, then SWIFTNet will reject the message. The SwSec:UserDN within the SwSec:AuthorisationContext must have the RBAC role "SnFRequestor" with the queue, as specified in the SwInt:NotifQueue as qualifier.

The queue in SwInt:NotifQueue is used to store failed delivery notifications. It must belong to the same institution as in the SwInt:Requestor. When the message is stored, SWIFTNet will indicate this in the Response.

Store and Forward FileAct

Store-and-forward FileAct services can only be used to send a file to a receiver and cannot be used to request a file.

A store-and-forward FileAct request looks similar to a real-time FileAct request message. The Sender must indicate that the file is to be stored using SnF, and also indicate the queue that will be used for storing any delivery notifications generated by SWIFTNet SnF. When the file delivery fails, the failed delivery notification indicating the reason why the file was not delivered will be put in the queue of the Sender, as specified in the RequestControl.

The following example shows an Sw:ExchangeFileRequest:

```
<Sw:ExchangeFileRequest>
     <SwSec:AuthorisationContext>
          <SwSec:UserDN>cn=abc,o=xxxx,o=swift</SwSec:UserDN>
     </SwSec:AuthorisationContext>
     <Sw:FileRequest>
          <Sw:FileRequestControl>
            <SwInt:RequestCrypto>FALSE</SwInt:RequestCrypto>
            <SwInt:DeliveryCtrl>
              <SwInt:DeliveryMode>SnF</SwInt:DeliveryMode>
```

```
               <SwInt:NotifQueue>xxxx_generic!x</SwInt:NotifQueue>
             </SwInt:DeliveryCtrl>
        </Sw:FileRequestControl>
        <Sw:FileRequestHeader>
              <SwInt:Requestor>o=xxxx, o=swift</SwInt:Requestor>
              <SwInt:Responder>o=xxxx, o=swift</SwInt:Responder>
              <SwInt:Service>swift.generic.fast!x</SwInt:Service>
        </Sw:FileRequestHeader>
        <Sw:FileOpRequest>
       <Sw:PutFileRequest>
                <Sw:TransferDescription>atlog.txt</Sw:TransferDescription>
                <Sw:PhysicalName>C:\atlog.txt</Sw:PhysicalName>
       </Sw:PutFileRequest>
       </Sw:FileOpRequest>
     </Sw:FileRequest>
</Sw:ExchangeFileRequest>
```

Retrieving Messages and Files from a Queue

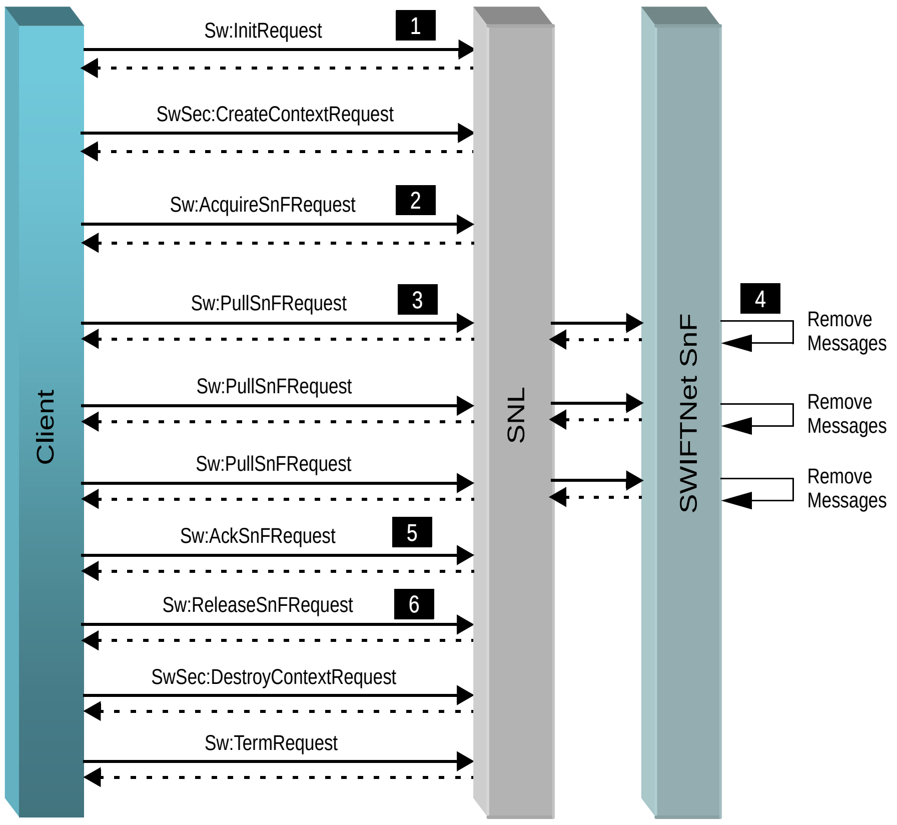
Messages and files can be retrieved from a queue using the pull or push modes.

Pull Mode

When the pull mode is used, the client process initiates the delivery of a message. It performs an SwCall() with an Sw:PullSnFRequest as the input primitive. The Sw:PullSnFResponse contains the message pulled from the queue.

The following diagram illustrates the store-and-forward InterAct pull session. See the table below the diagram for additional information.

Store and Forward InterAct Pull Session



Step	Description
1	The client sends the Sw:InitRequest to start the delivery of messages and files from a SnF queue. Then the client opens the desired security context using the SwSec:CreateContextRequest primitive.
2	The client sends a request to acquire the queue. After receiving the response, the client starts the delivery of messages from the queue by issuing the Sw:PullSnFRequest.
3	The first Sw:PullSnFRequest does not carry an acknowledgement, but all subsequent requests must acknowledge the message delivered in the previous pull request and avoid the same message being delivered again.
4	Messages are removed from the queue.

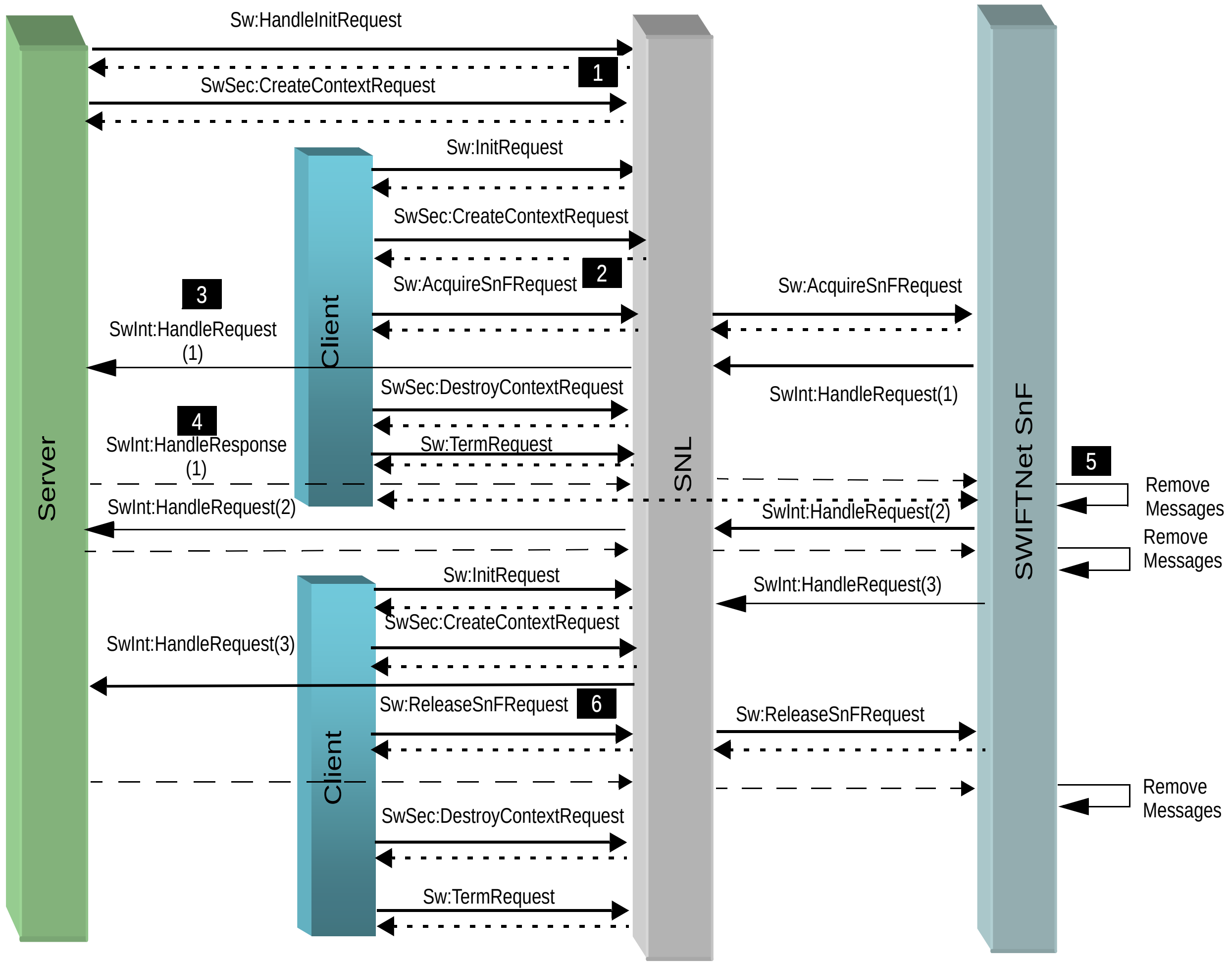
Step	Description
5	The client sends Sw:AckSnFRequest (Sw:ExchangeSnFRequest) along with the acknowledgement of the last delivered message as input primitive when it wants to stop the delivery of messages.
6	The client destroys the created security context and triggers the termination with the SNL.

Push Mode

When push mode is used, the initiative to deliver a message resides with SWIFTNet SnF. The message is pushed from SWIFTNet SnF and is received by the server module on SWIFTNet Link. In this server a regular SwCallback() is invoked. The input primitive is the message from the queue within an SwInt:HandleRequest or Sw:HandleFileRequest. The server module ensures that the message is stored safely, and then responds with an acknowledgement. This acknowledgement indicates to SWIFTNet SnF how the message was received.

The following diagram illustrates the store-and-forward InterAct Push session. See the table below the diagram for additional information.

Store and Forward InterAct Push Session



Step	Description
1	The server process opens the required security context with Sw:HandleInitRequest and SwSec:CreateContextRequest. The server gets ready to start processing the incoming requests.
2	The client process starts, sends the Sw:InitRequest, opens the desired security context, acquires the queue in pull mode, and starts the delivery of messages from the queue.
3	The server receives a SwInt:HandleRequest request.
4	The server sends an acknowledgement in SwInt:HandleResponse.

Step	Description
5	Messages are removed from the queue.
6	The client releases the queue.

Fetching a File from a Queue

If a FileAct message is received in either pull mode or push mode, a client process must fetch a file. A file is always fetched in a client process when the SWIFTNet SnF has delivered an Sw:NotifyFileRequestHandle to indicate the presence of a file to be fetched. In push mode, Sw:NotifyFileRequestHandle is delivered within Sw:HandleFileRequest, and in pull mode, Sw:NotifyFileRequestHandle is delivered within Sw:PullSnFResponse.

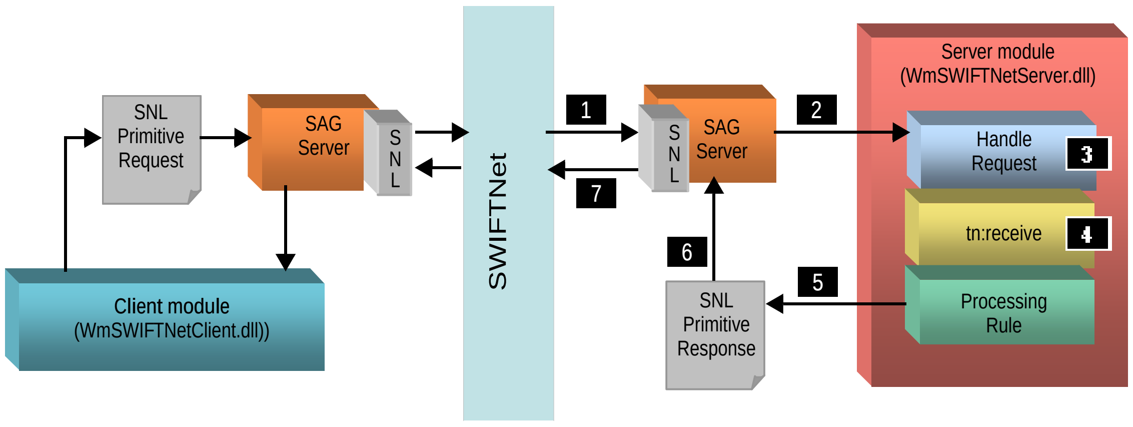
The Sw:FetchFileRequest simply copies the structure received in the Sw:FileRequestHandle pushed (or pulled). The Response to this Request is the TransferRef that is used to identify the file transfer from SWIFTNet SnF to the Receiver. For a pull session, no other message will be delivered for that queue until the file has been fetched and a delivery acknowledgement has been sent.

Important! When a file is fetched from the queue, the file will remain within SWIFTNet SnF until an explicit acknowledgement has been sent by a client process.

Server Module Processing of SNL Primitives

The following diagram illustrates how the SNL primitives are processed by the server module when a client module sends a request. See the table below the diagram for additional information.

webMethods SWIFTNet Server Module Processing



Step	Description
1	The Requestor's client module sends a request to the server via SWIFTNet.
2	The Responder's SAG server receives the SNL primitive request and invokes the wm.swiftnet.server.services:handleRequest service in the Integration Server on which the server module is installed.
3	The handleRequest service of the server module then invokes the wm.tn:receive service of the Trading Networks.
4	Trading Networks uses the TN document types that you create to recognize the incoming request, saves the request to the Trading Networks database, and invokes one of the processing rules you created and associated with the request's TN document type.
5	The processing rule processes the document as necessary and generates the XML response.
6	The XML response is sent to the SAG server.
7	The SAG server sends the response back to the Requestor's client module via SWIFTNet.

.

webMethods

CHAPTER 2

Installing the webMethods SWIFTNet Module

Overview

Important! The information in this chapter might have been updated since the guide was published. Go to the webMethods Advantage Web site at http://advantage.webmethods.com for the latest version of the guide.

This chapter contains installation instructions that address two different scenarios:

- **New installation:** For a new installation of webMethods SWIFTNet Module, you need to install the base product (webMethods SWIFTNet Module) and the service pack. Options in the Installer enable you to select both at the same time. Follow the instructions in "Installing the webMethods SWIFTNet Module" on page 30.
- **Existing installation:** If you already have the base product installed, you only need to apply the service pack. Follow the instructions in "Applying the webMethods SWIFTNet Module Service Pack 1" on page 32.

If you are installing the webMethods SWIFTNet Module with other webMethods components such as webMethods Integration Server, see the *webMethods Installation Guide* for instructions on installing those components.

Requirements

This section describes the system requirements that must be met before you can install the webMethods SWIFTNet Module.

Supported Platforms and Operating Systems

The webMethods SWIFTNet Module supports the following platforms and uses the same browsers and the same JVM versions as its host Integration Server:

- Microsoft Windows 2000 Professional
- Microsoft Windows 2003 Server
- Microsoft Windows XP Professional

Note: webMethods SWIFTNet Module supports Windows 2000 Professional platform only on Integration Server 6.1 and not on Integration Server 6.5.

Required webMethods Components

The following table lists the webMethods components you must install before you install the SWIFTNet Module:

Component	Version
webMethods Integration Server	6.1 or 6.5
webMethods Developer	6.1 or 6.5
webMethods Trading Networks	6.1 or 6.5

Note: Please download the webMethods components with the latest Service Packs from the webMethods Advantage Web site at http://advantage.webmethods.com.

SWIFTNet Module Requirements

The following table lists the SWIFT software required to operate the SWIFTNet Module.

Software	Release
SWIFTAlliance Gateway (SAG)	5.0.0 or later
SWIFTNet Link (SNL)	5.0.0 or later
Remote API Host Adapter (RAHA)	5.0.0 or later
Remote API Client (RA)	5.0.0 or later

Regardless of whether you are using the SWIFTNet Module for a client or server module, you must install a Remote API (RA) client on the same machine where the Integration Server is installed. Install a Remote API Host Adapter (RAHA) on the same machine as the SAG server. Both the RA client and the RAHA are provided by SWIFT. For more information, see your SWIFT documentation or go to http://www.swift.com.

Hardware Requirements

The webMethods SWIFTNet Module has no hardware requirements beyond those of its host Integration Server.

Installing the webMethods SWIFTNet Module

Important! You must have administrator privileges on the webMethods Integration Server to execute these procedures. If you do not have administrator privileges, have your webMethods Integration Server administrator perform these procedures.

Note: This section is for new SWIFTNet Module users. Follow the instructions provided in this section to install the webMethods SWIFTNet Module along with the Service Pack 1. To install just the Service Pack 1 over an existing installation of the SWIFTNet Module, see "Applying the webMethods SWIFTNet Module Service Pack 1" on page 32.

This section provides only instructions that are specific to installing the webMethods SWIFTNet Module. For complete instructions on using the webMethods Installer, see the *webMethods Installation Guide*.

Perform the following steps to install and use the webMethods SWIFTNet Module.

Step	Description
1	Install the webMethods SWIFTNet Module
2	Configure the webMethods SWIFTNet Module
3	Define the Trading Networks information

Step 1: Install webMethods SWIFTNet Module

Install the webMethods SWIFTNet Module on the same machine as the Integration Server. The Installer will automatically install the SWIFTNet Module in the Integration Server installation directory.

Important! Because client and server modules cannot co-exist in the same process, if you want to use both the client and server module services, you must install the webMethods SWIFTNet Module on two different Integration Servers. You must use only the WmSWIFTNetClient packages on one Integration Server and only the WmSWIFTNetServer packages on the other Integration Server.

Install the SWIFTNet Module

1 Download the latest webMethods Installer from the webMethods Advantage Web site at http://advantage.webmethods.com.

2 If you are going to install the webMethods SWIFTNet Module on an already installed Integration Server, shut down the Integration Server.

3. Start the Installer.
4. Choose the webMethods platform on which to install the webMethods SWIFTNet Module. If you are going to install the webMethods SWIFTNet Module on an existing Integration Server, select the platform that matches the release of that Integration Server. For example, if you are going to install the webMethods SWIFTNet Module on a 6.5 Integration Server, select the 6.5 platform.
5. Specify the webMethods SWIFTNet Module installation directory as the webMethods 6.5 installation directory (by default, webMethods6).
6. In the component selection list, navigate to **eStandards ▸ SWIFTNet Module 6.0.1** and select one of the following components to install:

   a. **SWIFTNet Client** to install the client component along with the Service Pack 1.

   b. **SWIFTNet Server** to install the server component along with the Service Pack 1.

   Each component includes the following selections:

   - **Documentation** (Optional). Contains the documentation for this package.
   - **Program Files** (Required). Contains the program files for this package.
   - **Samples** (Recommended). Contains the services that demonstrate how to use the SWIFTNet Module services.

   In addition, select any required webMethods components you have not installed. For a list of required components, see "Required webMethods Components" on page 29.
7. Click **Next** until you see the installation complete message.
8. Click **Close**.

Important! Before starting the Integration Server on which you have installed the SWIFTNet Module, you must first configure your SWIFTNet Module and then configure and start your SWIFTAlliance Gateway (SAG). If at any time the SAG server restarts, you must reload the WmSWIFTNetServer and WmSWIFTNetClient packages.

The webMethods SWIFTNet Module starts automatically when you start the webMethods Integration Server.

Step 2: Configure the SWIFTNet Module

To configure the Server module and the Client module to exchange messages and files over SWIFTNet using the SWIFTNet Module, edit the following files:

- Server module env.cnf
- Client module env.cnf
- Server module snl.cnf

For more information, see "Configuring the webMethods SWIFTNet Module" on page 39.

Step 3: Define the Trading Networks Information

You must perform the following procedures in Trading Networks to enable successful interaction with your trading partners:

- Define your Enterprise profile
- Define your Trading Partner profiles
- Define TN document types
- Define processing rules

For details regarding defining profiles, TN document types, and processing rules in Trading Networks, see "Defining Trading Networks Information" on page 49.

Applying the webMethods SWIFTNet Module Service Pack 1

If you have already installed the webMethods SWIFTNet Module and now would like to update it with Service Pack 1, perform the following steps.

To apply the webMethods SWIFTNet Module Service Pack 1 to the Server Module

Note: Follow this set of instructions only if you are installing the Service Pack 1 over an existing installation of the SWIFTNet Module.

1. Take a backup of the following packages in the *IntegrationServer*\packages directory:
   - WmSWIFTNetServer
   - WmSWIFTNetServerSamples
2. Back up the server.bat file present in the *IntegrationServer*\bin directory.
3. Back up the DLLs present in the *IntegrationServer*\lib directory, if any SAG DLLs are copied to this directory.

4. Disable the following TN document types related to the server module defined in the Trading Networks.
   - SwInt:HandleRequest
   - Sw:HandleFileRequest
   - Sw:HandleFileEventRequest
5. Delete the following processing rules related to the server module defined in the Trading Networks.
   - Process SwInt:HandleRequest
   - Process Sw:HandleFileRequest
   - Process Sw:HandleFileEventRequest
6. Shut down the Integration Server.
7. Edit the server.bat file present in the *IntegrationServer*\bin directory to remove the following environment variables:

```
ARCH=win32

GENLOG_DIR=C:\SWIFTAlliance\RA\Ra1\log

GENUTIL_DIR=C:\SWIFTAlliance\RA\bin

OSA_DIR=C:\SWIFTAlliance\RA\Ra1\log

OSA_INSTANCE=Ra1

Path=C:\SWIFTAlliance\RA\bin;C:\SWIFTAlliance\RA\lib;

PKIEXECDIR=C:\SWIFTAlliance\RA

SLP_ENV=DEFAULT

SLP_FILE=server.slp

SNL_DOMAIN_NAME=Ra1

SPK_DATA_DIR=C:\SWIFTAlliance\RA\data\pki

SWNET_BIN_PATH=C:\SWIFTAlliance\RA\Ra1\bin

SWNET_CFG_PATH=C:\SWIFTAlliance\RA\Ra1\cfg

SWNET_HOST=HOSTNAME

SWNET_HOME=C:\SWIFTAlliance\RA

SWNET_INST=Ra1

SWNET_LOG_PATH=C:\SWIFTAlliance\RA\Ra1\log

SWNET_SLP_PATH=C:\SWIFTAlliance\RA\data\

SWTRACE=C:\SWIFTAlliance\RA\Ra1\log
```

8 Delete the SAG DLLs in the *IntegrationServer*\lib directory, if any SAG DLLs are present in this directory.

9 Install the SWIFTNet Module 6.0.1 SP1 using the webMethods Installer. In the component selection list, navigate to **eStandards ▸ SWIFTNet Module 6.0.1 ▸ SWIFTNet Server**. Select **Service Pack 1** for installation. For the installation steps, see "Installing the webMethods SWIFTNet Module" on page 30.

10 Edit the env.cnf file in *IntegrationServer*\packages\WmSWIFTNetServer\config directory to specify the correct install path of the RA Client. For details, see "Step 2: Edit the webMethods Environment Configuration File" on page 41.

11 Edit the snl.cnf file in *IntegrationServer*\packages\WmSWIFTNetServer\config directory. For details, see "Step 3: Edit the webMethods SNL Configuration File" on page 42.

12 Start the Integration Server and use the webMethods SWIFTNet Module.

13 Import the TN document types and the processing rules in Trading Networks. For more information, see "Defining Trading Networks Information" on page 49. When you import the new document types, choose to overwrite the old document types of the SWIFTNet Module.

To apply the webMethods SWIFTNet Module Service Pack 1 to the Client Module

Note: Follow this set of instructions only if you are installing the Service Pack 1 over an existing installation of the SWIFTNet Module.

1 Take a backup of the following packages in the *IntegrationServer*\packages directory:

- WmSWIFTNetClient
- WmSWIFTNetClientSamples

2 Back up the server.bat file in the *IntegrationServer*\bin directory.

3 Back up the DLLs present in the *IntegrationServer*\lib directory, if any SAG DLLs are copied to this directory.

4 Shut down the Integration Server.

5 Edit the server.bat file in the *IntegrationServer*\bin directory to remove the following environment variables:

```
ARCH=win32

GENLOG_DIR=C:\SWIFTAlliance\RA\Ra1\log

GENUTIL_DIR=C:\SWIFTAlliance\RA\bin

OSA_DIR=C:\SWIFTAlliance\RA\Ra1\log

OSA_INSTANCE=Ra1
```

```
Path=C:\SWIFTAlliance\RA\bin;C:\SWIFTAlliance\RA\lib;
PKIEXECDIR=C:\SWIFTAlliance\RA
SLP_ENV=DEFAULT
SLP_FILE=server.slp
SNL_DOMAIN_NAME=Ra1
SPK_DATA_DIR=C:\SWIFTAlliance\RA\data\pki
SWNET_BIN_PATH=C:\SWIFTAlliance\RA\Ra1\bin
SWNET_CFG_PATH=C:\SWIFTAlliance\RA\Ra1\cfg
SWNET_HOST=HOSTNAME
SWNET_HOME=C:\SWIFTAlliance\RA
SWNET_INST=Ra1
SWNET_LOG_PATH=C:\SWIFTAlliance\RA\Ra1\log
SWNET_SLP_PATH=C:\SWIFTAlliance\RA\data\
SWTRACE=C:\SWIFTAlliance\RA\Ra1\log
```

6. Delete the SAG DLLs in the *IntegrationServer*\lib directory, if any SAG DLLs are present in this directory.

7. Install the SWIFTNet Module 6.0.1 SP1 using the webMethods Installer. In the component selection list, navigate to **eStandards** ▶ **SWIFTNet Module 6.0.1** ▶ **SWIFTNet Client**. Select **Service Pack 1** for installation. For the installation steps, see "Installing the webMethods SWIFTNet Module" on page 30.

8. Edit the env.cnf file in *IntegrationServer*\packages\WmSWIFTNetClient\config directory to specify the correct install path of the RA Client. For details, see "Step 2: Edit the webMethods Environment Configuration File" on page 46.

9. Start the Integration Server and use the webMethods SWIFTNet Module.

Uninstall the webMethods SWIFTNet Module

Important! This section provides only instructions that are specific to uninstalling the webMethods SWIFTNet Module. For complete instructions on using the webMethods Installer, see the *webMethods Installation Guide*.

Important! If you want to keep certain records or services that you use with the existing webMethods SWIFTNet Module packages on your webMethods Integration Server, make sure that you take a backup of the packages before performing the following procedure, which will remove all components in the packages.

To uninstall the SWIFTNet Module from webMethods Integration Server

1. Shut down the Integration Server that hosts the webMethods SWIFTNet Module.
2. In the **Add or Remove Programs** window, select **webMethods *release installation_directory*** as the program to uninstall, where ***release*** and ***installation_directory*** are the release and installation directory of the Integration Server on which the webMethods SWIFTNet Module is installed.
3. The webMethods Uninstaller starts. In the component selection list, navigate to **eStandards** and select **webMethods SWIFTNet Module** as the program to uninstall.
4. The uninstaller removes all webMethods SWIFTNet Module related files that were installed in the *IntegrationServer*\packages directory. The uninstaller does not delete the files that were created after you installed the webMethods SWIFTNet Module (for example, the user-created or the configuration files), nor does it delete the directory structure that contains these files.
5. If you do not want to save the files that are not deleted by the uninstaller, navigate to the *IntegrationServer*\packages directory and delete the related directory

To uninstall only the SWIFTNet Module Service Pack 1

Important! The following steps are valid only if you would like to uninstall only the Service Pack 1 and revert back to your previous installed version of the SWIFTNet Module. If you are a new user of SWIFTNet Module, and had directly installed the SWIFTNet Module along with the Service Pack 1, do not try to uninstall just the Service Pack 1, you must uninstall the entire the SWIFTNet Module. For more information on uninstalling the entire SWIFTNet Module, see "To uninstall the SWIFTNet Module from webMethods Integration Server" on page 36.

1. Shut down the Integration Server that hosts the webMethods SWIFTNet Module.
2. In the **Add or Remove Programs** window, select **webMethods *release installation_directory*** as the program to uninstall, where ***release*** and ***installation_directory*** are the release and

installation directory of the Integration Server on which the webMethods SWIFTNet Module Service Pack 1 is installed. The webMethods Uninstaller starts.

3 To uninstall the Service Pack 1 on the server module, in the component selection list, navigate to **eStandards** and select **webMethods SWIFTNet Module ▸ webMethods SWIFTNet Server ▸ Service Pack 1** as the program to uninstall.

4 To uninstall the Service Pack 1 on the client module, in the component selection list, navigate to **eStandards** and select **webMethods SWIFTNet Module ▸ webMethods SWIFTNet Client ▸ Service Pack 1** as the program to uninstall.

5 The uninstaller removes all webMethods SWIFTNet Module Service Pack 1 related files that were installed in the *IntegrationServer*\packages directory and reverts back to the previous installed version of the SWIFTNet Module.

6 Re-apply all the available webMethods SWIFTNet Module fixes. The patch history in the Integration Server may show that the fixes are already applied, but you need to re-apply them.

To uninstall portions of the SWIFTNet Module from webMethods Integration Server

1 Start the webMethods Integration Server and the Server Administrator.

2 In the Server Administrator, select **Package ▸ Management.**

3 From the Package list, locate the following packages:

- WmSWIFTNetClient
- WmSWIFTNetClientSample
- WmSWIFTNetServer
- WmSWIFTNetServerSample

4 Select one of the following options for each of the desired packages:

- Select **Delete** to delete the package without keeping a backup copy.
- Select **Safe Delete** to remove the package and keep a backup copy. (Backup copies are stored in the *ServerDirectory*\replicate\salvage directory on the server.)

5 Refresh your Web browser. The selected packages are removed.

CHAPTER 3

Configuring the webMethods SWIFTNet Module

Overview

This chapter describes how to prepare your server module or your client module to exchange messages and files over SWIFTNet using the SWIFTNet Module.

Important! The following steps assume that you already have installed the webMethods Integration Server, webMethods Trading Networks, and the webMethods SWIFTNet Module. For steps to install the webMethods SWIFTNet Module, see Chapter 2, "Installing the webMethods SWIFTNet Module". For more information about what SWIFT software you need, work with SWIFT to determine your software needs.

Configuring the SWIFTAlliance Gateway Server

To configure your SAG server to communicate with your RA client, SWIFTNet Module, and Integration Server, complete the following steps:

- **Install a Remote API Host Adapter (RAHA).** Install an RAHA on the same machine as the SAG server. Your RAHA enables your SAG server to exchange messages and files with the RA client on the same machine as your Integration Server. To obtain an appropriate RAHA, contact SWIFT.
- **Configure Message Partners and Endpoints**

Configure the server message partners for the server module, and the client message partners for the client module.

For more information about completing these steps, see the *SWIFTAlliance Gateway File Transfer Interface Guide*, the *SWIFTAlliance Gateway Operations Guide,* and the *SWIFTAlliance Gateway Remote API Operations Guide*.

Important! If at any time the SAG server restarts, you must reload the WmSWIFTNetServer and WmSWIFTNetClient packages.

Preparing the Server Module to Receive and Respond to Requests

To prepare your server module to receive and respond to requests using the SWIFTNet Module, you must complete the following steps:

- "Step 1: Install a Remote API Client"
- "Step 2: Edit the webMethods Environment Configuration File"
- "Step 3: Edit the webMethods SNL Configuration File"
- "Step 4: Configure Trading Networks"

Step 1: Install a Remote API Client

Install a Remote API (RA) client on the same machine as the Integration Server. The RA client enables the SWIFTNet Module to communicate with your SAG server and SNL through your Remote API Host Adapter (RAHA). To obtain an RA client, contact SWIFT.

Step 2: Edit the webMethods Environment Configuration File

Edit the Server environment configuration file, env.cnf, located in the *IntegrationServer*\packages\WmSWIFTNetServer\config directory. Set the environment variables in the env.cnf file according to the properties of the RA client installation. Following is an example of the contents of the `env.cnf` file:

```
SWNET_CFG_PATH=C:\\SWIFTAlliance\\RA\\Ra1\\cfg\\
SystemRoot=C:\\WINDOWS
SWNET_BIN_PATH=C:\\SWIFTAlliance\\RA\\lib
SWNET_HOME=C:\\SWIFTAlliance\\RA
```

Note: Use "\\" as the path separator instead of "\".

Set this parameter...	To...
`SWNET_CFG_PATH`	cfg folder of the RA instance in your system
`SystemRoot`	point to the correct system root folder. The value of the SystemRoot parameter depends on your operating system. Valid values: **`C:\WINNT`** `for` Windows 2000 **`C:\Windows`** for Windows 2003 or Windows XP
`SWNET_BIN_PATH`	lib folder of the RA instance in your system
`SWNET_HOME`	RA Home

Step 3: Edit the webMethods SNL Configuration File

When the Integration Server starts, the WmSWIFTNetServer package automatically registers itself with the SNL libraries on your SAG server as the server module and exchanges a series of pre-defined SNL primitives in sequence with your SNL libraries using your RA client. The information in the *IntegrationServer*\packages\WmSWIFTNetServer\config\snl.cnf file is required to populate these primitives. Following is an example of the contents of the server snl.cnf file:

```
# Properties for FileAct/InterAct Server
SAGMessagePartner=[MessagePartner]
server_pki_profile=[username]
server_pki_password=
Sign=TRUE
Decrypt=FALSE
Authorisation=TRUE
encryptDN=cn=[encryptCN], o=[bic], o=swift
AllFileEvents=TRUE
FullFileStatus=TRUE
SwEventEP=File_Status_Event_EP
ReceptionFolder=C:/tmp/
SwTransferEP=

#allowable values are 'Automatic' and 'Manual'
cryptoMode=Automatic
```

For the SWIFTNet Module to exchange information with your SAG server, you must set the parameters in this file using the information you used to configure your SAG server in "Configuring the SWIFTAlliance Gateway Server" on page 40 and as indicated in the following table.

Set this parameter...	To...
SAGMessagePartner	The "Server" message partner defined in your SAG server.
server_pki_profile	The User name of the server PKI profile defined in your SAG server.
server_pki_password	The password associated with the user name of the server PKI profile defined in your SAG server. This is used to unlock the Server PKI profile.
Sign, *Decrypt*, and *Authorisation*	The value is used to populate the SwSec:CreateContextRequest primitive exchanged during server initialization. Valid values: `True` and `False`.
encryptDN	The Distinguished Name to be used for encryption operations.
AllFileEvents and *FullFileStatus*	The value is used to populate the Sw:SubscribeFileEventRequest primitive exchanged during server initialization. Valid values: `True` and `False`.

Set this parameter...	To...
SwEventEP	The file transfer event end point to which file transfer events are are posted by your SAG server during FileAct operations. This value is used to populate the Sw:SubscribeFileEventRequest primitive exchanged during server initialization.
ReceptionFolder	The default folder to receive incoming files. When the ReceptionFolder parameter is blank in the snl.cnf file, the default value is *IntegrationServer*\packages\WmSWIFTNetServer\pub\SWIFTNetReceptionFolder. If the specified folder does not exist, it is created. Use “\\” or “/” as the path separator instead of “\”.
SwTransferEP	Default transfer endpoint of the remote file handler. If SwTransferEP is specified, the value must match a remote file handler endpoint running on the same machine as the Integration Server. For information on invoking the remote file handler, see “Invoking the Remote File Handler” on page 47.
cryptoMode	The value specifies whether your SAG server automatically performs crypto operations. Valid values: `Automatic` and `Manual`

Important! In the server snl.cnf file, if the value of the ReceptionFolder parameter or SwTransferEP parameter is changed, execute the wm.swiftnet.server.property:reloadProperties service. If any of the other parameter values are changed, reload the WmSWIFTNetServer package.

Step 4: Configure Trading Networks

To configure your Trading Networks, start the Integration Server on which you have installed the SWIFTNet Module and complete the following steps:

Step	Description
1	“Define Trading Partner Profiles”
2	“Define TN Document Types”
3	“Create Mapping Services”
4	“Define Processing Rules”

Define Trading Partner Profiles

In the webMethods Trading Networks Console, you define the trading partner profiles for yourself and all financial institutions with whom you want to exchange messages and files.

For more information about defining trading partner profiles for use with the SWIFTNet Module, see Chapter 4, "Defining Trading Networks Information".

Define TN Document Types

TN document types are definitions that tell webMethods Trading Networks how to identify the incoming SNL request primitives, and specify the processing rule you want to use to process the document. You must create a TN document type for each type of request you will be handling.

webMethods SWIFTNet Module provides sample TN document types in the *IntegrationServer*\packages\WmSWIFTNetServerSample\config\ServerDocTypes.dat file. You can import these TN document types into Trading Networks and then modify them or use them to create new TN document types.

For information about importing and creating TN document types, see Chapter 4, "Defining Trading Networks Information". For general information about using TN document types, see the *webMethods Trading Networks--Building Your Trading Network* guide.

Create Mapping Services

A mapping service defines what response to send back to the client module. You must create a mapping service for each type of request you will be handling.

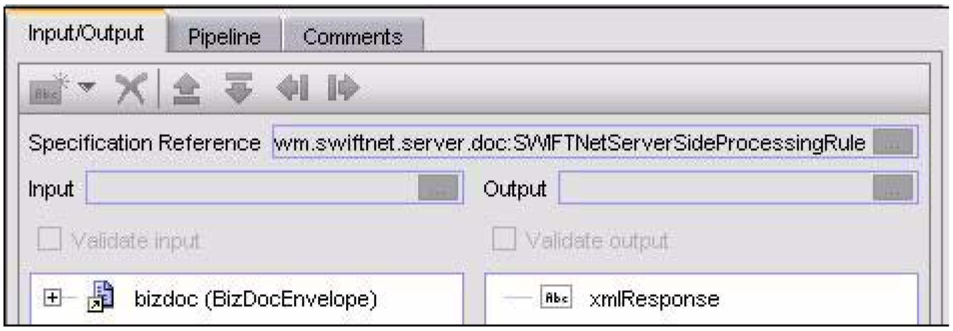
The webMethods SWIFTNet Module provides sample services in the *IntegrationServer*\packages\WmSWIFTNetServerSample directory that you can use or on which you can base new services.

Define Processing Rules

Processing rules enable you to process incoming requests. You assign a processing rule to a particular TN document type so that when Trading Networks identifies a request, it processes that request according to the settings in the processing rule, including which processing service should be invoked. You must create a processing rule for each type of request you will be handling.

All the services executed by the processing rules you created must meet the following requirements:

- The input/output signature must conform to the wm.swiftnet.server.doc:SWIFTNetServerSideProcessingRule specification in the WmSWIFTNetServer package. To do so, in Developer specify the SWIFTNetServerSideProcessingRule service in the **Specification Reference** field on the **Input/Output** tab of the service invoked by the processing rule as illustrated in the following figure.



- The output must contain a string *xmlResponse*. This is the response to the incoming request that the server module sends back to the client module.

The SWIFTNet Module provides sample processing rules in the *IntegrationServer*\packages\WmSWIFTNetServerSample\config\ServerProcessingRules.dat file. You can import these processing rules into Trading Networks and then modify them or use them to create new processing rules.

For information about importing and creating processing rules, see Chapter 4, "Defining Trading Networks Information". For general information about using processing rules, see the *webMethods Trading Networks--Building Your Trading Network* guide.

You are now ready to receive requests and send responses.

Preparing the Client Module to Send Requests and Receive Responses

To prepare your client module to exchange messages and files over SWIFTNet using the webMethods SWIFTNet Module, you must complete the following steps:

- "Step 1: Install a Remote API Client"
- "Step 2: Edit the webMethods Environment Configuration File"
- "Step 3: Invoke wm.swiftnet.client.services:swArguments"

Step 1: Install a Remote API Client

Install a Remote API (RA) client on the same machine as the Integration Server. The RA client enables the SWIFTNet Module to communicate with the SAG server and SNL through the Remote API Host Adapter (RAHA). To obtain an RA client, contact SWIFT.

Step 2: Edit the webMethods Environment Configuration File

Edit the Client environment configuration file, `env.cnf`, located in the *IntegrationServer*\packages\WmSWIFTNetClient\config directory. Set the environment variables in the env.cnf file according to the properties of RA client installation.

```
SWNET_CFG_PATH=C:\\SWIFTAlliance\\RA\\Ra1\\cfg\\
SystemRoot=C:\\WINDOWS
SWNET_BIN_PATH=C:\\SWIFTAlliance\\RA\\lib
SWNET_HOME=C:\\SWIFTAlliance\\RA
```

Note: Use "\\" as the path separator instead of "\".

Set this parameter...	To...
`SWNET_CFG_PATH`	cfg folder of the RA instance in your system
`SystemRoot`	The SystemRoot parameter should point to the correct system root folder. The value of the SystemRoot parameter depends on your operating system. Valid values: **`C:\WINNT`** for Windows 2000 **`C:\Windows`** for Windows 2003 or Windows XP
`SWNET_BIN_PATH`	lib folder of the RA instance in your system
`SWNET_HOME`	RA Home

Step 3: Invoke wm.swiftnet.client.services:swArguments

Before exchanging any primitives with the SAG server using the SWIFTNet Module, the client module must invoke the wm.swiftnet.client.services:swArguments service in the WmSWIFTNetClient package at least once. The only parameter required to be passed to this service is *SAGMessagePartner*, which is the message partner defined as the “Client” in your SAG server during configuration. The format in which this argument must be passed is illustrated in the wm.swiftnet.client.sample.fileAct:swExchangeFile service in the WmSWIFTNetClientSample package.

Invoking the Remote File Handler

You must invoke the Remote File Handler to enable transfer of files residing in your system.

To run the Remote File Handler

1 Run the following command:

```
RA_Installation_Directory\RA\bin\swiftnet.bat init -S ra_instance
```

where ra_*instance* is the instance of the Remote API on your system. For example, RA1.

2 Start the `swfa_handler` with the command-line arguments as follows:

```
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
```

Note: The `swfa_handler` is present in the *RA_HOME*\bin directory. `snlhost` is the host where SAG/SNL is installed.

webMethods.

CHAPTER 4

Defining Trading Networks Information

Overview

Trading partner profiles help define how you and your trading partners exchange SWIFT messages and files. TN document types enable webMethods Trading Networks to identify a type of business document and specify what to extract from the business document. Processing rules tell Integration Server how to process the incoming requests for server applications.

This chapter provides you with information about defining trading partner profiles, TN document types, and processing rules in webMethods Trading Networks.

Defining Trading Partner Profiles

A trading partner is any person or organization with whom you want to conduct business electronically. In the webMethods SWIFTNet Module, a trading partner is defined by several criteria that you specify in a trading partner profile, including company name and identifying information, contact information, and preferred delivery methods.

In addition to specifying trading partner profiles for all of your trading partners, you must specify a profile for your own organization.

Why Are Trading Partner Profiles Important?

Your trading partner profiles define how you exchange messages and files with your partners. In fact, the concise definition of profiles and the configuration of processing rules enable you to interact successfully with your trading partners.

For the SWIFTNet Module, you define a single trading partner profile for yourself (My Enterprise). You also must define a trading partner profile for each trading partner with whom you will be exchanging messages and files.

Defining Your Enterprise Profile

Before defining your trading partner profiles in Trading Networks and exchanging messages and files, you first must define your enterprise (My Enterprise) profile by completing the fields in the Profile Assistant in the Trading Networks Console.

For procedural information about defining your enterprise profile as well as descriptions of all the fields you must complete when defining your enterprise profile, see the *webMethods Trading Networks--Building Your Trading Network* guide.

Defining Trading Partner Profiles

Each trading partner with whom you want to exchange messages and files must have a trading partner profile in Trading Networks. After you have defined your enterprise profile, you are ready to define your trading partners' profiles.

You define a trading partner profile by completing the fields in the Profile Assistant in the Trading Networks Console.

For procedural information about defining a trading partner profile as well as descriptions of the fields you must complete when defining a trading partner profile, see the *webMethods Trading Networks--Building Your Trading Network* guide.

Defining TN Document Types and Attributes

TN document types are definitions that tell webMethods Trading Networks how to identify the incoming SNL request primitives and specify the processing rule you want to use to process the document. You must create a TN document type for each type of request you need to handle.

When the webMethods SWIFTNet Module receives a request, it invokes a Trading Networks service to recognize the incoming requests (SWIFTNet primitives) for server applications using the TN document types that you created. When Trading Networks recognizes the TN document type of the incoming request, Trading Networks extracts specific pieces of information from the business document based on the attributes specified in the TN document type.

The SWIFTNet Module provides sample TN document types in the *IntegrationServer*\packages\WmSWIFTNetServerSample\config\ServerDocTypes.dat file. You can import the following TN document types into Trading Networks and then modify them or create new TN document types:

- Sw:HandleFileRequest
- SwInt:HandleRequest
- Sw:HandleFileEventRequest
- Sw:HandleSnFRequest

For more information about TN document types, see the *webMethods Trading Networks--Building Your Trading Network* guide.

Importing the Sample TN Document Attributes

To import the sample TN attributes

1. Start the Integration Server, Server Administrator, and Trading Networks Console, if they are not already running.
2. In the Trading Networks Console, click **File** ▸ **Import**.
3. To select the TN document attributes file, click .
4. Navigate to the *IntegrationServer*\packages\WmSWIFTNetServerSample\config\ServerAttributes.dat file, and click **Open**. The TN attributes are listed on the **Import Data** screen.
5. Click to select the sample TN document attribute, SessionID, and then click **OK**. The TN document attribute is imported.
6. Now you can copy and modify the sample TN document attributes as necessary.

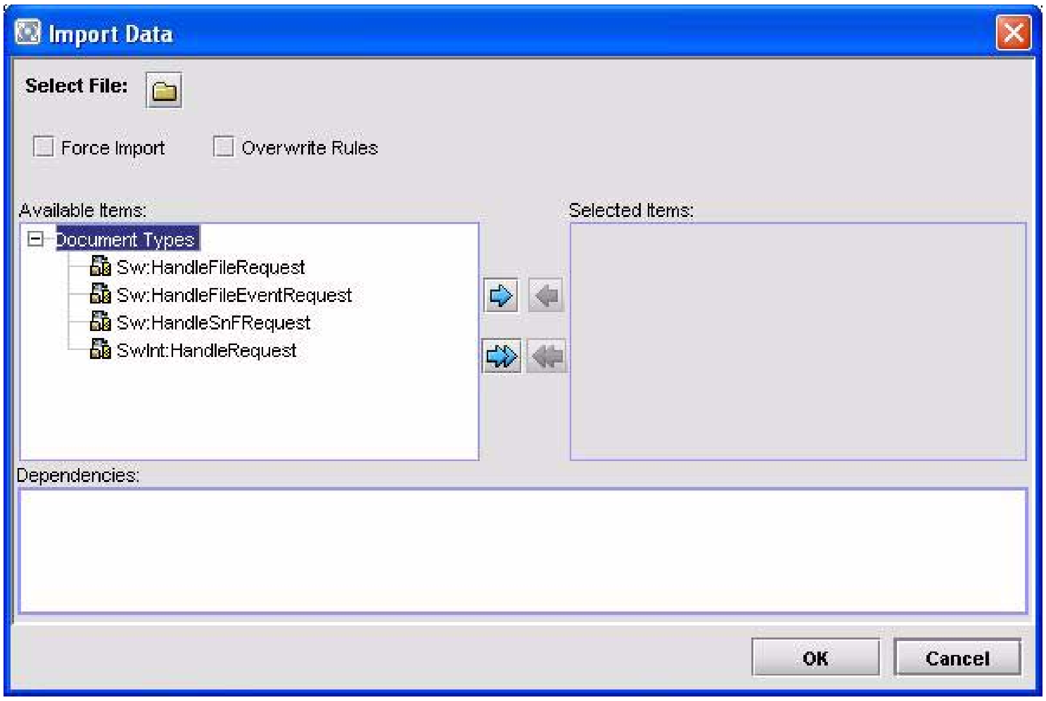
The TN document attributes are used in our sample implementation to differentiate between real-time and Store-and-Forward services. For more information about TN document attributes, see the *webMethods Trading Networks--Building Your Trading Network* guide.

Importing the Sample TN Document Types

To import the sample TN document types

1. Start the Integration Server, Server Administrator, and Trading Networks Console, if they are not already running.
2. In the Trading Networks Console, click **File** ▸ **Import**.
3. To select the TN document type file, click .

4 Navigate to the *IntegrationServer*\packages\WmSWIFTNetServerSample\config\ServerDocTypes.dat file, and click Open. The TN document types are listed on the Import Data screen.



5 Click ⇒ to select all of the TN document types and then click OK. The TN document types are imported.

6 Now you can copy and modify these TN document types as necessary.

For more information about TN document types, see the *webMethods Trading Networks--Building Your Trading Network* guide.

Creating TN Document Types

When you define an internal TN document type, you specify the root tag from within the SNL primitive that the TN document type is to match.

To create an internal TN document type

1 Start the Integration Server, Server Administrator, and Trading Networks Console, if they are not already running.

2 Select View ▸ Document Types.

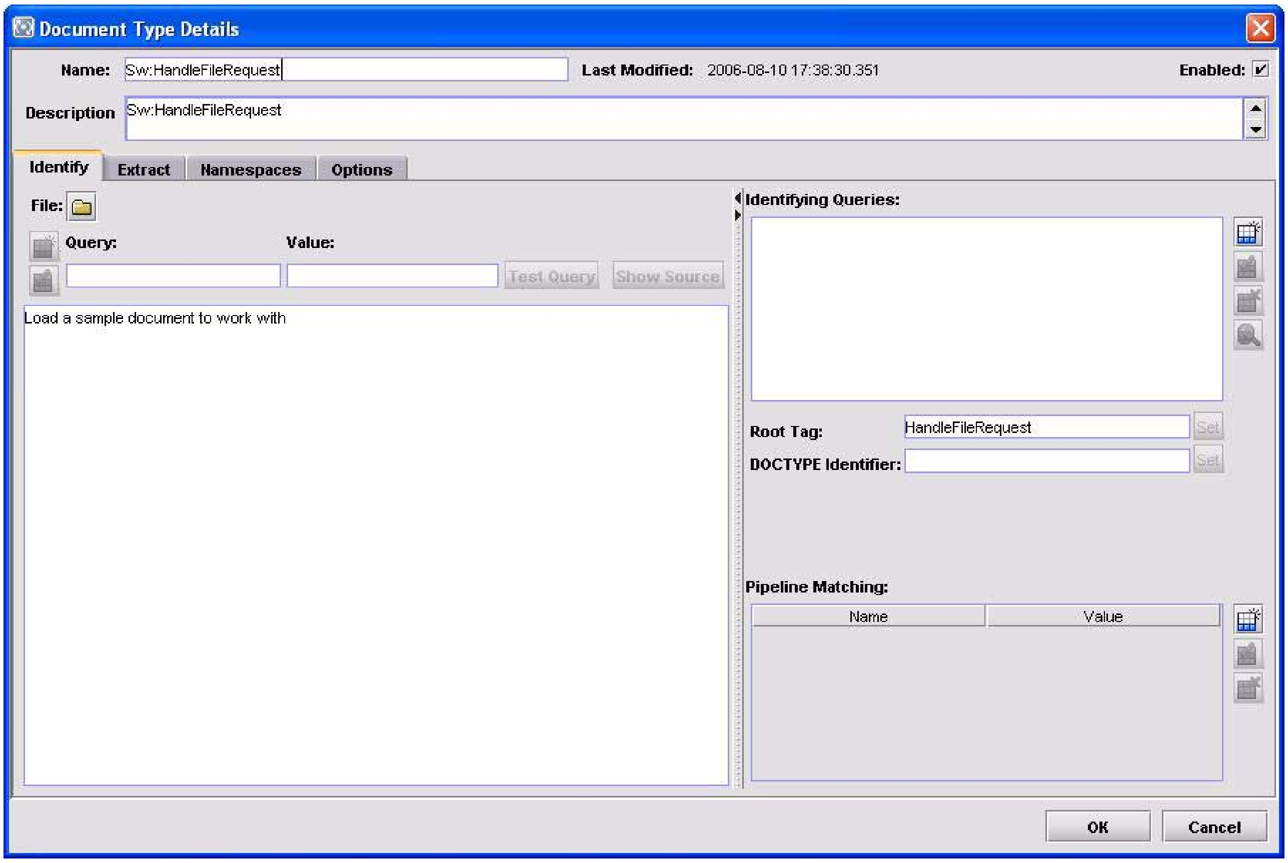
3 Select Types ▸ New ▸ XML.

4 In the **Document Type Details** screen, in the **Name** field, type the name you want to give to the internal TN document type.

5 In the **Description** field, type a description for the internal TN document type.

6 On the **Identify** tab, in the **Root Tag** field, type the value of the root tag of your internal document. For example, `HandleRequest`.

The following figure illustrates the **Identify** tab of the **Document Type Details** screen with the necessary fields completed.



Note: You can tell whether a TN document type is internal or external because an external TN document type always has a pipeline matching variable of processVersion. The TN document type in the preceding figure has no pipeline matching variable, so it is an internal TN document type.

7 Click **OK**.

For more information about TN document types, see the *webMethods Trading Networks--Building Your Trading Network* guide.

Defining Processing Rules

Processing rules process the incoming requests (SNL primitives) for server applications by invoking the mapping services that you create. The webMethods SWIFTNet Module provides server sample processing rules in the *IntegrationServer*\ packages\ WmSWIFTNetServerSample\config\ServerProcessingRules.dat file. You can import the following processing rules in this file using the Trading Networks Console and then modify them or create new processing rules:

- Process Sw:HandleFileRequest
- Process Sw:HandleFileRequest SnF
- Process Sw:HandleFileEventRequest
- Process SwInt:HandleRequest
- Process SwInt:HandleRequest SnF
- Process Sw:HandleSnFRequest

Importing the Sample Processing Rules

To import the processing rules

1. Start the Integration Server, Server Administrator, and Trading Networks Console, if they are not already running.
2. In Trading Networks, select **File ▸ Import**.
3. In the **Import** dialog box, click 📁 to select the processing rule file.
4. Navigate to the *IntegrationServer*\packages\WmSWIFTNetServerSample\config\ ServerProcessingRules.dat file, and click **Open**. The processing rules are listed on the **Import Data** screen.
5. Click ⏩ to select all of the processing rules and then click **OK**. The processing rules are imported. If the default processing rule appears above these rules, disable the default processing rule.

Creating Processing Rules

When you define a processing rule, you specify the service to be invoked to send the appropriate response back to the client application.

To create a processing rule

1 Start the Integration Server, Server Administrator, and Trading Networks Console, if they are not already running.

2 Select View ▶ Processing Rules.

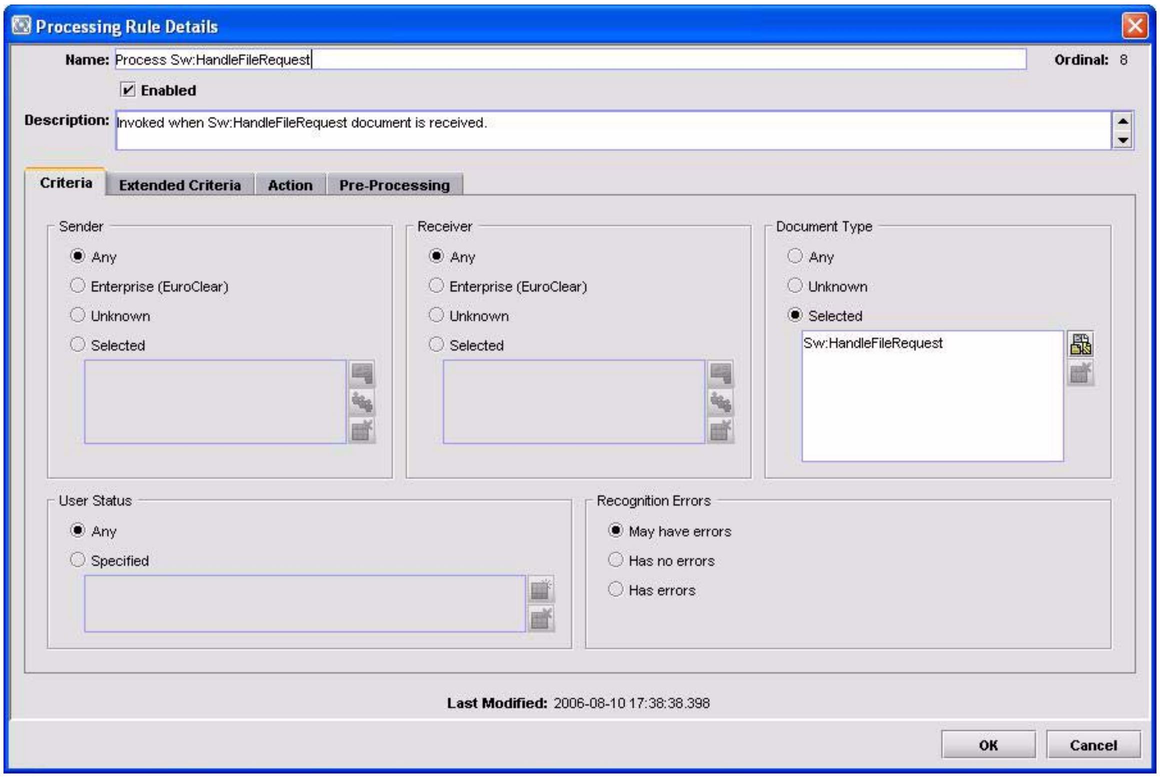
3 Select Rules ▶ Add ▶ Above/Below/Last.

4 In the Processing Rule Details screen, in the Name field, type the name you want to give to the processing rule.

5 In the Description field, type a description for the processing rule.

6 On the Criteria tab, select the TN document type associated with the processing rule.

The following figure illustrates the main screen and the Criteria tab of the Processing Rule Details screen with the necessary fields completed.



Name: Process Sw:HandleFileRequest
Ordinal: 8
☑ Enabled
Description: Invoked when Sw:HandleFileRequest document is received.

Sender: [✓ Any] Enterprise (EuroClear) Unknown Selected

Receiver: [✓ Any] Enterprise (EuroClear) Unknown Selected

Document Type: Any Unknown [✓ Selected]
Sw:HandleFileRequest

User Status: [✓ Any] Specified

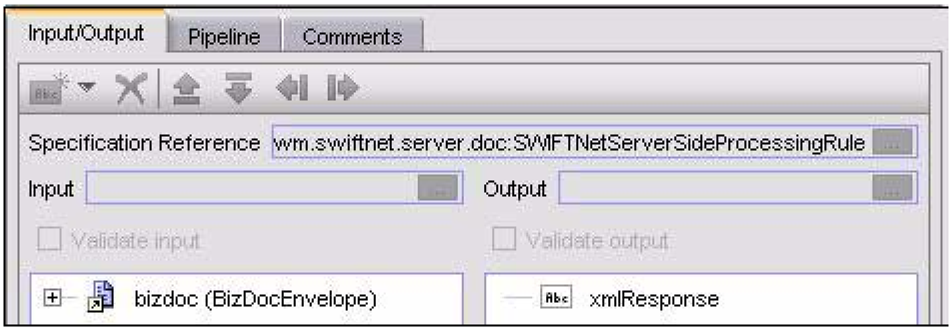
Recognition Errors: [✓ May have errors] Has no errors Has errors

Last Modified: 2006-08-10 17:38:38.398

7 On the **Action** tab, select the **Execute a Service** check box and specify the service that you want this processing rule to execute.

To function properly, the service you specify for a processing rule must meet the following requirements:

- The input/output signature must conform to the specification wm.swiftnet.server.doc:SWIFTNetServerSideProcessingRule in the WmSWIFTNetServer package. To do so, in Developer specify the SWIFTNetServerSideProcessingRule service in the **Specification Reference** field on the **Input/Output** tab of the mapping service, as illustrated in the following figure:



Specification Reference: wm.swiftnet.server.doc:SWIFTNetServerSideProcessingRule

Input

Output

bizdoc (BizDocEnvelope)

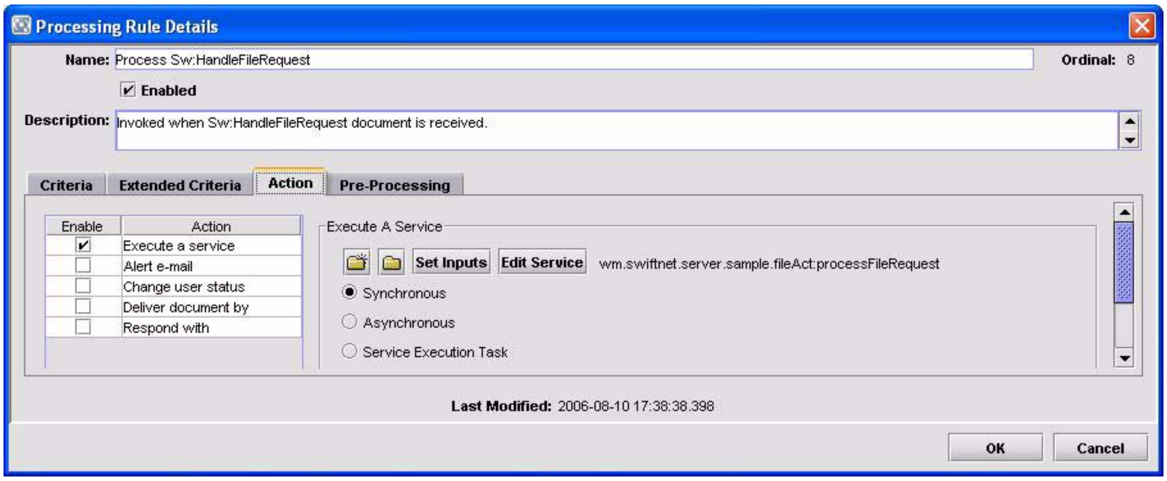
xmlResponse

- The output must contain a string *xmlResponse*. This is the response to the incoming request that the server application sends back to the client application.

The SWIFTNet Module provides sample services in the `IntegrationServer`\packages\WmSWIFTNetServerSample directory that you can use or on which you can base the new services.

8 Click OK.

The following figure illustrates the main screen and the Action tab of the Processing Rule Details screen with the necessary fields completed.


Processing Rule Details

Name: Process Sw:HandleFileRequest   Ordinal: 8

[✓] Enabled

Description: Invoked when Sw:HandleFileRequest document is received.

Enable	Action
✓	Execute a service
	Alert e-mail
	Change user status
	Deliver document by
	Respond with

Execute A Service

wm.swiftnet.server.sample.fileAct:processFileRequest

[● Synchronous]
Asynchronous
Service Execution Task

Last Modified: 2006-08-10 17:38:38.398

CHAPTER 5

webMethods SWIFTNet Module Services

Important! Because client and server applications cannot co-exist in the same process, if you want to use both the client and server application services, you must install the webMethods SWIFTNet Module on two different Integration Servers. You must use only the WmSWIFTNetClient packages on one Integration Server and only the WmSWIFTNetServer packages on the other Integration Server.

Services and the SNL Request and Response Primitives

The webMethods SWIFTNet Module services make calls to the following SNL request and response primitives that are involved in communication between the client module, the server module, and SWIFTNet:

Sw:ExchangeFileRequest	Sw:ExchangeFileResponse
Sw:ExchangeSnFRequest	Sw:ExchangeSnFResponse
Sw:FetchFileRequest	Sw:FetchFileResponse
Sw:HandleFileEventRequest	Sw:HandleFileEventResponse
Sw:HandleFileRequest	Sw:HandleFileResponse
Sw:HandleInitRequest	Sw:HandleInitResponse
Sw:HandleSnFRequest	Sw:HandleSnFResponse
Sw:InitRequest	Sw:InitResponse
Sw:PullSnFRequest	Sw:PullSnFResponse
SwSec:CreateContextRequest	SwSec:CreateContextResponse
SwSec:DestroyContextRequest	SwSec:DestroyContextResponse
Sw:SubscribeFileEventRequest	Sw:SubscribeFileEventResponse
Sw:TermRequest	Sw:TermResponse
SwInt:ExchangeRequest	SwInt:ExchangeResponse
SwInt:HandleRequest	SwInt:HandleResponse
SwInt:SendRequest	SwInt:SendResponse
SwInt:WaitRequest	SwInt:WaitResponse

Services Quick Reference

The following table provides a list of all services, by package, that are available for use in webMethods SWIFTNet Module.

Package and Service Name	Description	Page
WmSWIFTNetClient package		
wm.swiftnet.client.init:printRemoteErrors	Logs the standard output and standard error from the client process connected to the SAG server. Errors are logged to the Integration Server console.	64
wm.swiftnet.client.init:shutdown	Terminates the client process connected to the SAG server.	65
wm.swiftnet.client.init:startup	Starts a client process that connects to the SAG server.	65
wm.swiftnet.client.services:createContextRequest	Sends SwSec:CreateContextRequest to the SAG server over RA to create a security context.	65
wm.swiftnet.client.services:destroyContextRequest	Sends SwSec:DestroyContextRequest to the SAG server over RA to destroy a security context.	66
wm.swiftnet.client.services:exchangeFileRequest	Sends Sw:ExchangeFileRequest to the SAG server over RA to perform a FileAct operation (a get file or a put file).	66
wm.swiftnet.client.services:exchangeRequest	Sends SwInt:ExchangeRequest to the SAG server over RA to send an InterAct message.	67
wm.swiftnet.client.services:exchangeSnFRequest	Sends Sw:ExchangeSnFRequest to the SAG server over RA to send an SnF message.	67
wm.swiftnet.client.services:fetchFileRequest	Sends Sw:FetchFileRequest to the SAG server over RA to fetch a file from SnF queue.	68
wm.swiftnet.client.services:getFileStatusRequest	Sends Sw:GetFileStatusRequest to the SAG server over RA to get the status of a file transfer.	68
wm.swiftnet.client.services:initRequest	Sends Sw:InitRequest to the SAG server over RA.	69

Package and Service Name	Description	Page
wm.swiftnet.client.services:pullSnFRequest	Sends Sw:PullSnFRequest to the SAG server over RA to pull an SnF message.	69
wm.swiftnet.client.services:sendRequest	Sends SwInt:SendRequest to the SAG server over RA to send an asynchronous InterAct message.	70
wm.swiftnet.client.services:sendSynchronousRequest	Converts the request in an IS Document to an XML string format (primitive) and sends it to the SAG server via swCall.	70
wm.swiftnet.client.services:signEncryptRequest	Sends SwSec:SignEncryptRequest to the SAG server over RA to sign and/or encrypt payload.	71
wm.swiftnet.client.services:swArguments	Initializes client application by sending arguments to the SNL libraries.	71
wm.swiftnet.client.services:swCall	Invokes the “SwCall()” function in the SNL libraries to send a request primitive to the SAG server	72
wm.swiftnet.client.services:termRequest	Sends Sw:TermRequest to the SAG server over RA to terminate a session.	72
wm.swiftnet.client.services:verifyDecryptRequest	Sends SwSec:VerifyDecryptRequest to the SAG server over RA to verify a signed/encrypted message.	73
wm.swiftnet.client.services:waitRequest	Sends SwInt:WaitRequest to the SAG server over RA to retrieve response asynchronously.	73
wm.swiftnet.client.util:formatXML	Formats an XML string by appending the proper namespace after the root tag.	74
WmSWIFTNetServer package		
wm.swiftnet.server.init:printRemoteErrors	Logs the standard output and standard error from the server process connected to the SAG server. Errors are logged to the Integration Server console.	75
wm.swiftnet.server.init:shutdown	Terminates the server process connected to the SAG server.	75
wm.swiftnet.server.init:startup	Starts a server process that connects to the SAG server.	76

Package and Service Name	Description	Page
wm.swiftnet.server.property:getCommonProperties	Retrieves the most commonly used properties from the *webMethods6\IntegrationServer\* packages\WmSWIFTNetServer\ config\snl.cnf file.	76
wm.swiftnet.server.property:getProperty	Retrieves the value of the specified property from the *webMethods6\IntegrationServer\* packages\WmSWIFTNetServer\ config\snl.cnf file.	77
wm.swiftnet.server.property:listProperties	Retrieves all properties specified in the *webMethods6\IntegrationServer\* packages\WmSWIFTNetServer\ config\snl.cnf file.	77
wm.swiftnet.server.property:reloadProperties	Reloads all properties specified in the *webMethods6\IntegrationServer\* packages\WmSWIFTNetServer\ config\snl.cnf file.	77
wm.swiftnet.server.services:handleRequest	Processes incoming requests from the SAG server as a TN document type and invokes the processing rule specified by the user.	78
wm.swiftnet.server.services:swCall	Invokes the “SwCall()” function in the SNL libraries to send a request primitive to the SAG server.	78
wm.swiftnet.server.util:formatXML	Formats an XML string by appending the proper namespace after the root tag.	79

WmSWIFTNetClient Package

This package contains the elements (flow services, Java services, record descriptions, and wrapper services) that support webMethods SWIFTNet Module client-side functionality. This package contains the following folders:

Folder	Description
wm.swiftnet.client.doc	This folder contains the NS records that represent the SNL primitives exchanged for FileAct and InterAct operations.
wm.swiftnet.client.init	This folder contains services that start and terminate the client process.
wm.swiftnet.client.services	This folder contains services that exchange SNL primitives with the SAG server over RA. The wm.swiftnet.client.services:swArguments service must be invoked prior to invoking any other services in this folder. The services in this folder can be invoked in a predefined sequence to perform FileAct and InterAct realtime and SnF operations. In essence the services in this folder are the building blocks to perform higher level FileAct and InterAct operations.
wm.swiftnet.client.util	This folder contains utility services.

wm.swiftnet.client.init:printRemoteErrors

This service logs the standard output and standard error from the client process that is connected to the Swift Alliance Gateway (SAG) server. Errors are logged to the Integration Server console. This service should be used to trace an error and should not be used otherwise.

Input Parameters

None.

Output Parameters

None.

wm.swiftnet.client.init:shutdown

This service is registered as a shutdown service for the WmSWIFTNetClient package. It terminates the client process that is connected to the SWIFT Alliance Gateway (SAG) server.

Input Parameters

None.

Output Parameters

None.

wm.swiftnet.client.init:startup

This service starts a client process that connects to the Swift Alliance Gateway (SAG) server. This client process connects to the SAG server whenever a request needs to be sent over SWIFTNet.

Input Parameters

None.

Output Parameters

None.

wm.swiftnet.client.services:createContextRequest

This service requests the SAG server to create a security context.The service sends the SwSec:CreateContextRequest to the SAG server over RA and returns the SwSec:CreateContextResponse received from the SAG server.

Input Parameters

SwSecCreateContextRequest	IData Request to create a security context.

Output Parameters

SwSecCreateContextResponse	IData Response indicating success or failure of security context creation in the SAG server.
error	String Whether an error occurred. Valid values: `true` and `false`.
errorXMLString	String Optional. Error details received from the SAG server.

wm.swiftnet.client.services:destroyContextRequest

This service requests the SAG server to destroy a security context. The service sends the SwSec:DestroyContextRequest to the SAG server over RA and returns the SwSec:DestroyContextResponse received from the SAG server.

Input Parameters

SwSecDestroyContextRequest	IData Request to destroy a security context.

Output Parameters

SwSecDestroyContextResponse	IData Response indicating success or failure of security context destruction in the SAG server.
error	String Whether an error occurred. Valid values: `true` and `false`.
errorXMLString	String Optional. Error details received from the SAG server.

wm.swiftnet.client.services:exchangeFileRequest

This service requests the SAG server to perform a FileAct operation (realtime get file or put file and SnF put file). The information whether to put a file or get a file is specified in the Sw:ExchangeFileRequest primitive. The service sends the Sw:ExchangeFileRequest to the SAG server over RA and returns the Sw:ExchangeFileResponse received from the SAG server.

Input Parameters

SwExchangeFileRequest	IData Request to perform a FileAct operation.

Output Parameters

SwExchangeFileResponse	IData Response indicating success or failure of the FileAct operation.
error	String Whether an error occurred. Valid values: `true` and `false`.
errorXMLString	String Optional. Error details received from the SAG server.

wm.swiftnet.client.services:exchangeRequest

This service requests the SAG server to send an InterAct message. The service sends the SwInt:ExchangeRequest to the SAG server over RA and returns the SwInt:ExchangeResponse received from the SAG server.

Input Parameters

SwIntExchangeRequest	IData Request to exchange a synchronous request.

Output Parameters

SwIntExchangeResponse	IData Synchronous response received.
error	String Whether an error occurred. Valid values: `true` and `false`.
errorXMLString	String Optional. Error details received from the SAG server.

wm.swiftnet.client.services:exchangeSnFRequest

This service requests the SAG server to send an SnF message. The service sends the Sw:ExchangeSnFRequest to the SAG server over RA and returns the Sw:ExchangeSnFResponse received from the SAG server.

Input Parameters

SwExchangeSnFRequest	IData Request related to SnF protocol. For example, acquire a queue.

Output Parameters

SwExchangeSnFResponse	IData Response returned by SWIFTNet.
error	String Whether an error occurred. Valid values: `true` and `false`.
errorXMLString	String Optional. Error details received from the SAG server.

wm.swiftnet.client.services:fetchFileRequest

This service sends the Sw:FetchFileRequest to the SAG server over RA and returns the Sw:FetchFileResponse received from the SAG server.

Input Parameters

SwFetchFileRequest	IData Request to fetch a file from an SnF queue.

Output Parameters

SwFetchFileResponse	IData Response returned by the SWIFTNet for a fetch file request.
error	String Whether an error occurred. Valid values: `true` and `false`.
errorXMLString	String Optional. Error details received from the SAG server.

wm.swiftnet.client.services:getFileStatusRequest

This service sends the Sw:GetFileStatusRequest to the SAG server over RA and returns the Sw:GetFileStatusResponse received from the SAG server.

Input Parameters

SwGetFileStatusRequest	IData Request to get the status of a file transfer.

Output Parameters

SwGetFileStatusResponse	IData File transfer status response.
error	String Whether an error occurred. Valid values: `true` and `false`.
errorXMLString	String Optional. Error details received from the SAG server.

wm.swiftnet.client.services:initRequest

This service sends the Sw:InitRequest to the SAG server over RA and returns the Sw:InitResponse received from the SAG server. This is the initialization primitive exchanged before any other primitives are exchanged.

Input Parameters

SwInitRequest	IData Initialization request primitive.

Output Parameters

SwInitResponse	IData Initialization primitive response.
error	String Whether an error occurred. Valid values: `true` and `false`.
errorXMLString	String Optional. Error details received from the SAG server.

wm.swiftnet.client.services:pullSnFRequest

This service sends the Sw:PullSnFRequest to the SAG server over RA and returns the Sw:PullSnFResponse received from the SAG server.

Input Parameters

SwPullSnFRequest	IData Request to pull a message from SnF queue.

Output Parameters

SwPullSnFResponse	IData Response returned by SWIFTNet for a pull request.
error	String Whether an error occurred. Valid values: `true` and `false`.
errorXMLString	String Optional. Error details received from the SAG server.

wm.swiftnet.client.services:sendRequest

This service sends the SwInt:SendRequest to the SAG server over RA and returns the SwInt:SendResponse received from the SAG server. This is the asynchronous version of SwInt:ExchangeRequest primitive.

Input Parameters

SwIntSendRequest	IData Asynchronous request primitive.

Output Parameters

SwIntSendResponse	IData Immediate response received from the SAG server, without waiting for the actual response from SWIFTNet.
error	String Whether an error occurred. Valid values: `true` and `false`.
errorXMLString	String Optional. Error details received from the RA client.

wm.swiftnet.client.services:sendSynchronousRequest

This service formats the input request primitive into an XML string and then invokes the wm.swiftnet.client.services:swCall service to send the request primitive to the SAG server over RA. The response XML string received is then formatted into the appropriate response primitive.

Input Parameters

requestDocument	IData Request primitive to be sent.
requestDocNSName	String NS record name of request primitive.
responseDocNS Name	String NS record name of response primitive.

Output Parameters

responseDocument	IData Response primitive received.
error	String Whether an error occurred. Valid values: `true` and `false`.
errorXMLString	String Optional. Error details received from the RA client.

wm.swiftnet.client.services:signEncryptRequest

This service sends the SwSec:SignEncryptRequest to the SAG server over RA and returns the SwSec:SignEncryptResponse received from the SAG server.

Input Parameters

SwSecSignEncryptRequest	IData Request sign and/or encrypt payload.

Output Parameters

SwSecSignEncryptResponse	IData Response received from the SAG server.
error	String Whether an error occurred. Valid values: `true` and `false`.
errorXMLString	String Optional. Error details received from the SAG server.

wm.swiftnet.client.services:swArguments

This service initializes the client application by invoking the SwArguments() function defined in the SNL libraries. The service takes a String[ ] of arguments as input. The only mandatory parameter to be passed is the *SAGMessagePartner* defined in the SAG server.

For example:

```
args[0] = “WmSWIFTNetClient”

args[1] = “-SagMessagePartner”

args[2] = “<message partner name defined in SAG>”
```

Input Parameters

args	String Initialization arguments to be passed to the SNL libraries.

Output Parameters

None.

wm.swiftnet.client.services:swCall

This service invokes the SwCall() function in the SNL libraries to send a request primitive to the SAG server and returns the response primitive received from the SAG server.

Input Parameters

xmlRequest	String Request primitive to be sent to the SAG server.

Output Parameters

xmlResponse	String Response received from the SAG server.
error	String Whether an error occurred. Valid values: `true` and `false`.
errorXMLString	String Optional. Error details received from the SAG server.

wm.swiftnet.client.services:termRequest

This service sends the Sw:TermRequest to the SAG server over RA and returns the Sw:TermResponse received from the SAG server.

Input Parameters

SwTermRequest	IData Session termination request to the SAG server.

Output Parameters

SwTermResponse	IData Session termination response from the SAG server.
error	String Whether an error occurred. Valid values: `true` and `false`.
errorXMLString	String Optional. Error details received from the SAG server.

wm.swiftnet.client.services:verifyDecryptRequest

This service sends the SwSec:VerifyDecryptRequest to the SAG server over RA and returns the SwSec:VerifyDecryptResponse received from the SAG server.

Input Parameters

SwSecVerifyDecryptRequest	IData Request to verify a signed/encrypted message.

Output Parameters

SwSecVerifyDecryptResponse	IData Message decryption response from the SAG server.
error	String Whether an error occurred. Valid values: `true` and `false`.
errorXMLString	String Optional. Error details received from the SAG server.

wm.swiftnet.client.services:waitRequest

This service sends the SwInt:WaitRequest to the SAG server over RA and returns the SwInt:WaitResponse received from the SAG server. This is the primitive exchanged to retrieve a response asynchronously.

Input Parameters

SwIntWaitRequest	IData Request to retrieve response asynchronously.

Output Parameters

SwIntWaitResponse	IData Asynchronous response received.
error	String Whether an error occurred. Valid values: `true` and `false`.
errorXMLString	String Optional. Error details received from the SAG server.

wm.swiftnet.client.util:formatXML

This service formats an XML string by appending the following namespaces after the root tag. The namespaces are “Sw”, “SwInt”, “SwGbl” and “SwSec”. If these namespaces are not appended to the root tag, the incoming XML response primitives cannot be converted into IData objects in the Integration Server.

Input Parameters

xmlRequest	String XML string to be formatted with namespaces.

Output Parameters

formattedXML	String XML string with namespaces appended after the root tag.

WmSWIFTNetServer Package

This package contains the elements (flow services, Java services, record descriptions, and wrapper services) that support webMethods SWIFTNet Module server-side functionality. This package contains the following folders:

Folder	Description
wm.swiftnet.server.doc	This folder contains the NS records that represent the SNL primitives exchanged for FileAct and InterAct operations.
wm.swiftnet.server.init	This folder contains services that start and terminate the server process.
wm.swiftnet.server.property	This folder contains services that load properties specified in the *IntegrationServer*\packages\WmSWIFTNetServer\config\snl.cnf file.
wm.swiftnet.server.services	This folder contains services to handle incoming requests.
wm.swiftnet.server.util	This folder contains utility services.

wm.swiftnet.server.init:printRemoteErrors

This service logs the standard output and standard error from the server process connected to the Swift Alliance Gateway (SAG) server. Errors are logged to the Integration Server console. This service should be used to trace an error and should not be used otherwise.

Input Parameters

None.

Output Parameters

None.

wm.swiftnet.server.init:shutdown

This service is registered as a shutdown service for the WmSWIFTNetServer package. It terminates the server process connected to the SWIFT Alliance Gateway (SAG) server.

Input Parameters

None.

Output Parameters

None.

wm.swiftnet.server.init:startup

This service starts a server process that connects to the Swift Alliance Gateway (SAG) server. The server process is registered as the server application for the message partner specified in the *IntegrationServer* packages\WmSWIFTNetServer\config\snl.cnf file. The following primitives are exchanged with the SAG server on startup in this order:

1. Sw:HandleInitRequest
2. SwSec:CreateContextRequest
3. SwSec:CreateContextResponse
4. Sw:SubscribeFileEventRequest
5. Sw:SubscribeFileEventResponse
6. Sw:HandleInitResponse

Input Parameters

None.

Output Parameters

None.

wm.swiftnet.server.property:getCommonProperties

This service retrieves the most commonly used properties from the *webMethods6\IntegrationServer\* packages\WmSWIFTNetServer\config\snl.cnf file.

Input Parameters

None.

Output Parameters

SAGMessagePartner	String Must correspond to a “Server” type message partner defined in the SAG server.
server_pki_profile	String User name of the profile defined in the SAG server.
server_pki_password	String Password associated with the user name used to unlock the security information in the SAG server.
Sign,Decrypt and Authorization	String Values are used for populating SwSec:CreateContextRequest primitive exchanged during server initialization. Valid values: `True` and `False`.

encryptDN	String Distinguished Name to be used for encryption operations.
cryptoMode	String Specifies whether crypto operations are performed automatically by the SAG server/SNL. Valid values: `Automatic` and `Manual`.

wm.swiftnet.server.property:getProperty

This service retrieves the value of the specified property from the *IntegrationServer*\packages\WmSWIFTNetServer\config\snl.cnf file.

Input Parameters

propertyName	String Property value to be retrieved.

Output Parameters

value	String Value of the property.

wm.swiftnet.server.property:listProperties

This service retrieves all the properties specified in the *IntegrationServer*\packages\WmSWIFTNetServer\config\snl.cnf file.

Input Parameters

None.

Output Parameters

properties	IData All properties in the snl.cnf file.

wm.swiftnet.server.property:reloadProperties

This service reloads all the properties specified in the *IntegrationServer*\packages\WmSWIFTNetServer\config\snl.cnf file. This could be useful if more properties are added or existing properties have been changed and the changes need to be reflected in the Integration Server immediately.

Input Parameters

None.

Output Parameters

properties	IData All properties reloaded from the snl.cnf file.

wm.swiftnet.server.services:handleRequest

This service is invoked by the SwCallBack function in WmSWIFTNetServer.dll when a request is received from the SAG server. This service then recognizes the incoming request primitive as a TN document type and invokes the processing rule specified by the user. The output of the service specified by the user for the processing rule must contain the string variable *xmlResponse* which is then passed back to the SAG server as the response for the incoming request.

Input Parameters

xmlRequest	String Incoming request primitive.
SwSecUserDN	String User DN returned by security context created in the SAG server at startup.

Output Parameters

xmlResponse	String Outgoing response primitive.

wm.swiftnet.client.services:swCall

This service invokes the SwCall() function in the SNL libraries to send a request primitive to the SAG server. The response primitive received from the SAG server is then output to the pipeline.

Input Parameters

xmlRequest	String Request primitive to be sent to the SAG server.

Output Parameters

xmlResponse	String Response primitive received from the SAG server.
error	String Whether an error occurred. Valid values: `true` and `false`.
errorXMLString	String Optional. Error details received from the SAG server.

wm.swiftnet.server.util:formatXML

This service formats an XML string by appending the following namespaces after the root tag. The namespaces are “Sw”, “SwInt”, “SwGbl” and “SwSec”. If these namespaces are not appended to the root tag, the incoming XML response primitives cannot be converted into IData objects in the Integration Server.

Input Parameters

xmlRequest	**String** Request primitive to be sent to the SAG server.

Output Parameters

formattedXML	**String** XML string with namespaces appended after the root tag.

APPENDIX A

Samples

Overview

The webMethods SWIFTNet Module includes sample services for the server module and the client module. This appendix contains information on how to use these sample services.

webMethods SWIFTNet Module Sample Packages

The webMethods SWIFTNet Module contains the following sample packages, which contain webMethods sample services and related files, that you install on the webMethods Integration Server.

Package	Description
WmSWIFTNetClientSample	Contains sample services, TN document types, processing rules, and implementation of the SWIFTNet Module client-side functionality.
WmSWIFTNetServerSample	Contains sample services, TN document types, processing rules, and implementation of the SWIFTNet Module server-side functionality.

Preparing the SWIFTNet Server Module to Use Sample Services

To use the server sample services, ensure that you have completed the following procedures. For more information regarding configuring the SWIFTNet Module, see "Configuring the webMethods SWIFTNet Module" on page 39.

Step	Description
1	Edit the server module snl.cnf file present in the *IntegrationServer*\packages\WmSWIFTNetServer\config directory. For more information, see "Step 3: Edit the webMethods SNL Configuration File" on page 42.
2	Import the server module sample TN document types from the *IntegrationServer*\packages\WmSWIFTNetServerSample\config\ServerDocTypes.dat file. For more information, see "Importing the Sample TN Document Types" on page 52.
3	Import the server module sample processing rules from the *IntegrationServer*\packages\WmSWIFTNetServerSample\config\ServerProcessingRules.dat file. For more information, see "Importing the Sample Processing Rules" on page 55.
4	Set up aliases for remote Integration Servers. For more information, see "Setting Up Aliases for Remote Integration Servers" on page 101.

Preparing the SWIFTNet Client Module to Use Sample Services

Perform the following steps to use the client sample services. For more information regarding configuring the client module, see "Preparing the Client Module to Send Requests and Receive Responses" on page 46.

Step	Description
1	"Edit the SWIFTNet Client Module SNL Configuration File"
2	"Import the Sample TN Document Types"
3	"Import the Sample Processing Rules"

Edit the SWIFTNet Client Module SNL Configuration File

The parameters defined in the client module snl.cnf file are used by the sample services. The information required to send SNL primitives to the server module is retrieved from the *IntegrationServer*\packages\ WmSWIFTNetClientSample\config\snl.cnf file. Following is an example of the contents of the client snl.cnf file:

```
# Properties for FileAct and InterAct
SAGMessagePartner=[MessagePartner]
client_pki_profile=[username]
client_pki_password=
requestor=o=[bic], o=swift
responder=o=[bic], o=swift
service=
encryptDN=cn=[encryptCN], o=[bic], o=swift
Sign=TRUE
Decrypt=FALSE
Authorisation=TRUE
ReceptionFolder=C:/tmp/
SwTransferEP=
SwEventEP=File_Status_Event_EP
```

Set the parameters in this file as indicated in the following table.

Set this parameter...	To...
SAGMessagePartner	A "Client" message partner defined in the SAG server.
client_pki_profile	The user name of a client PKI profile defined in the SAG server.
client_pki_password	The password associated with the user name of a client PKI profile defined in the SAG server. This is used to unlock the security information in the SAG server.
requestor	The Distinguished Name of the Requestor.
responder	The Distinguished Name of the Responder.
service	The default SWIFTNet service to be used.

Set this parameter...	To...
encryptDN	The Distinguished Name to be used for encryption operations.
Sign, *Decrypt*, and *Authorisation*	The value is used to populate the SwSec:CreateContextRequest primitive exchanged during server initialization. Valid values: `True` and `False`.
ReceptionFolder	The name of the default folder to receive the incoming files. When the ReceptionFolder parameter is blank in the snl.cnf file, the default value is *webMethods6\IntegrationServer*\packages\ WmSWIFTNetClient\pub\SWIFTNetReceptionFolder. If the specified folder does not exist, it will be created. **Note:** Use “\\” or “/” as the path separator instead of “\”
SwTransferEP	Default transfer endpoint of the remote file handler. For more information, see “Invoking the Remote File Handler” on page 47.
SwEventEP	File transfer event end point where file transfer events associated with file transfers carried out while fetching the files, should be sent. When SnF Pull or Push is used, to send the file transfer events to the Integration Server of the sever module, the value specified for the client SwEventEP parameter should be the same as the SwEventEP parameter specified in the server module snl.cnf.

Important! In the client snl.cnf file, if the value of the SAGMessagePartner parameter is changed, reload the WmSWIFTNetClient package. If any of the other parameter values are changed, execute the wm.swiftnet.client.sample.property:reloadProperties service.

Import the Sample TN Document Types

To import the sample TN document types:

1 Start the Integration Server, Server Administrator, and Trading Networks Console, if they are not already running.

2 In the Trading Networks Console, click **File** ▸ **Import**.

3 To select the TN document type file, click 📁.

4 Navigate to *IntegrationServer*\packages\WmSWIFTNetClientSample\config\ClientDocTypes.dat, and click **Open**. The TN document types are listed on the **Import Data** screen.

5 Click ⇒ to select all of the TN document types and then click **OK**. The TN document types are imported.

6 Now you can copy and modify the sample TN document type, Sw:PullSnFResponse, as necessary.

For more information about TN document types, see the *webMethods Trading Networks--Building Your Trading Network* guide.

Import the Sample Processing Rules

To import the processing rules:

1. Start the Integration Server, Server Administrator, and Trading Networks Console, if they are not already running.
2. In Trading Networks, select File ▸ Import.
3. In the Import dialog box, click to select the processing rule file.
4. Navigate to *IntegrationServer*\packages\WmSWIFTNetClientSample\config\ClientProcessingRules.dat, and click Open. The processing rules are listed on the Import Data screen.
5. Click to select all of the processing rules and then click OK. The processing rules are imported.
6. Now you can copy and modify the sample processing rule, Process Sw:PullSnFResponse, as necessary.

Sample Services Quick Reference

The following table provides a list of all sample services, by package, that are available for use in webMethods SWIFTNet Module.

Package and Service Name	Description	Page
WmSWIFTNetClientSample package		
wm.swiftnet.client.sample.SnF:swAcquireSnFQueue	Constructs and sends Sw:ExchangeSnFRequest to SWIFTNet to acquire an SnF queue in push mode.	94
wm.swiftnet.client.sample.SnF:swFetchFile	Constructs and sends Sw:FetchFileRequest to SWIFTNet to fetch a file from an SnF queue.	95
wm.swiftnet.client.sample.SnF:swPullMessageOrFileSnF	Pulls and processes messages in the SnF queue, based on message type.	96
wm.swiftnet.client.sample.SnF:swReleaseSnFQueue	Constructs and sends Sw:ExchangeSnFRequest to SWIFTNet to release an SnF queue.	97
wm.swiftnet.client.sample.fileAct:swExchangeFile	Constructs and sends Sw:ExchangeFileRequest to SWIFTNet for a real-time FileAct service.	88

Package and Service Name	Description	Page
wm.swiftnet.client.sample.fileAct:swExchangeFileSnF	Constructs and sends Sw:ExchangeFileRequest to SWIFTNet for an SnF FileAct service.	89
wm.swiftnet.client.sample.fileAct:swGetFileStatus	Constructs and sends Sw:GetFileStatusRequest to SWIFTNet to obtain the status of a file transfer.	90
wm.swiftnet.client.sample.interAct:swExchangeRequest	Constructs and sends SwInt:ExchangeRequest (or SwInt:SendRequest) to SWIFTNet for a real-time InterAct service.	91
wm.swiftnet.client.sample.interAct:swExchangeRequestSnF	Constructs and sends SwInt:ExchangeRequest (or SwInt:SendRequest) to SWIFTNet for an SnF InterAct service.	92
wm.swiftnet.client.sample.property:getCommonProperties	Retrieves the most commonly used properties from the *webMethods6\IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file.	93
wm.swiftnet.client.sample.property:getProperty	Retrieves the value of the specified property from the *webMethods6\IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file.	93
wm.swiftnet.client.sample.property:listProperties	Retrieves all properties specified in the *webMethods6\IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file.	94
wm.swiftnet.client.sample.property:reloadProperties	Reloads all properties specified in the *webMethods6\IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file.	94

Package and Service Name	Description	Page
WmSWIFTNetServerSample package		
wm.swiftnet.server.sample.fileAct:processFileEventRequest	Constructs an XML response for a file event request (Sw:HandleFileEventRequest), received from SWIFTNet.	98
wm.swiftnet.server.sample.fileAct:processFileRequest	Constructs an XML response for a real-time FileAct handle request (Sw:HandleFileRequest), received from SWIFTNet.	99
wm.swiftnet.server.sample.interAct:processHandleRequest	Constructs an XML response for a real-time InterAct handle request (SwInt:HandleRequest), received from SWIFTNet.	99
wm.swiftnet.server.sample.SnF:processHandleSnFRequest	Constructs an XML response for an SnF delivery notification (Sw:HandleSnFRequest), received from SWIFTNet.	99
wm.swiftnet.server.sample.SnF:processSnFFileRequest	Constructs an XML response for an SnF FileAct handle request (Sw:HandleFileRequest), received from SWIFTNet.	100
wm.swiftnet.server.sample.SnF:processSnFHandleRequest	Constructs an XML response for an SnF InterAct handle request (SwInt:HandleRequest), received from SWIFTNet.	100

WmSWIFTNetClientSample Package

This package contains the elements for a sample implementation of the webMethods SWIFTNet Module client-side functionality.

It provides sample services for sending SNL primitives to the server application and to the SnF queue. These services call other services in the sequence required by SWIFTNet. You can use these services or use them as guides for creating your own services. This package also provides sample mapping services that map a given set of inputs into the XML primitive required by SWIFTNet.

This package contains the following folders:

Folder	Description
wm.swiftnet.client.sample.fileAct	This folder contains sample services that provide FileAct functionality.
wm.swiftnet.client.sample.interAct	This folder contains sample services that provide InterAct functionality.
wm.swiftnet.client.sample.property	This folder contains sample services that perform property-based functions.
wm.swiftnet.client.sample.SnF	This folder contains sample services that provide Store and Forward functionality.

wm.swiftnet.client.sample.fileAct:swExchangeFile

This service constructs an Sw:ExchangeFileRequest for a real-time FileAct service and sends the request to SWIFTNet by invoking SwCall. The service waits for the response from SWIFTNet (Sw:ExchangeFileResponse).

Input Parameters

transferType	String (get/put) Indicates whether it is a get file request or a put file request.
cryptoMode	String (Automatic/Manual) Indicates the crypto mode for this request.
requestType	String Optional. Indicates the request type (SwInt:RequestType) specific to the SWIFTNet service.
requestRef	String Optional. Used to populate element SwInt:RequestRef.
physicalName	String Full path of the file to be transferred. When the request is a get file request, if there is no value specified for this parameter, then the path is constructed by appending the current date to the default location specified on the ReceptionFolder parameter in the *IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file.
logicalName	String Optional. A logical name of the transferred file.
transferDescription	String Optional. Description of the transfer.

serviceName	String Optional. SWIFTNet service to be used (for example, swift.generic.fa). If there is no value specified for this parameter, then the default value (as set on the service parameter in the *IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file) is used.
clientPKIpassword	String Optional. Password used to unlock the client PKI profile. If there is no value specified for this parameter, then the default value (as set on the client_pki_password parameter in the *IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file) is used.
SwTransferEP	String Optional. Transfer end point when a remote file handler is used. A remote file handler with this transfer point must be running on the client machine. If there is no value specified for this parameter, then the default value (as set on the SwTransferEP parameter in the *IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file) is used. For more information about using a remote file handler, see "Invoking the Remote File Handler" on page 47.
NRIndicator	String (TRUE/FALSE) Indicates whether Non Repudiation is requested.

Output Parameters

SwExchangeFileResponse	IData Response returned by SWIFTNet.

wm.swiftnet.client.sample.fileAct:swExchangeFileSnF

This service constructs an Sw:ExchangeFileRequest for SnF FileAct service and sends the request to SWIFTNet by invoking SwCall. The service waits for the response from SWIFTNet (Sw:ExchangeFileResponse).

Input Parameters

cryptoMode	String (Automatic/Manual) indicates the crypto mode for this request.
requestType	String Optional. Indicates the request type (SwInt:RequestType) specific to the SWIFTNet service.
requestRef	String Optional. Used to populate element SwInt:RequestRef.
physicalName	String Full path of the file to be transferred.
logicalName	String Optional. A logical name of the transferred file.
transferDescription	String Optional. Description of the transfer.
notifQueue	String Name of the SnF queue used to store delivery notification for this SnF request.

serviceName	String Optional. SWIFTNet service to be used (for example, swift.generic.fa). If there is no value specified for this parameter, then the default value (as set on the service parameter in the *IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file) is used.
clientPKIpassword	String Optional. Password used to unlock the client PKI profile. If there is no value specified for this parameter, then the default value (as set on the client_pki_password parameter in the *IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file) is used.
SwTransferEP	String Optional. Transfer end point when a remote file handler is used. A remote file handler with this transfer point must be running on the client machine. If there is no value specified for this parameter, then the default value (as set on the SwTransferEP parameter in the *webMethods6\IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file) is used. For more information about using a remote file handler, see "Invoking the Remote File Handler" on page 47.
NRIndicator	String (TRUE/FALSE) Indicates whether Non Repudiation is requested.

Output Parameters

SwExchangeFileResponse	IData Response returned by SWIFTNet.

wm.swiftnet.client.sample.fileAct:swGetFileStatus

This service constructs an Sw:GetFileStatusRequest to obtain the status of a file transfer and sends the request to SWIFTNet by invoking SwCall. The service waits for the response from SWIFTNet (Sw:GetFileStatusResponse).

Input Parameters

cryptoMode	String (Automatic/Manual) Indicates the crypto mode for this request.
transferRef	String TransferRef of the file transfer.
fullFileStatus	String (TRUE/FALSE) Indicates whether full file status is requested.

Output Parameters

SwGetFileStatusResponse	IData Response returned by SWIFTNet.

wm.swiftnet.client.sample.interAct:swExchangeRequest

This service constructs an SwInt:ExchangeRequest (or SwInt:SendRequest) for a real-time InterAct service and sends the request to SWIFTNet by invoking SwCall. The service waits for the response from SWIFTNet (SwInt:ExchangeResponse or SwIntSendResponse).

Input Parameters

requestMode	String (Synchronous/Asynchronous) Indicates whether Synchronous request mode is used. For a Synchronous request, SwInt:ExchangeRequest is constructed. For an Asynchronous request, SwInt:SendRequest is constructed. In case of an asynchronous request, the *wait* and *waitAny* parameters are used to indicate whether to wait for the response from SWIFTNet.
cryptoMode	String (Automatic/Manual) Indicates the crypto mode for this request.
cryptoType	String (none/sign/encrypt/signAndEncrypt) Indicates whether crypto algorithms need to be applied to the request.
requestType	String Optional. Indicates the request type (SwInt:RequestType) specific to the SWIFTNet service.
memberRef	String Elements that need to be signed in the crypto block. (For example, RequestPayload.)
payload	String Business payload (must comply with the rules specified by the specific swift solutions used).
SwInt:NRIndicator	String (TRUE/FALSE) Indicates whether Non Repudiation is requested.
SwSec:CryptoUserInfo	String Optional. Used to populate element SwSec:CryptoUserInfo block.
wait	String (TRUE/FALSE) Indicates whether to wait for response.
waitAny	String (TRUE/FALSE) Indicates the option to receive a particular response.
serviceName	String Optional. SWIFTNet service to be used (for example, swift.generic.ia). If there is no value specified for this parameter, then the default value (as set on the service parameter in the *IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file) is used.
clientPKIpassword	String Optional. Password used to unlock the client PKI profile. If there is no value specified for this parameter, then the default value (as set on the client_pki_password parameter in the *IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file) is used.

Output Parameters

SwIntExchangeResponse	IData Response returned by SWIFTNet.

wm.swiftnet.client.sample.interAct:swExchangeRequestSnf

This service constructs an SwInt:ExchangeRequest (or SwInt:SendRequest) for an SnF InterAct service and sends the request to SWIFTNet by invoking SwCall. The service waits for the response from SWIFTNet (SwInt:ExchangeResponse or SwIntSendResponse).

Input Parameters

requestMode	String (Synchronous/Asynchronous) Indicates whether Synchronous request mode is used or not. For Synchronous request, then SwInt:ExchangeRequest is constructed. For Asynchronous request, SwInt:SendRequest is constructed. In case of asynchronous request, the *wait* and *waitAny* parameters are used to indicate whether to wait for the response from SWIFTNet.
cryptoMode	String (Automatic/Manual) Indicates the crypto mode for this request.
cryptoType	String (none/sign/encrypt/signAndEncrypt) Indicates whether crypto algorithms need to be applied to the request.
requestType	String Optional. Indicates the request type (SwInt:RequestType) specific to the SWIFTNet service.
memberRef	String Elements that need to be signed in the crypto block. (for example, RequestPayload).
payload	String Business payload (must comply with the rules specified by the specific SWIFT solutions used).
SwInt:NRIndicator	String (TRUE/FALSE) Indicates whether Non Repudiation is requested.
SwSec:CryptoUserInfo	String Optional. Used to populate element SwSec:CryptoUserInfo block.
wait	String (TRUE/FALSE) Indicates whether to wait for response.
waitAny	String (TRUE/FALSE) Indicates the option to receive a particular response.
notifQueue	String Name of the SnF queue used to store delivery notification for this SnF request.
serviceName	String Optional. SWIFTNet service to be used (for example,swift.generic.ia). If there is no value specified for this parameter, then the default value (as set on the service parameter in the *IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file) is used.
clientPKIpassword	String Optional. Password used to unlock the client PKI profile. If there is no value specified for this parameter, then the default value (as set on the client_pki_password parameter in the *IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file) is used.

Output Parameters

SwIntExchangeResponse	IData Response returned by SWIFTNet.

wm.swiftnet.client.sample.property:getCommonProperties

This service retrieves the most commonly used properties from the *IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file.

Input Parameters

None.

Output Parameters

SAGMessagePartner	String Must correspond to a “Client” type message partner defined in the SAG server.
client_pki_profile	String User name of the profile defined in the SAG server.
client_pki_password	String Password associated with the user name used to unlock the security information in the SAG server.
Sign,Decrypt and Authorization	String Values are used for populating SwSec:CreateContextRequest primitive exchanged during server initialization. Valid values: `True` and `False`.
encryptDN	String Distinguished Name to be used for encryption operations.
requestor,responder	String Values used to populate SwInt:Requestor and SwInt:Responder elements in the SNL primitives.
service	String Specifies the name of the default SWIFTNet service to be used when interacting with SWIFTNet.

wm.swiftnet.client.sample.property:getProperty

This service retrieves the value of the specified property from the *IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file.

Input Parameters

propertyName	String Property value to be retrieved.

Output Parameters

value	String Value of the property.

wm.swiftnet.client.sample.property:listProperties

This service retrieves all the properties specified in the *IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file.

Input Parameters

None.

Output Parameters

properties	IData	All properties in the snl.cnf file.

wm.swiftnet.client.sample.property:reloadProperties

This service reloads all the properties specified in the *IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file. This service could be useful if more properties are added or existing properties have been changed and the changes need to be reflected in the Integration Server immediately.

Input Parameters

None.

Output Parameters

properties	IData	All properties reloaded from the client snl.cnf file.

wm.swiftnet.client.sample.SnF:swAcquireSnFQueue

This service constructs an Sw:ExchangeSnFRequest in order to acquire an SnF queue in push mode, and sends the request to SWIFTNet by invoking SwCall. The service waits for the response from SWIFTNet (Sw:ExchangeSnFResponse).

Input Parameters

cryptoMode	String	(Automatic/Manual) Indicates the crypto mode for this request.
queue	String	Name of the SnF queue to be acquired.

Output Parameters

SwExchangeSnFResponse	IData	Response returned by SWIFTNet.

wm.swiftnet.client.sample.SnF:swFetchFile

This service constructs an Sw:FetchFileRequest to fetch a file from an SnF queue, and then sends the request to SWIFTNet by invoking SwCall. The service waits for the response from SWIFTNet (Sw:FetchFileResponse).

Input Parameters

cryptoMode	String (Automatic/Manual) Indicates the crypto mode for this request.
Sw:FileRequestHandle	IData Represents the XML block containing Sw:FileRequestHandle, which contains the Sw:NotifyFileRequestHandle element (either pulled using Sw:PullSnFRequest, or pushed from SWIFTNet if a queue is acquired in push mode).
SwTransferEP	String Optional. Transfer end point when a remote file handler is used. A remote file handler with this transfer point must be running on the client machine. If there is no value specified for this parameter, then the default value (as set on the SwTransferEP parameter in the *IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file) is used. For more information about using a remote file handler, see "Invoking the Remote File Handler" on page 47. If no value is set on the SwTransferEP parameter in this service or in the client module snl.cnf file (*IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf), the fetched file is saved on the SAG server machine at the path specified on the *physicalName* parameter. In this case, you must ensure that the path specified on the *physicalName* parameter is present in the SAG server machine.
physicalName	String Optional. Full path where the fetched file should be stored. If there is no value specified for this parameter, then the path is constructed by appending the current date to the default location specified on the ReceptionFolder parameter in the *IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file.
SwEventEP	String Optional. File Transfer Event End point where file transfer events associated with file transfers carried out while fetching the files, should be sent. If there is no value specified for this parameter, then the default value (as set on the SwEventEP parameter in the *IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file) is used. The value specified for the SwEventEP parameter should be the same as is specified in the server module snl.cnf (*IntegrationServer*\packages\WmSWIFTNetServer\config\snl.cnf) file for the file transfer events to get posted to the server module Integration Server
queue	String Name of the SnF queue from which the file is fetched.

Output Parameters

SwFetchFileResponse	IData	Response returned by SWIFTNet.

wm.swiftnet.client.sample.SnF:swPullMessageOrFileSnF

This service processes messages in the SnF queue, based on message type. The service does several things:

1 It constructs an Sw:ExchangeSnFRequest to acquire an SnF queue in pull mode, and then sends the request to SWIFTNet by invoking SwCall.

2 On receiving a successful acquire queue response from SWIFTNet (Sw:ExchangeSnFResponse), the service sends Sw:PullSnFRequest to pull a message from the SnF queue. The pulled message is contained in Sw:PullSnFResponse. The service processes the messages based on the type:

- If the pulled message is an InterAct message, the service posts it to Trading Networks, and sends an acknowledgement back to SWIFTNet.
- If the pulled message is a FileAct message, the service constructs an Sw:FetchFileRequest and sends it to SWIFTNet to fetch the file from the SnF queue. After the file transfer is complete, the service posts the pulled message to Trading Networks, and sends an acknowledgement back to SWIFTNet. The SwEventEP parameter in the client module snl.cnf file (*IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf) must be the same as in the server module snl.cnf file (*IntegrationServer*\packages\WmSWIFTNetServer\config\snl.cnf) for the file transfer events to get posted to the server Integration Server.
- If the pulled message is a delivery notification, the service posts it to Trading Networks, and sends an acknowledgement back to SWIFTNet.

3 Step 2 is repeated until the SnF queue is empty.

Input Parameters

cryptoMode	String	(Automatic/Manual) Indicates the crypto mode for this request.
queue	String	Name of the SnF queue from where to pull messages.
clientPKIPassword	String	Optional. Password used to unlock the client PKI profile. If there is no value specified for this parameter, then the default value (as set on the client_pki_password parameter in the *IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file) is used.

SwTransferEP	String Optional. Transfer end point when a remote file handler is used. A remote file handler with this transfer point must be running on the client machine. Applicable if FileAct messages are anticipated. If there is no value specified for this parameter, then the default value (as set on the SwTransferEP parameter in the *IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file) is used. For more information about using a remote file handler, see "Invoking the Remote File Handler" on page 47. If no value is set on the SwTransferEP parameter in this service or in the client module snl.cnf file (*IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf), the pulled file is saved on the SAG server machine at the path specified on the *physicalName* parameter. In this case, you must ensure that the path specified on the *physicalName* parameter is present in the SAG server machine.
physicalName	String Optional. Full path where the pulled files should be stored. Applicable if FileAct messages are anticipated. If there is no value specified for this parameter, then the path is constructed by appending the current date to the default location specified on the ReceptionFolder parameter in the *IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file.

Output Parameters

SwPullSnFResponse	IData Response returned by SWIFTNet for the last Sw:PullSnFRequest.
SwFetchFileResponse	IData Response returned by SWIFTNet for the last Sw:FetchFileRequest.
SwAckSnFResponse	IData Response returned by SWIFTNet for the last Sw:AckSnFRequest.

wm.swiftnet.client.sample.SnF:swReleaseSnFQueue

This service constructs an Sw:ExchangeSnFRequest in order to release an SnF queue, and then sends the request to SWIFTNet by invoking SwCall. The service waits for the response from SWIFTNet (Sw:ExchangeSnFResponse).

Input Parameters

cryptoMode	String (Automatic/Manual) Indicates the crypto mode for this request.
sessionId	String SnF Session ID. Must match the id returned in the response at the time the queue is acquired.

Output Parameters

SwExchangeSnFResponse	IData Response returned by SWIFTNet.

WmSWIFTNetServerSample Package

This package contains the elements for a sample implementation of the webMethods SWIFTNet Module server-side functionality.

It provides sample services for handling incoming SNL primitive requests that need to be processed and returned to the client. These services call other services in the sequence required by SWIFTNet. You can use these services or use them as guides for creating your own services. This package also provides sample mapping services that map a given set of inputs into the XML primitive required by SWIFTNet.

This package contains the following folders:

Folder	Description
wm.swiftnet.server.sample.fileAct	This folder contains sample services that provide FileAct functionality.
wm.swiftnet.server.sample.interAct	This folder contains sample services that provide InterAct functionality.
wm.swiftnet.server.sample.SnF	This folder contains sample services that provide Store and Forward functionality.

wm.swiftnet.server.sample.fileAct:processFileEventRequest

This service constructs an XML response for a file event request (Sw:HandleFileEventRequest), received from SWIFTNet.

Input Parameters

bizdoc	**Object** Document containing the SWIFTNet request Sw:HandleFileEventRequest.

Output Parameters

xmlResponse	**String** Response constructed (Sw:HandleFileEventResponse).

wm.swiftnet.server.sample.fileAct:processFileRequest

This service constructs an XML response for a real-time handle file request (Sw:HandleFileRequest), received from SWIFTNet.

Input Parameters

bizdoc	**Object** Document containing the SWIFTNet request Sw:HandleFileRequest.

Output Parameters

xmlResponse	**String** Response constructed (Sw:HandleFileResponse). Uses parameters SwTransferEP and ReceptionFolder from the *IntegrationServer*\packages\WmSWIFTNetServer\config\snl.cnf file to populate the SwTransferEP and Sw:PhysicalFile (in case of put file request) elements, respectively.

wm.swiftnet.server.sample.interAct:processHandleRequest

This service constructs an XMLresponse for a real-time InterAct handle request (SwInt:HandleRequest), received from SWIFTNet.

Input Parameters

bizdoc	**Object** Document containing the SWIFTNet request SwInt:HandleRequest.

Output Parameters

xmlResponse	**String** Response constructed (SwInt:HandleResponse).

wm.swiftnet.server.sample.SnF:processHandleSnFRequest

This service constructs an XML response for an SnF delivery notification (Sw:HandleSnFRequest), received from SWIFTNet.

Input Parameters

bizdoc	**Object** Document containing the SWIFTNet request Sw:HandleSnFRequest.

Output Parameters

xmlResponse	**String** Response constructed (Sw:HandleSnFResponse). Contains the acknowledgement for the notification.

wm.swiftnet.server.sample.SnF:processSnFFileRequest

This service constructs an XML response for an SnF handle file request (Sw:HandleFileRequest), received from SWIFTNet. The server application sends an Sw:HandleFileResponse for the Sw:HandleFileRequest with an empty Sw:NotifyFileResponse. This empty Sw:NotifyFileResponse indicates that the file will be fetched by a client process. The service then remote-invokes the wm.swiftnet.client.sample.SnF:swFetchFile service on the SWIFTNet Client. A remote server alias pointing to the SWIFTNet Client module with the name 'SwiftNetClient' must be created in order to use this service. This file transfer is done by the remote file handler specified on the SwTransferEP parameter in the *IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file on the SWIFTNet Client installation. For more information about using a remote file handler, see "Invoking the Remote File Handler" on page 47.

The swFetchFile service fetches the file waiting in the SnF queue, and then sends the acknowledgement after the file transfer is complete. Keep in mind the following points for the swFetchFile service:

- The file is saved at the location specified on the ReceptionFolder parameter in the *IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf file.
- If no value is set for the SwTransferEP parameter in the client module snl.cnf file (*IntegrationServer*\packages\WmSWIFTNetClientSample\config\snl.cnf), the fetched file is saved on the SAG server machine at the location indicated in the ReceptionFolder parameter in the client module snl.cnf file. In this case, the user must ensure that the path given in the ReceptionFolder parameter is present in the SAG server machine.
- The SwEventEP value in the client module snl.cnf file must be the same as in the server module snl.cnf file (*IntegrationServer*\packages\WmSWIFTNetServerSample\config\snl.cnf) in order for the file transfer events to be posted to the server Integration Server.

Input Parameters

bizdoc	**Object** Document containing the SWIFTNet request Sw:HandleFileRequest.

Output Parameters

xmlResponse	**String** Response constructed (Sw:HandleFileResponse).

wm.swiftnet.server.sample.SnF:processSnFHandleRequest

This service constructs an XML response for an SnF InterAct handle request (SwInt:HandleRequest), received from SWIFTNet.

Input Parameters

bizdoc	**Object** Document containing the SWIFTNet request SwInt:HandleRequest.

Output Parameters

xmlResponse	**String** Response constructed (SwInt:HandleResponse). Payload contains the acknowledgment for the request.

Setting Up Aliases for Remote Integration Servers

You can set up aliases for remote servers. Communication through the alias is optimized, making transactions with the remote server faster. In addition, using an alias is more convenient because it saves you from specifying connection information each time you communicate with the remote server.

The alias grants access to a remote service by allowing the user to impersonate an authorized user on the remote server. Therefore, to prevent unauthorized users from accessing services on remote servers, the alias also contains access control information.

Use the following procedure to add an alias for a remote Integration Server.

1. Open the Integration Server Administrator if it is not already open.
2. In the **Settings** menu of the Navigation panel, click **Remote Servers**.
3. Click **Create Remote Server Alias**.
4. Set the **Remote Server Alias Properties** as follows:

   Alias = SwiftNetClient

   Host Name or IP Address = <*Host Name or IP Address of the client Integration Server*>

   Port Number = <*Port number of the client Integration Server*>

   User Name = <*Username of the client Integration Server*>

   Password = <*Password of the client Integration Server*>

   Execute ACL = Anonymous

For more information, see the *webMethods Integration Server Administrator's Guide*.