webMethods

webMethods SWIFT FIN Module

Installation and User's Guide

VERSION 6.0.1 SERVICE PACK 2

Document ID: ESTD-SWIFTFIN-IUG-601-20040423

Contents

About This Guide

This guide describes how to install, configure, and use the webMethods SWIFT FIN Module.

To use this guide effectively, you should:

■ Have a basic knowledge of SWIFT, SWIFT FIN, 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, the webMethods Developer, the webMethods Trading Networks (server side and console side) software, the webMethods Modeler (server side and client side) software, the webMethods Monitor (server side and client side) software, and the webMethods SWIFT FIN Module software. For more information about installing these components, see the *webMethods Integration Platform Installation Guide*.

■ Be familiar with the webMethods Integration Server, the Server Administrator, and 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 webMethods Trading Networks Console and understand the concepts and procedures described in the various *webMethods Trading Networks* guides.

■ Be familiar with webMethods Modeler and understand the concepts and procedures described in the *webMethods Modeler User's Guide*.

■ Be familiar with webMethods Monitor and understand the concepts and procedures described in the *webMethods Monitor User's Guide*.

■ Be familiar with webMethods Workflow and understand the concepts and procedures described in the *webMethods Workflow Concepts Guide* and the *webMethods Workflow User's Guide*.

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.
\	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.

Concepts

What Is the SWIFT Network?

SWIFT and its networks provide a secure, global financial IP-based messaging platform that enables financial institutions to exchange formatted financial information and transactional data. The SWIFT networks enable you to exchange SWIFT FIN messages using the original SWIFT Transport Network (STN) or the new SWIFT Secure IP Network (SIPN).

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

What Is a SWIFT FIN Message?

SWIFT FIN messages transmit financial information from one financial institution to another. These messages are classified into message categories. Each category contains a number of messages relating to a particular topic, such as Category 5, which contains messages related to Securities.

Each SWIFT message is represented by a three-digit number (for example, `MT 541`). The `MT` represents SWIFT's 'Message Type'. The first number (`5`) identifies the category to which the message belongs while the second and third numbers (`41`) identify the particular message.

All SWIFT FIN messages conform to a defined block structure. SWIFT FIN messages consist of a number of blocks, including one or more reference headers, the body text of the message, and one or more control trailers. Each block within a message contains data of a particular type, for a particular purpose, and begins and ends with a brace character '{' and '}'. All main blocks are numbered, and the block number, followed by a colon, is always the first character within any block (for example, `1:`).

The body of each message consists of message tags (such as, `22F::MICO`), each of which correspond to a business name (`Method of Interest Computation Indicator`). The values of a message tag follow the message tag itself (for example, `22F::MICO/A2C4E6G8/A2C4`).

Sample SWIFT FIN Message, *MT 541, Receive Against Payment*

```
{1:F01CLSAHKHHXXXX0116013185}{2:I541CLSAHKHHXXXXN}
{3:{108:MT535 004 OF 006}}
{4:
:16R:GENL
:20C::SEME//01430
:23G:NEWM/CODU
:98C::PREP//19991231232359
:99B::SETT//123
:16R:LINK
:22F::LINK/A2C4E6G8/A2C4
:13A::LINK//513
:20C::PREV//x
```

```
:16S:LINK
:16S:GENL
:16R:TRADDET
:94B::TRAD//EXCH/30x
...
:97A::CASH//x
:97A::SAFE//x
:16S:OTHRPRTY
-}
```

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

What Is the webMethods SWIFT FIN Module?

The webMethods SWIFT FIN Module enables the webMethods Integration Platform to do the following:

■ Receive inbound SWIFT FIN messages from other systems through one of SWIFT's networks.

■ Convert SWIFT FIN messages into your back-end format and process according to your settings.

■ Send SWIFT FIN messages to the SWIFT network with correct header information according to your settings.

The webMethods SWIFT FIN Module supports both SWIFT networks, through which you can send and receive messages using the original SWIFT Transport Network:

■ **SWIFT Transport Network (STN)**. This is SWIFT's original network, which is accessed using x.25 transport technologies, SWIFT Alliance Access (SAA), and a transport protocol (MQSA, CASmf, or AFT).

■ **SWIFT Secure IP Network (SIPN)**. This is SWIFT's new network, which may be accessed using IP and related technologies, the SWIFTAlliance Gateway (SAG), and an MQseries host adapter (MQHA). The webMethods SWIFT FIN Module accesses the SIPN by first accessing the STN, and then connecting to the SIPN. With this network, SWIFT has developed a *single window concept* to give users access to various SWIFTNet services using one interface, the SWIFTAlliance Gateway (SAG). The SAG focuses on automated application-to-application and facilitates the connectivity to SWIFT.

For both STN and SIPN, regardless of transport protocol (MQSA, CASmf, or AFT), the webMethods SWIFT FIN Module provides the ability to seamlessly integrate SWIFT FIN messages as webMethods documents into a solutions architecture and validate those messages at the syntax and network level. Messages sent and received by the webMethods SWIFT FIN Module are validated at the individual field level and across the fields using network validation rules. It also supports Market Practices among partners located in a particular market.

The following diagram presents the entire webMethods SWIFT solution as you might use it in conjunction with your back-end systems.

webMethods SWIFT Solution Architecture
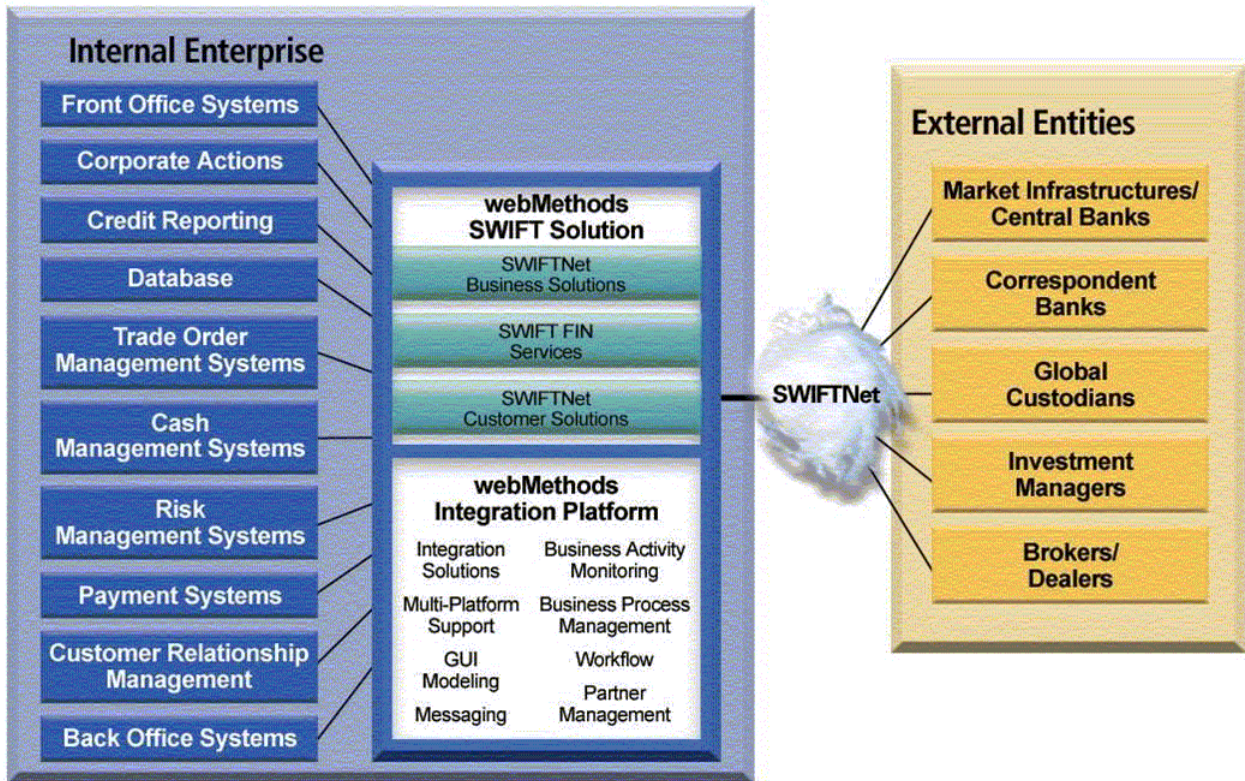
SWIFT FIN Message Support

webMethods parses and validates all SWIFT FIN messages in version nov03. But webMethods supports network validation rules only for the following nov03 SWIFT FIN messages:

- *MT 103 Single Customer Credit Transfer*

- *MT 202 General Financial Institution Transfer*

- *MT 300 Foreign Exchange Confirmation*

- *MT 502 Order to Buy or Sell*

- *MT 545 Receive Against Payment Confirmation*

- *MT 547 Deliver Against Payment Confirmation*

- *MT 548 Settlement Status and Processing Advice*

- *MT 564 Corporate Action Notification*

- *MT 515 Client Confirmation of Purchase or Sale*

- *MT 535 Statement of Holdings*

- *MT 536 Statement of Transactions*

- *MT 537 Statement of Pending Transactions*

- *MT 540 Receive Free*

- *MT 541 Receive Against Payment*

- *MT 542 Deliver Free*

- *MT 543 Deliver Against Payment*

- *MT 565 Corporate Action Instruction*

- *MT 566 Corporate Action Confirmation*

- *MT 567 Corporate Action Status and Processing Advice*

- *MT 568 Corporate Action Narrative*

- *MT 900 Confirmation of Debit*

- *MT 910 Confirmation of Credit*

- *MT 940 Customer Statement Message*

- *MT 950 Statement Message*

You also may create additional network validation rules for other SWIFT FIN messages and versions. For steps to create validation rules, see "Creating Validation Rules" on page 71 of Chapter 7, "Creating Message Records and Validation Rules" in this guide.

webMethods SWIFT FIN Module Components

The following components comprise and support the webMethods SWIFT FIN Module:

- **SWIFT FIN Packages**. The webMethods SWIFT FIN Module includes a number of packages containing services, mappings, records, and samples for using the webMethods SWIFT FIN Module with the webMethods Integration Platform. For a complete list of packages, see "webMethods SWIFT FIN Module Packages" on page 16 in this chapter.

- **Transport Protocols**. You can connect to SWIFT using one of the following transport protocols:

    - **webMethods MQSeries Adapter**. For more information about using the MQSeries Adapter with the webMethods SWIFT FIN Module, see "Using the MQSeries Adapter to Communicate with SWIFT" on page 52 of Chapter 5, "Configuring Transport Protocols" in this guide.

    - **webMethods CASmf Adapter**. For more information about using the webMethods CASmf Adapter, see "Using the CASmf Adapter to Communicate with SWIFT" on page 54 of Chapter 5, "Configuring Transport Protocols" in this guide.

    - **Automated File Transfer (AFT)**. For more information about using AFT with the webMethods SWIFT FIN Module, see "Using AFT to Communicate with SWIFT" on page 58 of Chapter 5, "Configuring Transport Protocols" in this guide.

■ **Integration Server**. This is the underlying server of the webMethods Integration Platform. 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*.

■ **webMethods Trading Networks**. webMethods Trading Networks enables your enterprise to link with other financial institutions and marketplaces to form a business-to-business trading network. For more information, see the *webMethods Trading Networks—Getting Started with Trading Networks* and the *webMethods Trading Networks—Building Your Trading Network* guides.

■ **webMethods Modeler**. You use webMethods Modeler to create visual models of business processes using process models. For more information, see the *webMethods Modeler User's Guide*.

■ **webMethods Monitor**. You use webMethods Monitor to manage and monitor business processes. The Monitor displays information about a business process by retrieving information from the Process Logging Database. For more information, see the *webMethods Monitor User's Guide*.

■ **webMethods Workflow**. You use webMethods Workflow as the natural extension of webMethods Modeler. Modeler enables you to create Workflow steps that define the tasks that require human intervention. When you generate your process models, Modeler generates Workflow components for you. For more information, see the *webMethods Workflow User's Guide*.

webMethods SWIFT FIN Module Packages

The webMethods SWIFT FIN Module contains several packages (sets of webMethods services and related files) that you install on the webMethods Integration Server. The following table describes the contents of each package. For detailed information about the contents of a package, see Appendix A, "webMethods SWIFT FIN Module Services" in this guide.

Package	Description
WmCASmf	Contains support services used to send and receive SWIFT FIN messages using CASmf.
WmFIN	Contains services used to implement and support the SWIFT FIN-compliant functionality of the webMethods SWIFT FIN Module.
WmFINDev	Contains the records and services that enable users to create message records, TPAs, TN document types, validation rules, and Processing Rules, as well as migrate SWIFT FIN messages between ISO 7775 and ISO 15022.

Package	Description
WmFINMarketPractice	Contains 16 common services that support Market Practices for some Category 5 SWIFT FIN messages.
WmFINSamples	Contains services that demonstrate how to send and receive SWIFT FIN messages using the webMethods SWIFT FIN Module.
WmFINTransport	Contains the services needed to send and receive SWIFT FIN messages using Automated File Transfer (AFT) and MQSeries.
WmIPCore	Contains generic services for using the webMethods SWIFT FIN Module with the webMethods Integration Server.

webMethods SWIFT FIN Module Features

The webMethods SWIFT FIN Module, which runs on top of the webMethods Integration Server, provides the following functionality:

■ **Current Messages**. The webMethods SWIFT FIN Module always supports the latest release of SWIFT FIN messages.

■ **Data Field Dictionary.** The SWIFT FIN Module provides a data field dictionary (DFD) based on the ISO 15022 standards for SWIFT FIN messages. This enables translation of a message tag number (for example, 22F::SFRE) into a meaningful business name (for example, Statement Frequency Indicator). In addition, the SWIFT FIN Module enables you to choose how you want to display each message in webMethods Developer:

  ■ Tag number only (for example, 22F::SFRE)

  ■ Equivalent message business name only (for example, Statement Frequency Indicator)

  ■ Both the tag number and the equivalent message business name (for example, "22F::SFRE_Statement Frequency Indicator)

  ■ XML data tag (for example, F22FSFRE)

■ **Archiving messages**. All SWIFT FIN messages can be archived in webMethods Trading Networks.

■ **Transport Protocols.** The webMethods SWIFT FIN Module provides out-of-box support for MQSA, CASmf, and Automated File Transfer (AFT) transport protocols. For this purpose, webMethods provides the webMethods MQSeries Adapter, the webMethods CASmf Adapter, and the File Polling Listener. For more information, see Chapter 5, "Configuring Transport Protocols" in this guide.

■ **BIC and BIC+ Validation and Searching**. A Bank Identification Code (BIC) is a unique code assigned to a financial institution or one of its specific departments. The BIC for the sender and receiver of a message appears in SWIFT FIN messages in the B1 (Basic Header). SWIFT provides a list of valid BICs for all existing SWIFT financial institutions. All SWIFT FIN messages are validated against the BIC or BIC+ list to make sure the sender and receiver BICs are valid. In some SWIFT FIN messages, the BIC appears in the B4 block. In these cases, the BIC code is validated against the BIC or BIC+. You also can search for bank information based on the BIC code and bank description. For more information, see Chapter 6, "Working with BIC and BIC+ Lists" in this guide.

■ **Syntax and Network Validation**. The SWIFT FIN Module enables you to validate the message structure, field formats, and network rules of inbound and outbound SWIFT FIN messages. webMethods provides network validation rules for a few commonly used message types. In addition to these rules, the webMethods SWIFT FIN Module enables you to create network, Market Practice, and usage validation rules for additional messages as well. For more information, see Chapter 7, "Creating Message Records and Validation Rules" and Chapter 14, "Working with Market Practices" in this guide.

■ **Market Practices**. Market Practices are specific requirements for individual markets. Using Trading Partner Agreements (TPAs), the webMethods SWIFT FIN Module supports the customization of SWIFT FIN messages based on specific trading partner pairs. For more information about SWIFT-related Market Practices and TPAs, see Chapter 9, "Customizing Trading Partner Agreements" and Chapter 14, "Working with Market Practices" in this guide.

■ **Processing Rule support**. You can use the webMethods Trading Networks Processing Rules that you can choose to create along with each SWIFT message record instead using the process run time and process models to manage the execution of SWIFT FIN messages. For information about creating Processing Rules, see the *webMethods Trading Networks--Building Your Trading Network* guide.

■ **Process Modeling**. This feature enables webMethods Modeler users to construct complex, end-to-end processes depicting various document exchanges using the Process Run Time. For more information about creating and modifying process models for SWIFT, see Chapter 11, "Creating or Modifying a Process Model" in this guide or see the *webMethods Modeler User's Guide*.

■ **SWIFT Error Codes**. The webMethods SWIFT FIN Module supports SWIFT error codes for field level and network validation. The webMethods SWIFT FIN Module also supplies resource bundles so that all error codes can be localized.

■ **Repair and Resubmit Messages to SWIFT**. When an outbound message produces an error, the webMethods SWIFT FIN Module enables you to repair message and then resubmit it for delivery to SWIFT. For more information, see Chapter 13, "Repairing and Resubmitting Messages" in this guide.

- **ISO 7775 to ISO 15022 Migration**. The webMethods SWIFT FIN Module supports the migration of four SWIFT FIN messages from ISO 7775 to ISO 15022 and from ISO15022 to ISO 7775. You also can create additional migration services. For more information about migrating messages, see Chapter 15, "Migrating Messages" in this guide.

- **Integration Server and SWIFTNet Clustering**. The webMethods SWIFT FIN Module supports multiple instances of the webMethods SWIFT FIN Module talking to one instance of SWIFTNet, as well as multiple instances of SWIFTNet talking to multiple instances of the webMethods SWIFT FIN Module.

- **Subfield Parsing**. The webMethods SWIFT FIN Module automatically parses messages into blocks and fields. You can configure the SWIFT FIN Module to further parse the fields into subfields using the *subfieldFlag* variable, which is included in the following services:

  - wm.fin.dev:importFINItem This service creates all items needed in a SWIFT FIN Message transaction, including the TPA with the **SubfieldFlag** parameter set to true.

  - wm.fin.dfd:convertTagFormat This service converts FIN IData from TAGONLY to a specified format.

  - wm.fin.dfd:convertBizNameFormat This service converts FIN IData from a specified format to TAGONLY.

  For inbound SWIFT FIN messages, setting the *subfieldFlag* variable to true causes the webMethods SWIFT FIN Module to automatically parse messages to the subfield level and then remove the SWIFT delimiter (/) from between subfields. For outbound SWIFT FIN messages, setting the *subfieldFlag* variable to true causes the webMethods SWIFT FIN Module to add the SWIFT delimiter (/) between subfields.

webMethods SWIFT FIN Module Architecture

The webMethods SWIFT FIN Module uses either the publish and subscribe Process Run Time (PRT) functionality of the webMethods Integration Server, or webMethods Trading Networks Processing Rules to send and receive SWIFT FIN messages. It leverages the archiving, TPA, and document type components of the webMethods Trading Networks to work with your enterprise to exchange SWIFT FIN messages.

webMethods SWIFT FIN Module Architecture
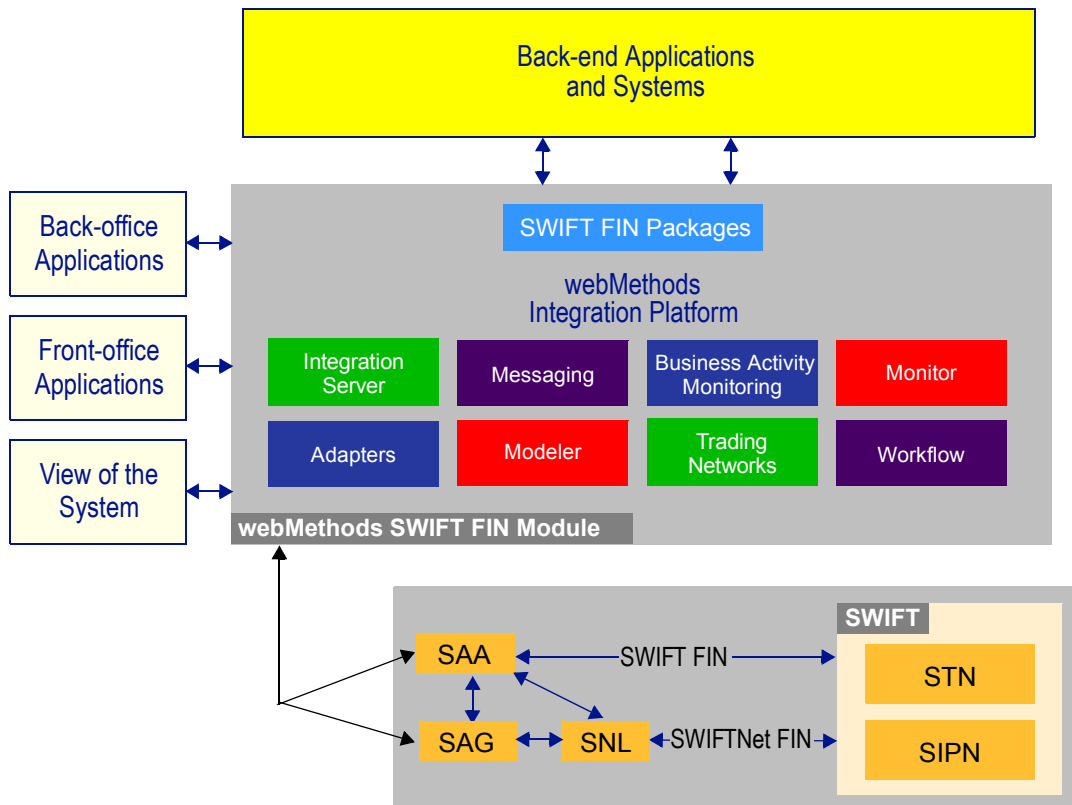
The webMethods SWIFT FIN Module consists of a set of design-time and run-time components, both of which are discussed in this chapter. For information about design-time components, see "Design-Time Architecture/Components" on page 21. For information about run-time components, see "Run-Time Architecture/Components" on page 23.

When the webMethods SWIFT FIN Module creates an outbound document, it formats, validates, and publishes the SWIFT message. When the webMethods SWIFT FIN Module receives an inbound document, it parses, formats, validates, and publishes the message for a back-end application. To communicate with SWIFT using the SAA (SWIFT Alliance Access), you use the webMethods MQSeries Adapter to interface with MQSA, the

webMethods CASmf Adapter to interface with CASmf, or the File Polling Listener and File Drop capabilities to interface with AFT. The following diagram illustrates the end-to-end architecture of the webMethods SWIFT FIN Module FIN messaging solution.

webMethods SWIFT FIN Messaging Architecture
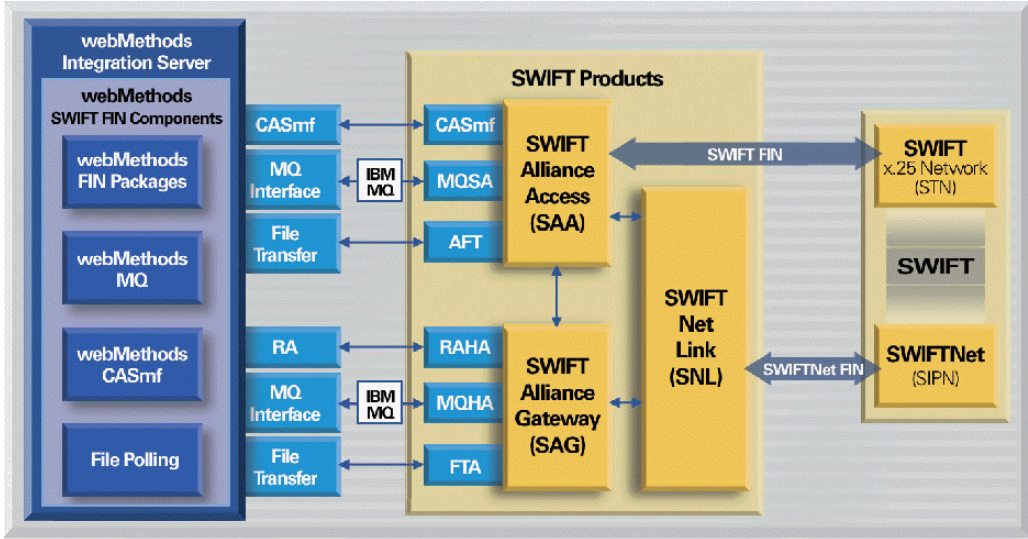
Design-Time Architecture/Components

The following figure illustrates the design-time architecture, components, and component relationships of the webMethods SWIFT FIN Module, and the component relationships. For further explanation, see the table that follows the next figure.

Design-Time Architecture/Components

Design-Time Components

Component	Description
webMethods SWIFT FIN Module	The webMethods SWIFT FIN Module enables you to manage the execution of SWIFT FIN messages using either process models or Trading Networks Processing Rules. It contains sample process models that you can use to define new process models using Modeler, and enables you to install Processing Rules.
webMethods Trading Networks	Trading Networks enables your enterprise to link to financial institutions with whom you want to exchange SWIFT FIN messages. During design time, you define your trading partner profiles in the Trading Networks Console. The profiles contain the information that your system needs to exchange messages with other financial institutions. In addition to defining trading partner profiles in the Trading Networks Console during design time, you also can customize the TPAs and view the TN document types that are created when you create your message records. For more information about Trading Networks, trading partner profiles, TN document types, and TPAs, see the *webMethods Trading Networks--Building Your Trading Network* guide. You also can find information about trading partner profiles and TN XML document types in Chapter 8, "Defining Trading Partner Profiles and TN Document Types", and information about TPAs in Chapter 9, "Customizing Trading Partner Agreements" in this guide.
TN Database	Trading Networks saves trading partner profiles, TN document types, and TPA information, among other things, to its database and retrieves this information when needed.
webMethods Modeler	Modeler is a Java GUI. You use Modeler to create process models, which you can base on the samples provided by webMethods. You create a process model by specifying how the process model is to interact with your back-end systems, specifying the services that are invoked by the steps of the process model, and using TN document types as inputs, among other things. When you generate a process model, the system generates the run-time elements (services, triggers, and process fragments). For more information about Modeler, see the *webMethods Modeler User's Guide*. For information about customizing a process model, see Chapter 11, "Creating or Modifying a Process Model" in this guide.

Component	Description
	Note: You can use webMethods Trading Networks Processing Rules installed with each SWIFT message record instead of the process run time and process models to manage the execution of SWIFT FIN messages. For information about creating Processing Rules, see Chapter 4, "Sending and Receiving SWIFT Messages" in this guide.
Modeler Database	The Modeler database is a storage area that Modeler uses to save process model and run-time information. For more information about the Modeler database, see the *webMethods Modeler User's Guide*.
webMethods Integration Server	The Integration Server contains the documents, services, and IS documents that you will want to access when creating your process models.

Run-Time Architecture/Components

The following figure illustrates the run-time architecture, components, and component relationships of the webMethods SWIFT FIN Module. For further explanation, see the table that follows the next figure.

Run-Time Architecture/Components

Run-Time Components

Component	Description
webMethods SWIFT FIN Module	During run time, the webMethods SWIFT FIN Module receives a message from a back-end system. It invokes a Trading Networks service to recognize the message. The webMethods SWIFT FIN Module may pass the message to the process run time (PRT). For more information about the PRT, see the Process Run Time row in this table and the *webMethods Modeler User's Guide*.
	Note: You can use webMethods Trading Networks Processing Rules installed with each SWIFT message record instead of the Process Run Time and process models to manage the execution of SWIFT FIN messages. For information about creating Processing Rules, see Chapter 4, "Sending and Receiving SWIFT Messages" in this guide.
webMethods Trading Networks	Trading Networks enables your institution to link to the financial institutions with whom you want to exchange messages.
	During run time, the webMethods SWIFT FIN Module uses Trading Networks services and TN document types to recognize messages that it receives, create BizDocEnvelopes, and save BizDocEnvelopes to the Trading Networks database. The webMethods SWIFT FIN Module uses the trading partner profiles in Trading Networks to identify, for example, the method by which to send messages to other financial institutions.
	You can find information about trading partner profiles and TN document types in Chapter 8, "Defining Trading Partner Profiles and TN Document Types", and information about TPAs in Chapter 9, "Customizing Trading Partner Agreements" in this guide. For more information about Trading Networks, trading partner profiles, TN document types, and TPAs, see the *webMethods Trading Networks--Building Your Trading Network* guide.
TN Database	The Trading Networks database stores TN document types, TPA, and trading partner profile information, among other things.
webMethods Monitor	You use Monitor to manage and monitor processes. Monitor displays information about a process by retrieving information from the Process Logging Database. You manage a process by performing such commands as suspend, resume, restart, and terminate.

Component	Description
Process Run Time	The process run time (PRT) is a facility of the Integration Server that manages the execution of SWIFTNet processes. The PRT ensures the integrity, traceability, observability, and controllability of SWIFTNet processes by performing the following functions: ■ Accepting messages from Trading Networks. ■ Determining which process model to use for a given message. ■ Processing a message based on the type of message received and who sent it. ■ Recording the status of the message to the Process Logging Database. **Note:** You can use webMethods Trading Networks Processing Rules that you can install with each SWIFT message record instead of the Process Run Time and process models to manage the execution of SWIFT FIN messages. For information about creating Processing Rules, see Chapter 4, "Sending and Receiving SWIFT Messages" in this guide.
Process Logging Database	webMethods Monitor and the PRT log and retrieve audit data about running processes to and from the Process Logging Database.
webMethods Integration Server	The Integration Server contains the run-time elements (services, triggers, and process fragments) that were generated by the automated controlled steps within the process model.

Installing the webMethods SWIFT FIN 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.

System Requirements

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

Platform and Operating System Requirements

The following table lists the supported platforms for the webMethods SWIFT FIN Module and the Java versions to use for each platform.

Platform and Operating System	Supported JREs
Hewlett-Packard HP-UX 11i (64-bit) and 11.0	JRE 1.3
Microsoft Windows NT 4.0 SP4	JRE 1.2.2 and 1.3
Microsoft Windows 2000 and XP Professional	JRE 1.2.2 and 1.3
Sun Solaris 2.8, 2.7 (all 64-bit)	JRE 1.2.2 and 1.3

webMethods Software Requirements

The following table lists the webMethods components you must install before you can install the webMethods SWIFT FIN Module:

Component	Version
webMethods Integration Server	6.0.1 and 6.1
webMethods Developer	6.0.1 and 6.1
webMethods Trading Networks	6.0.1 and 6.1

Because the webMethods SWIFT FIN Module requires a method of communicating with SWIFT, it is recommended that in addition to the above components, you also install one of the following webMethods components:

Component	Version
webMethods CASmf Package	6.0.1
webMethods WebSphere MQ Adapter	3.0

For more information about the webMethods CASmf Adapter, see Chapter 5, "Configuring Transport Protocols" in this guide.

The following table lists the webMethods components you should install if you plan to use process models to manage the execution of SWIFT FIN messages.

Component	Version
webMethods Modeler	6.0.1 and 6.1
webMethods Monitor	6.0.1 and 6.1

Third-Party Software Requirements

If you are using CASmf as your transport protocol, you must install a CASmf client (provided by SWIFT) on your Integration Server. For more information, see your SWIFT documentation or go to http://www.swift.com.

Hardware Requirements

The webMethods SWIFT FIN Module does not have any requirements in excess of those for webMethods Integration Server.

Installing webMethods SWIFT FIN Module

Important! This section provides only instructions that are specific to installing the webMethods SWIFT FIN Module. For complete instructions on using the webMethods Installer, see the *webMethods Integration Platform Installation Guide*.

Install webMethods SWIFT FIN Module 6.0.1 and the needed SWIFT message records on the same machine as Integration Server. The installer will automatically install the webMethods SWIFT FIN Module in the Integration Server installation directory.

> **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.

To install the webMethods SWIFT FIN Module

1 Download webMethods Installer 6.0.1 or 6.1 from the webMethods Advantage Web site at http://advantage.webmethods.com and start the installer.

2 Run the installer using the username and password you received from webMethods. Specify the installation directory as the webMethods installation directory (by default, webMethods6).

3 In the component selection list, navigate to **webMethods Platform ▶ eStandards ▶ webMethods SWIFT FIN Module** and select the desired components:

- **Documentation 6.0** (Optional). Contains the documentation for this package.

- **Messages** (Optional). Select the nov02 or nov03 DFDs for the SWIFT FIN messages that you will be using.

> **Important!** To be able to run the webMethods SWIFT FIN Module sample provided, you **must** install the nov03 Category 5 messages. For more information about the sample, see Appendix C, "webMethods SWIFT FIN Module Sample" in this guide.

- **Program Files** (Required). Contains the program files for this package.

- **Samples 6.0** (Optional). Contains the services that demonstrate how to send and receive SWIFT FIN messages over SWIFTNet.

- Any required webMethods components you have not installed.

Component Selection List
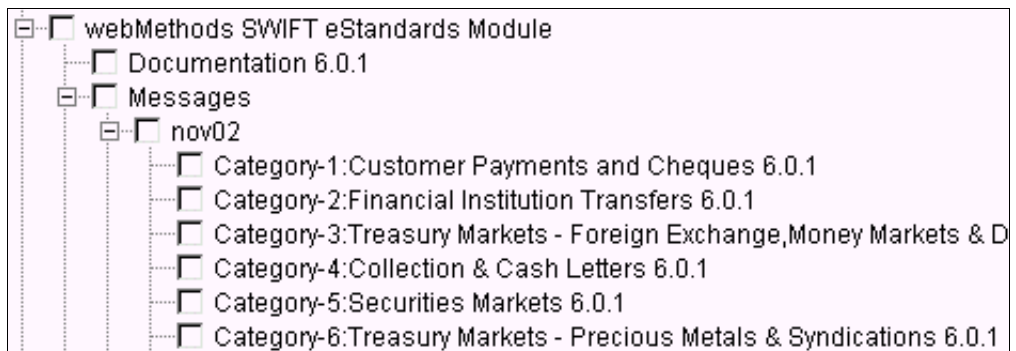
4 Click **Next** until the installer displays the **Installation Complete** panel.

5 Click **Close**.

6   If you are using the webMethods Broker with the webMethods Integration Server:

a   Navigate to and open the following file:

webMethods6\IntegrationServer\packages\WmFIN\transport\config\
fintransport.cnf

where *webMethods6\IntegrationServer* is the location of the webMethods
Integration Server.

b   Change the *fin.message.publishLocal* parameter to `false` to enable publication of
messages to the Broker.

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

Upgrading webMethods SWIFT FIN Module

This section describes how to upgrade from previous versions of the webMethods SWIFT
FIN Module to webMethods SWIFT FIN Module 6.0.1 and 6.0.1 Service Pack 2.

Upgrading from webMethods SWIFT FIN Module 6.0.1

When you want to upgrade from the webMethods SWIFT FIN Module 6.0.1 or 6.0.1
Service Pack 1 to webMethods SWIFT FIN Module 6.0.1 Service Pack 2, you need to
complete the following steps:

1   Download webMethods Installer 6.0.1 or 6.1 from the webMethods Advantage Web
site at http://advantage.webmethods.com and start the installer.

2   Run the installer using the username and password you received from webMethods.
Specify the installation directory as the webMethods installation directory (by default,
webMethods6).

3   In the component selection list, navigate to **webMethods Platform ▸ eStandards ▸
webMethods SWIFT FIN Module** and select the following component:

- **Service Pack 2.** Contains the service pack for subfield support as well as the
  contents of Service Pack 1.

4   Click **Next** until the installer displays the **Installation Complete** panel.

5   Click **Close**.

Upgrading from webMethods SWIFT FIN Module 4.6

When you want to upgrade from the webMethods SWIFT FIN Module 4.6 to webMethods SWIFT FIN Module 6.0.1, you need to complete the following steps:

1 Uninstall your webMethods Integration Server 4.6 as described in "Uninstalling webMethods SWIFT FIN Module" on page 32.

2 Install webMethods Integration Server 6.0.1. For more information about upgrading from webMethods Integration Server 4.6 to webMethods Integration Server 6.0.1, see the *webMethods Integration Platform Upgrade Guide*.

3 Install the webMethods SWIFT FIN Module 6.0.1 as described in "Installing webMethods SWIFT FIN Module" on page 29.

Uninstalling webMethods SWIFT FIN Module

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

You perform the following procedure when you want to completely uninstall the webMethods SWIFT FIN Module.

Important! If you want to keep certain records or services that you use with the existing webMethods SWIFT FIN Module packages on your webMethods Integration Server, make sure that you export them to a new package (from webMethods Developer, select the package to export, and select **File ▸ Export**) before performing the following procedure, which will remove all components in the packages.

To uninstall the webMethods SWIFT FIN Module from webMethods Integration Server

1 The instructions in the *webMethods Integration Platform Installation Guide* say to shut down all webMethods components and applications that are running on your machine. For webMethods SWIFT FIN Module, you need to shut down only the Integration Server.

2 Select **webMethods SWIFT FIN Module** as the program to uninstall.

3 The uninstaller removes all webMethods SWIFT FIN Module-related files that were installed into the *Integration Server_directory*\packages directory. The uninstaller does not delete files created after you installed the webMethods SWIFT FIN Module (for example, user-created or configuration files), nor does it delete the directory structure that contains the files.

4 If you want to delete the files that the uninstaller did not delete, navigate to the *Integration Server_directory*\packages directory and delete the WmFIN-related folders.

You perform the following procedure when you want to uninstall portions of the webMethods SWIFT FIN Module.

To uninstall portions of the webMethods SWIFT FIN Module from webMethods Integration Server

1   Start the webMethods Integration Server and the Server Administrator.

2   Under the **Package** menu in the Server Administrator navigation area, click **Management**.

3   From the **Package** list, locate the following packages:

   ◾ WmFIN

   ◾ WmFINDev

   ◾ WmFINMarketPractice

   ◾ WmFINMessages

   ◾ WmFINSamples

   ◾ WmFINTransport

   ◾ WmIPCore

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.

To uninstall the webMethods SWIFT FIN Module Service Pack

1   The instructions in the *webMethods Integration Platform Installation Guide* say to shut down all webMethods components and applications that are running on your machine. For webMethods SWIFT FIN Module, you need to shut down only the Integration Server.

2   Select the **Service Pack** in the **webMethods SWIFT FIN Module** as the program to uninstall.

3   The uninstaller removes all webMethods SWIFT FIN Module-related files that were installed into the *Integration Server_directory*\packages directory. The uninstaller does not delete files created after you installed the webMethods SWIFT FIN Module (for example, user-created or configuration files), nor does it delete the directory structure that contains the files.

Getting Started

How Do I Use the webMethods SWIFT FIN Module?

This chapter describes how to configure your webMethods Integration Server to prepare to send and receive SWIFT FIN messages using the services in the webMethods SWIFT FIN Module. The subsequent chapters in this guide provide more detailed information about each of these steps.

After you have completed the steps in this chapter, see Chapter 4, "Sending and Receiving SWIFT Messages" in this guide for detailed information about sending and receiving SWIFT FIN messages.

For a sample of how to use the webMethods SWIFT FIN Module to send and receive SWIFT FIN messages, see Appendix C, "webMethods SWIFT FIN Module Sample" in this guide.

Important! The following steps assume that you already have installed the webMethods Integration Server, webMethods Trading Networks, webMethods Modeler, webMethods Monitor, the necessary SWIFT software, the webMethods SWIFT FIN Module packages, and the appropriate transport protocol software. For more information about what SWIFT software you need, work with SWIFT to determine your software needs. For more information about installing the webMethods SWIFT FIN Module, see Chapter 2, "Installing the webMethods SWIFT FIN Module" in this guide.

Step 1: Configure Transport Protocol

You can connect to SWIFT using one of the following transport protocols:

- **IBM MQSeries**. If you are connecting to SWIFT through MQSA, you must install the webMethods MQ Adapter.

- **CASmf**. If you are connecting to SWIFT through CASmf, you must install the WmCASmf Package, which is included in the webMethods SWIFT FIN Module.

- **AFT**. If you are using Automated File Transfer (AFT), you must configure the File Polling Listener in the Integration Server Administrator (and the AFT settings in the message TPA.

For more information about configuring and using these protocols with the webMethods SWIFT FIN Module, see Chapter 5, "Configuring Transport Protocols" in this guide.

Step 2: Import a SWIFT BIC or BIC+ List

To uniquely identify each financial institution or one of its specific departments, SWIFT assigns a Bank Identification Code (BIC) to a each entity. SWIFT publishes a list of valid BICs for all existing SWIFT financial institutions in the SWIFT BIC and BIC+ directories. All SWIFT FIN messages are validated against the BIC or BIC+ list to make sure the sender and receiver are valid.

To import a list, you first must create a database table to hold the BIC or BIC+ list. For steps to create a BIC or BIC+ database table and import a BIC or BIC+ list into webMethods Integration Server, see Chapter 6, "Working with BIC and BIC+ Lists" in this guide.

Step 3: Create Message Records

For each SWIFT message that you want to exchange with your partner financial institutions, you must do the following to create a message record:

■ Install the SWIFT message DFD for the message (which includes a parsing template) into the webMethods SWIFT FIN Module.

■ Run the wm.fin.dev:importFINItems service for the message DFD.

For more information about creating message records, see Chapter 7, "Creating Message Records and Validation Rules" in this guide.

Step 4: Define Trading Partner Profiles

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

For more information about defining trading partner profiles for use with the webMethods SWIFT FIN Module, see Chapter 8, "Defining Trading Partner Profiles and TN Document Types" in this guide.

Step 5: Modify Trading Partner Agreements

When running the wm.fin.dev:importFINItems service for a message DFD to create a record, you can select to create a corresponding trading partner agreement (TPA), which is a set of parameters that governs how you exchange this SWIFT message with a trading partner. Before you can use a TPA, you must modify its parameters to process the message according to your company's needs, and then set the TPA's status to "Agreed".

Important! The **SubfieldFlag** input parameter in the TPA supports the parsing of SWIFT messages to the subfield level. If you change the value of the **SubfieldFlag** after creating the message records, you must delete the message records pertaining to the TPA and recreate them.

For information about modifying TPAs for use with SWIFT FIN messages, see Chapter 9, "Customizing Trading Partner Agreements" in this guide.

Step 6: Write Inbound and Outbound Mapping Services

To send and receive messages, you create inbound and outbound mapping services that define how each message will be handled. You create an *outbound* mapping service to map an internal message received from the back-end to a SWIFT message that you want to send to another financial institution. You create an *inbound* mapping service to map a SWIFT message received from a another financial institution to an internal message that is then sent to a back-end system. These services are used in the management of SWIFT message execution, which you define in the next step.

For more information about mapping business documents, see Chapter 10, "Mapping a SWIFT FIN Module Process" in this guide.

Step 7: Manage SWIFT Message Execution

To manage the execution of SWIFT FIN messages, you can use either the Process Run Time (PRT) and an process model, or the webMethods Trading Networks Processing Rule that you can choose to create when creating the SWIFT message record.

Executing Messages Using the PRT and Process Models

Based on the SWIFT FIN messages you are using, you can use or modify one of the sample process models provided by webMethods in the *webMethods6\IntegrationServer*\packages\WmFINSamples\data directory, where *webMethods6\IntegrationServer* is the location of the webMethods Integration Server, or you can create your own process models from scratch.

To begin using the webMethods SWIFT FIN Module to send and receive SWIFT FIN messages using the Process Run Time to manage the execution of SWIFT FIN messages, you must run the process models that you create.

For information about sending and receiving SWIFT FIN messages using the PRT and process models, see Chapter 4, "Sending and Receiving SWIFT Messages" in this guide. For information about working with process models and SWIFT FIN messages, see Chapter 11, "Creating or Modifying a Process Model" in this guide. For information about starting and monitoring a SWIFT business process, see Chapter 12, "Monitoring a Process" in this guide. For general information about webMethods Modeler, see the *webMethods Modeler User's Guide*.

Executing Messages Using TN Processing Rules

A webMethods Trading Networks Processing Rule enables you to manage the execution of SWIFT FIN messages without using the PRT. If you choose to create a Processing Rule for a SWIFT FIN messages, the rule is created in the WmFINMessages package in the appropriate version folder (for example, *webMethods6\IntegrationServer*\packages\ WmFINMessages\nov03).

For information about sending and receiving SWIFT FIN messages using Processing Rules, see Chapter 4, "Sending and Receiving SWIFT Messages" in this guide. For general information about using Processing Rules, see the *webMethods Trading Networks--Building Your Trading Network* guide.

Sending and Receiving SWIFT Messages

Overview

The webMethods SWIFT FIN Module enables you to send and receive a SWIFT message using either a business process or a Trading Networks Processing Rule.

Sending Outbound Messages to SWIFT

You can send outbound messages to SWIFT using either a business process or a service that is invoked by a Trading Networks Processing Rule.

Sending Outbound Messages Using a PRT Business Process

To use a process run time (PRT) business process (also called a conversation) to send outbound SWIFT FIN messages, you use webMethods Modeler to create a process model that defines the steps that you want to take in your business process. For a guideline, see the sample process model (*CorpActions_Account Servicer*) provided in the SWIFT FIN Module sample. For more information, see Appendix C, "webMethods SWIFT FIN Module Sample" in this guide.

After designing the process model, use webMethods Modeler to generate it. When you generate a process model, webMethods Modeler generates the run-time elements (for example, triggers, flow services, etc.) that are executed for the business process.

The first step of a business process is a step that waits for a document. At run time, you send a back-end document to the PRT, which is the facility of the Integration Server that manages business processes. The PRT determines which process model to use for the document and either instantiates a new instance of the process model to start the business process, or when subsequent documents aimed at the business process arrive, the PRT delivers them to the appropriate business process to rejoin a running process.

The PRT uses the Conversation ID to determine whether documents belong to the same instance of a business process. All documents that belong to the same instance of a business process use the same Conversation ID.

Before You Can Send Outbound Messages

■ Define a TN document type for the back-end format document that you will convert to a SWIFT FIN message. For instructions to create TN document types, see the *webMethods Trading Networks--Building Your Trading Network* guide.

When you create this TN document type, be sure to extract the **SenderID** and **ReceiverID** system attributes. The values you extract for the **SenderID** and **ReceiverID** should be the sender's and receiver's BIC, respectively.

◼ Create two TPAs as described in the following tables:

TPA One -- Used by first step of the business process

Sender	The sender of the back-end document -OR- **Unknown**.
	If you specify a specific partner for **Sender**, you must specify a specific partner for **Receiver**. Likewise, if you specify **Unknown** for **Sender**, you must specify **Unknown** for **Receiver**.
Receiver	The receiver of the SWIFT FIN message that is to be sent to SWIFT.
AgreementID	Name of the TN document type for the back-end system format document.
IS Document Type	`wm.fin.doc:UserParameters`

TPA Two -- Used to govern the creation and sending of the outbound SWIFT FIN message

Sender	The sender of the back-end document -OR- Unknown.
	If you specify a specific partner for **Sender**, you must specify a specific partner for **Receiver**. Likewise, if you specify **Unknown** for **Sender**, you must specify **Unknown** for **Receiver**.
Receiver	The receiver of the SWIFT message that is to be sent to SWIFT.
AgreementID	User-defined. You might want to specify a name that identifies the type of SWIFT FIN message that the TPA governs, for example, MT564.
IS Document Type	`wm.fin.doc:UserParameters`

Designing the Process Model

Your process model will typically contain the following steps:

1 Wait for the document from the back-end system -and- invoke the wm.ip.cm:waitStepInit service. This service takes the actions necessary to prepare for the business process. During processing, it retrieves the TPA associated with the AgreementID set to the name of the TN document type. Processing will fail if the first step does *not* invoke this service -or- if the service fails to retrieve the TPA.

2 Invoke a mapping service to map to the SWIFT FIN message.

   a Map data from the back-end system document to webMethods DFD format, which represents the {B4} block of the SWIFT FIN message.

   b Add the *payload* variable to the existing *documents* variable in the pipeline, and then set the *documents/payload* variable to the IData object that represents the

webMethods DFD format document. For an example of how to do this, see the outbound map service provided in the WmFINSamples package.

c Add the *tpaAgreementID* variable to the existing *documents* variable in the pipeline, and then set the *documents/tpaAgreementID* variable to the AgreementID associated with the second TPA, which governs the creation and sending of the outbound SWIFT FIN message.

3 Invoke the wm.fin.trp:send service to send the SWIFT FIN message to SWIFT. This service:

a Creates the header blocks {B1-B3} and {B5} using information from the second TPA that you created.

b Determines whether to validate the SWIFT message using the **Validate...** parameters in the second TPA. If any of the **Validate...** parameters for example, **ValidateContent**) are set to **yes**, the service performs the appropriate validation.

c Sends the SWIFT FIN message using the transport specified in the message TPA (for example, MQ, CASmf, AFT).

Executing the Business Process

To run the business process, send the back-end system document that starts the business process to the Integration Server by invoking the wm.ip.cm:processDocument service. The wm.ip.cm:processDocument service performs the following actions:

■ Recognizes the back-end system document using TN document types.

■ Forms a BizDocEnvelope for the back-end system document.

■ Sets the Conversation ID for the back-end system document.

   ■ If this is the first document of a business process, the processDocument service sets the Conversation ID to the <Sender's BIC>-<Unique IDentifier>. This unique identifier is mapped into the MUR section in {B3} block of the outgoing SWIFT FIN message.

   ■ If this document is to join a running business process, you must supply the Conversation ID and pass it as input to the processDocument service.

■ Saves the content of the back-end system document to the Trading Networks database.

■ Passes the back-end system document to the PRT. When the PRT receives the back-end system document, it determines whether this is the first document for a business process, or whether it is a document that is rejoining a running business process. If it is the first document of a business process, the PRT instantiates a new instance of the process model. Otherwise, it sends the document to rejoin a running business process.

Note: When you use the wm.ip.cm:processDocument service, the document does *not* go through typical Trading Networks processing. This service invokes the Trading Networks services it needs to accomplish the tasks described above. The document does *not* go to Trading Networks Processing Rules.

Sending Outbound Messages Using a Processing Rule

To use a Processing Rule to send outbound SWIFT FIN messages, you define a Processing Rule in Trading Networks that invokes a service to send the message.

Before You Can Send Outbound Messages

■ Define a TN document type for the back-end format document that you will convert to a SWIFT message. For instructions to create TN document types, see the *webMethods Trading Networks--Building Your Trading Network* guide.

When you create this TN document type, be sure to extract the **SenderID** and **ReceiverID** system attributes. The values you extract for the **SenderID** and **ReceiverID** should be the sender's and receiver's BICs, respectively.

■ Create a TPA as described in the following table:

TPA used to govern the creation and sending of the outbound SWIFT message

Sender	The sender of the back-end document -OR- **Unknown**.
	If you specify a specific partner for **Sender**, you must specify a specific partner for **Receiver**. Likewise, if you specify **Unknown** for **Sender**, you must specify **Unknown** for **Receiver**.
Receiver	The receiver of the SWIFT message that is to be sent to SWIFT.
AgreementID	Name of the TN document type for the back-end system format document.
IS Document Type	`wm.fin.doc:UserParameters`

Defining the Processing Rule

For instructions to define Processing Rules, see the *webMethods Trading Networks--Building Your Trading Network* guide. The following sections describe specific settings that you should use when defining your Processing Rule for use with sWIFT messages.

Processing Rule Criteria

Set the following criteria on the **Criteria** tab of the Processing Rule.

Criteria tab field	Set to...
Document Type	The TN document type for the back-end format document.

Setting Processing Actions

On the **Action** tab, select **Perform the following actions**, and then set the following fields on the **Action** tab of the Processing Rule.

Action tab field	Set to...
Execute a service	A service you created that maps the back-end format document to webMethods DFD format. For more information about creating this service, see "Creating a Service to Map the Back-end Document to webMethods DFD Format" on page 46.
	Important! You *must* select to invoke this service synchronously by selecting **synchronous**.
Deliver Document By	**Immediate Delivery** and select **FINTransport**
	The FINTransport delivery service uses variables specified in the TPA to govern the creation and sending of the outbound SWIFT message.

Creating a Service to Map the Back-end Document to webMethods DFD Format

The service you create should contain logic to do the following:

1   Get the content of the back-end system document from the BizDocEnvelope to an IData object by invoking the wm.tn.doc.xml:bizdocToRecord service.

2   Map data from the back-end system document to webMethods DFD format, which represents the {B4} block of the SWIFT FIN message.

3   Convert the webMethods DFD format IData object to an XML String by invoking the pub.xml:documentToXMLString service.

4   Convert the XML String to bytes by invoking the pub.string:stringToBytes service.

5   Invoke the wm.tn.doc:addContentPart service to add the bytes to the BizDocEnvelope as a new content part. When you add the content part, you must name the content part `DFD Data`.

Using the Processing Rule at Run Time

At run time, to submit the back-end system document to the Trading Networks, invoke the wm.tn:receive service.

Receiving Inbound Messages from SWIFT

You can process inbound messages from SWIFT using either a business process or a service that is invoked by a Trading Networks Processing Rule.

Receiving Inbound Messages Using a PRT Business Process

To use a process run time (PRT) business process (also called a conversation), you use webMethods Modeler to create a process model that defines the steps that you want to take in your business process. For a guideline, see the sample process model (*CorpActions_Account Owner*) provided in the SWIFT FIN Module sample. For more information, see Appendix C, "webMethods SWIFT FIN Module Sample" in this guide.

After designing the process model, use webMethods Modeler to generate it. When you generate a process model, webMethods Modeler generates the run-time elements (for example, triggers, flow services, etc.) that are executed for the business process.

The first step of a business process is a step that waits for a document. At run time, you receive a SWIFT message via a transport protocol (MQ, CASmf, or AFT). The message is recognized as a TN document type and submitted to the PRT, which is the facility of the Integration Server that manages business processes. The PRT determines which process model to use for the document and either instantiates a new instance of the process model to start the business process, or when subsequent documents aimed at the business process arrive, the PRT delivers them to the appropriate business process to rejoin a running process.

The PRT uses the Conversation ID to determine whether documents belong to the same instance of a business process. All documents that belong to the same instance of a business process use the same Conversation ID.

Before You Can Receive Inbound Messages

■ When creating message records (using the wm.fin.dev:importFINItems service) for the SWIFT FIN messages you will be receiving, you must set the following parameters to **true**:

   ■ **createDocType**. Creates the TN document type for the message record.

   ■ **createTPA**. Creates the TPA for the message record.

■ Edit the message TPA parameters as necessary. For example, set the **Validate...** parameters such as **ValidateContent**.

Designing the Process Model

Your process model will typically contain the following steps:

1 Wait for the SWIFT message from the sender -and- invoke the wm.ip.cm:waitStepInit service. This service takes necessary actions to prepare for the business process. During the processing, it retrieves the TPA with the AgreementID set to the name of the TN document type. Processing will fail if the first step does *not* invoke this service -or- if the service fails to retrieve the TPA.

2 Invoke the wm.fin.validation:validateFinMsg, which validates the incoming message based on the parameters specified in the message TPA. The output of this service includes the *finIData* and *convertedFinIData* variables.

3 Map data from the inbound SWIFT message into a back-end format document using the following variables available in the pipeline.

   ■ *finIData*. This is the message in TAG format.

   ■ *convertedFinIData*. This is the format specified in the message TPA (for example, `TAG_BIZNAME`).

Executing the Business Process

When an inbound SWIFT message is received by the wm.fin.trp:receive service, Integration Server completes the following processing automatically:

■ Recognizes the SWIFT message using TN document types.

■ Forms a BizDocEnvelope for the SWIFT message.

■ Sets the Conversation ID for the SWIFT message. The Conversation ID is created in the following format: `Sender BIC-Unique Identifier`. The Unique Identifier is retrieved from the inbound message's MUR section in the {B3} block.

■ Saves the content of the SWIFT message to the Trading Networks database.

■ Passes the SWIFT message to the PRT. When the PRT receives the SWIFT message, it determines whether this is the first document for a business process or whether it is a document that is rejoining a running business process. If it is the first document of a business process, the PRT instantiates a new instance of the process model. Otherwise, it sends the document to rejoin a running business process.

Receiving Inbound Messages Using a Processing Rule

To use a Processing Rule to receive inbound SWIFT FIN messages, you define a Processing Rule in Trading Networks that invokes a service to validate and process the inbound message.

Before You Can Receive Inbound Messages

■ When creating message records (using the wm.fin.dev:importFINItems service) for the SWIFT FIN messages you will be receiving, you must set the following parameters to **true**:

■ **createDocType**. Creates the TN document type for the message record.

■ **createProcessingRule.** Creates the Processing Rule for the message record.

■ **createTPA**. Creates the TPA for the message record.

■ Edit the message TPA parameters as necessary. For example, set the **Validate...** parameters such as **ValidateContent**.

Creating the Service to Map the webMethods DFD Format to the Back-end Document

Create a mapping service using the following variables available in the pipeline.

■ *finIData*. This is the message in TAG format.

■ *convertedFinIData*. This is the format specified in the message TPA (for example, TAG_BIZNAME).

Then specify the name of this service in the message TPA **InboundProcessingRulesService** parameter.

Defining the Processing Rule

For instructions to define Processing Rules, see the *webMethods Trading Networks--Building Your Trading Network* guide. The following sections describe specific settings that you should use when defining your Processing Rule for use with a SWIFT message.

Processing Rule Criteria

On the **Criteria** tab of the Processing Rule, set the following criterion.

Criteria tab field	Set to...
Document Type	The TN document type for the SWIFT message.

Setting Processing Actions

On the **Action** tab of the Processing Rule, select **Perform the following actions**, and then set the following criterion.

Action tab field	Set to...
Execute a service	wm.fin.validate:validateFINMessage
	Important! You *must* select to invoke this service synchronously by selecting **synchronous**.

Using the Processing Rule at Run Time

When an inbound SWIFT message is received by the wm.fin.trp:receive service, the Integration Server does the following automatically:

■ Recognizes the SWIFT message using TN document types.

■ Forms a BizDocEnvelope for the SWIFT message.

■ Sets the Conversation ID for the SWIFT message.

■ Sets the Conversation ID for the SWIFT message. The Conversation ID is created in the following format: *Sender BIC-Unique Identifier*. The Unique Identifier is retrieved from the inbound message's MUR section in the {B3} block.

■ Saves the content of the SWIFT message to the Trading Networks database.

■ Submits the SWIFT message to the Trading Networks Processing Rules engine, which validates the message. If validation succeeds, the output of the validation includes the *finIData* and *convertedFinIData* variables. Trading Networks then invokes the service specified in the message TPA **InboundProcessingRulesService** parameter.

Configuring Transport Protocols

Overview

The transport protocols used with the webMethods SWIFT FIN Module enable you to send outbound messages to sWIFT as well as receive inbound messages from SWIFT. You can connect to SWIFT using one of the following transport protocols:

■ **IBM MQSeries**. If you are connecting to SWIFT through MQSA, you must install the webMethods MQSeries Adapter on the Integration Server. For more information about configuring and using MQ with the webMethods SWIFT FIN Module, see Chapter 5, "Configuring Transport Protocols" in this guide.

■ **CASmf**. If you are connecting to SWIFT through CASmf, you must install the WmCASmf Package on the Integration Server. For more information about configuring and using CASmf with the webMethods SWIFT FIN Module, see Chapter 5, "Configuring Transport Protocols" in this guide.

■ **AFT**. If you are using Automated File Transfer (AFT), you must configure the File Polling Listener and AFT settings in the message TPA. For more information about configuring and using the File Polling Listener with the webMethods SWIFT FIN Module as well as using File Drop for outbound messages, see Chapter 5, "Configuring Transport Protocols" in this guide.

Using the MQSeries Adapter to Communicate with SWIFT

The webMethods MQSeries Adapter enables the webMethods Integration Server to send and receive messages through the MQSeries interface and systems using an IBM MQSeries message queue. This capability lets you route documents—or any piece of information—from the Integration Server to systems that use MQSeries message queuing as their information interface. The webMethods MQSeries Adapter enables you to connect to SWIFT through MQSA. For detailed information about the webMethods MQSeries Adapter, see the *webMethods MQSeries Adapter User's Guide*.

> ⚠ **Important!** The following procedure assumes that you already have configured your MQSeries and SAA to communicate with one another, as well as installed the webMethods MQ Adapter. For more information, see the *webMethods MQSeries Adapter User's Guide*.

Configuring the MQSeries Adapter

To configure the webMethods MQSeries Adapter for the webMethods SWIFT FIN Module

1   If it is not running, start your Integration Server.

2   In the Server Administrator, click **WebSphereMQ...** under **Adapters**.

3   In the left Navigation Panel, click **Queue Manager Settings**.

4   Click **Add Queue Manager Settings** to define a Queue Manager using the instructions and field descriptions in the *webMethods MQSeries Adapter User's Guide*, and then click **Save**. Make sure to set the **Queue Manager Name** field to the same name that you used when defining your Queue Manager in MQSeries and SAA.

5   Define two queues using the following steps:

    a   Click **Queue Settings** in the left Navigation Panel.

    b   From the **Server Settings** drop-down list, select the Queue Manager that you defined in the previous step.

    c   Click **Find Queues on this Queue Manager**.

    d   Select the queues you defined for communication between MQSeries and SAA, and then click **Save**.

6   Using the instructions in the *webMethods MQSeries Adapter User's Guide*, click **Msg Handlers** in the left Navigation Panel, select your Queue Manager, and then click **Add Message Handler** to define two message handlers for the two queues you created in the previous step.

    a   For the 'put' message handler:

Set this field...	To...
Transaction Type	**IS-to-WebSphere MQ**
Target Client	**Non-JMS Compliant Client**

        1   Note the **Folder** and **Service Name** that you define for this handler service.

        2   Click **Generate Service**.

        3   Click **Continue**.

        4   Click **Enabled**.

    b   For the 'get' message handler:

Set this field...	To...
Transaction Type	**WebSphere MQ-to-IS**

      ■   Under **Inbound Service Details**, click **Listen**, and then set the following fields.

Set this field...	To...
Msg Service	`wm.fin.transport.MQSeries:getListenerService`

Set this field...	To...
Exception Service	The name of your exception service.
Target Client	**Non-JMS Compliant Client**

- Click **Generate Service**.

c Click **Continue**.

d Click **Enabled**.

7 In the TPA for the SWIFT message you will be sending and receiving using MQSeries, set the following fields:

Set this parameter...	To...
Transport	MQ
putMessage HandlerService	The name of the 'put' handler service, which you defined in the **Service Name** field.

For more information about TPAs, see Chapter 9, "Customizing Trading Partner Agreements" in this guide.

8 In Developer, navigate to the 'put' message handler service you created, and set the following:

- *msgHeader/MsgType* variable to 8

- *msgHeader/ReplyToQueueManager* variable to the same queue manager name that you defined in the MQSeries and SAA

- *msgHeader/ReplyToQ* variable to the appropriate 'get' queue

- *msgHeader/Report* variable to 3 to get reports for all of the SWIFT FIN messages you send to SAA

Using the CASmf Adapter to Communicate with SWIFT

The webMethods CASmf Package enables the webMethods Integration Server to send and receive messages through the CASmf interface. A CASmf client and Adapter enable the Integration Server to exchange information with other systems. This capability lets you route documents—or any piece of information—from the Integration Server to systems that use CASmf as their information interface. The webMethods CASmf Adapter enables you to connect to SWIFT using the CASmf Adapter.

webMethods CASmf Package Architecture

The following diagram illustrates how the webMethods CASmf Package fits into the webMethods Integration Platform architecture.

webMethods CASmf Package Architecture
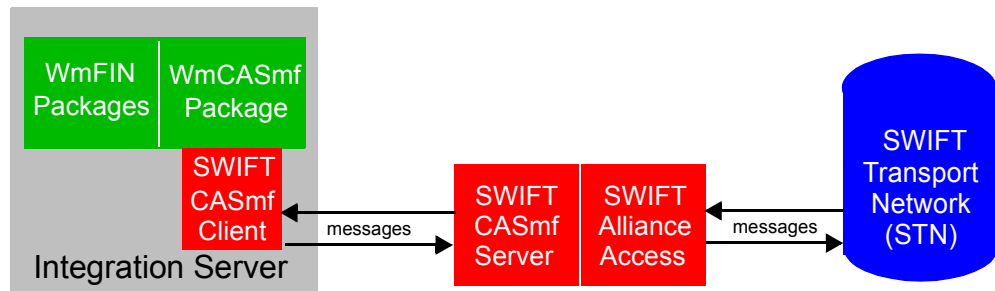
The following table describes the components that interact with the webMethods CASmf Package.

Component	Description
FIN Packages	The FIN packages included in the webMethods SWIFT FIN Module.
WmCASmf Package	Product described in this guide that transfers SWIFT FIN messages to and from SAA system using CASmf.
CASmf Client	The CASmf client enables the CASmf Package to interface with the CASmf server.
CASmf Server	The CASmf server enables communication between the CASmf client and SAA.
SWIFT Alliance Access (SAA)	SWIFT software configured to access the SWIFT Transport Network (STN), SWIFT's original network accessed using x.25 transport technologies.
SWIFT Transport Network (STN)	The existing SWIFT interface, a computer system provided and operated by the user, which enables communication with the SWIFT network.

Configuring the CASmf Adapter

Important! The following procedure assumes that you already have configured your CASmf server and SAA to communicate with one another. For more information, see your CASmf server and SWIFT documentation.

To configure the webMethods CASmf Adapter for the webMethods SWIFT FIN Module

1   Install a CASmf client on the same machine as webMethods Integration Server. For more information, see your CASmf documentation.

2   Install the webMethods CASmf Package on the same machine as webMethods Integration Server. This package can be installed with the webMethods SWIFT FIN Module. For more information, see Chapter 2, "Installing the webMethods SWIFT FIN Module" in this guide.

3   From the *webMethods6\IntegrationServer*\packages\WmCASmf\lib directory, copy the WmCASmf.dll (WmCASmf.so in Unix) file into the *webMethods6\IntegrationServer*\lib directory.

   where *webMethods6\IntegrationServer* indicates where the webMethods Integration Server is installed.

4   Specify your *wm.casmf.send.mapid* and *wm.casmf.receive.mapid* variables in the following configuration file:

   webMethods6\IntegrationServer\packages\WmCASmf\config\wmcasmf.cnf

   These mapIDs must match exactly the two **I_mapid** fields in the CASmf client dmapid.dat file. Typically, this is located in the following directory:

   $CASmfInstallationFolder\dat

   where *$CASmfInstallationFolder* is the directory in which the CASmf client is installed. You also can locate this file using the folder listed in your DATTOP environment variable.

Note: The *wm.casmf.send.message.folder* variable in the wmcasmf.cnf file specifies the folder in which all outbound SWIFT FIN messages will be queued before being sent to SWIFT via the CASmf Adapter. You do not need to change this location, but you may do so at your discretion.

5   In the TPA for each SWIFT message you will be sending and receiving using CASmf, set the following field:

Set this field...	To...
Transport	**CASmf**

For more information about TPAs, see Chapter 9, "Customizing Trading Partner Agreements" in this guide.

6 In Integration Server Administrator, create a scheduling service to run the wm.casmf.trp:casmfSendReceiveSchedule service on intervals:

a Under **Server**, click **Scheduler**. The Scheduler screen appears.

b Click **Create a scheduled task**. The **Service Information** screen appears.

c Set the fields according to the instructions in the *webMethods Integration Server Administrator's Guide*, but set the fields listed in the next two steps as indicated.

d Under **Schedule Type and Details** and **Repeating Tasks With a Simple Interval**, select **Repeating** and then set the following fields:

■ In the **Interval** field, webMethods recommends that you set this service to run at intervals of at least 15-20 minutes.

■ Select the **Repeat from end of invocation** check box.

e Click **Save Tasks**.

For more information about this service, see Appendix A, "webMethods SWIFT FIN Module Services" in this guide.

Using AFT to Communicate with SWIFT

Important! To use Automated File Transfer, you must have the WmFlatFile package installed, which is installed by default with the webMethods Integration Server.

Automated File Transfer (AFT) enables the webMethods Integration Server to exchange information with other systems. If you are using AFT to receive inbound SWIFT FIN messages through the File Polling Listener, and File Drop capabilities to send outbound SWIFT FIN messages, the File Polling Listener and the message TPA must be configured properly.

Configuring AFT for Inbound Messages

To configure the webMethods File Polling Listener for the webMethods SWIFT FIN Module

1 In the Server Administrator, click **Security ▸ Ports ▸ Add Port**.

2 Select **webMethods/FilePolling** and click **Go to Step 2**.

3 Configure the File Polling Listener's general fields as described in the "Configuring Ports" section of the "Configuring the Server" chapter in the *webMethods Integration Server Administrator's Guide.*

4 Set the following fields so that the File Polling Listener handles SWIFT FIN messages properly.

Set this field...	To...
Content Type	`application/x-wmflatfile`
Folder location	The fully qualified path of the folder in which SAA will be sending SWIFT FIN messages. The folder must be accessible to both SAA and webMethods Integration Server.
Processing Service	`wm.fin.transport.AFT:processInboundFile`

Configuring AFT for Outbound Messages

To configure AFT using File Drop capabilities to send outbound SWIFT FIN messages using the webMethods SWIFT FIN Module, you must complete the following procedure.

To configure File Drop for the webMethods SWIFT FIN Module

1   Map a network directory in which you want to drop files to be picked up by SAA, which can be any directory in the webMethods Integration Server.

2   In the TPA for the SWIFT message, set the following parameters:

Set this field...	To...
Transport	**AFT**
Folder location	The fully qualified path of the folder in which the webMethods Integration Server will be dropping SWIFT FIN messages. The folder must be accessible to both SAA and the Integration Server.
FileExtension	**.inp**

Working with BIC and BIC+ Lists

Overview

The SWIFT home page in the Server Administrator enables you to import and search BIC and BIC+ lists.

SWIFT Home Page

Importing BIC and BIC+ Lists

A BIC list provides a list of valid BICs for all existing SWIFT financial institutions. All SWIFT FIN messages are validated against this BIC list to make sure the sender and receiver are valid.

Before you can import a BIC or BIC+ list into webMethods Integration Server, you must create a database table to hold the BIC data. webMethods provides database scripts in multiple formats to create an empty database table () to hold the BIC information that you import.

After you have created the BIC table(s) in your database, you then can import the most recent list of SWIFT BIC or BIC+ codes into your database table from the CD provided to you by SWIFT. To do so, you use the **Import BIC List** tool on the SWIFT home page in the Server Administrator.

> ⚠️ **Important!** SWIFT provides BIC lists in fixed-length ASCII format. SWIFT provides BIC+ lists to you in Access database format. To import a BIC+ list into the webMethods SWIFT FIN Module, you first must convert the list into fixed-length ASCII format.

To create a BIC database table

1   With your database server running, make sure you have a database in which to add a BIC table.

2   Import the following file into your selected database to create an empty BIC table in your database:

webMethods6\IntegrationServer\packages\WmFIN\config\bic\create_BIC_*db*.sql

where *db* is the type of database sever you are using, such as Oracle. webMethods provides scripts for SQL Server, Oracle, and Sybase databases.

For the procedure to complete this step, see you database server documentation.

To import a BIC or BIC+ List

1   In the Server Administrator, on SWIFT home page, click **Import BIC List**.

SWIFT Import BIC List Screen

2   Select **BIC** or **BIC+** to indicate the type of BIC list you are importing.

⚠ **Important!** If you are importing a BIC+ list, you first must convert the list into fixed-length ASCII format.

3   In the **File Name** box, type the full directory path and name of the BIC or BIC+ list file that you want to import.

4   Click **Import BIC List**. The new BIC list is imported into your BIC database table and made available to your webMethods SWIFT FIN Module. This process may take a few minutes.

> **Note:** A sample BIC+ database is located in the
> *webMethods6\IntegrationServer*\packages\WmFINSamples\data directory. Import
> this database into the sample.

Searching for BICs

Using the **Search BIC List** tool on the SWIFT home page in the Server Administrator, you
can import the most recent SWIFT Bank Identification Code (BIC) lists provided by
webMethods into the webMethods SWIFT FIN Module.

A BIC list provides a list of valid BICs for all existing SWIFT financial institutions. All
SWIFT FIN messages are validated against this BIC list to make sure the sender and
receiver BICs are valid.

To search a BIC list

1   In the Server Administrator, on SWIFT home page, click **Search BIC List**.

SWIFT Search BIC List Screen

2   On the **Search BIC List** screen, provide the information on which you want to search.
    You must enter information in at least one field, but to limit your search, you can
    enter information in as many fields as necessary.

> **Note:** All fields on this screen are case-sensitive.

Field Name	Description
Code	The institution's BIC code. For example, type ABCDEFGHIJK.
Institution	The institution's name. For example, type Citibank.
Branch	The institution's branch name. For example, type Main.
City	The institution's city. For example, type Miami.
Modified Flag	From the drop-down list, select the modification flag for which you want to search: ■ **New**. Identifies a new BIC entry. ■ **Update**. Identifies a BIC entry currently being updated. ■ **Modified**. Identifies a modified BIC entry. ■ **Deleted**. Identifies a deleted BIC entry.
Location	The institution's location. For example, type Mall.
Country Name	The institution's country. For example, type USA.

3 Click **Search**. The **Search BIC List** screen displays up to the first 50 matching BICs.

SWIFT Search BIC List Results Screen

Search criteria	
Code	
Institution	%Citi%
Branch	
City	
Modified Flag	
Location	
Country Name	
Documents Retrieved	50

Received documents

Code	Institution	Branch	City	Modified Flag	Location
BHWAPLPWBIA	BANK HANDLOWY W WARSZAWIE SA A MEMBER OF CITIGROUP(FORMERLY BANK HANDLOWY W WARSZAWIE S.A.)		BIALYSTOK	D	15213 BIALYSTOK
BHWAPLPWBIE	BANK HANDLOWY W WARSZAWIE SA A MEMBER OF CITIGROUP(FORMERLY BANK HANDLOWY W WARSZAWIE S.A.)		BIELSKO BIALA	D	43300 BIELSK BIALA

Creating Message Records and Validation Rules

Overview

To send and receive SWIFT FIN messages, you first must create a message record for each SWIFT message that you will be sending and receiving. You also can use network and usage validation rules in your maps to validate your messages.

You use SWIFT message records to create inbound and outbound maps that define how this particular message will be handled. For information about mapping, see Chapter 10, "Mapping a SWIFT FIN Module Process" in this guide.

Creating Message Records

To create a message record for a SWIFT message, you must do the following:

■ Install the SWIFT message DFD (which includes the message parsing template) into the webMethods SWIFT FIN Module. For more information, see "Installing SWIFT Message DFDs and Parsing Templates" on page 68 in this chapter.

■ Run the wm.fin.dev:importFINItems service located in the WmFINDev package. For more information, see "Running the wm.fin.dev:importFINItems Service" on page 68 in this chapter.

Installing SWIFT Message DFDs and Parsing Templates

To create a message record for a SWIFT message, you must install the message DFD (which includes its parsing template) using webMethods Installer. For more information about installing message DFDs using the webMethods Installer, see Chapter 2, "Installing the webMethods SWIFT FIN Module" in this guide.

For more information about parsing templates, see Appendix B, "XML Parsing Templates" in this guide.

Note: When the nov03 version of a message is not available, you can use the nov02 version of that message to create a nov03 version. To do so, copy the applicable DFDs and template files from `\IntegrationServer\packages\WmFINDev\import\nov02` to `\IntegrationServer\packages\WmFINDev\import\nov03`.

Running the wm.fin.dev:importFINItems Service

When you run the wm.fin.dev:importFINItems service located in the WmFINDev package for a particular SWIFT message DFD, the service will create the following items for each message:

■ IS Document Type

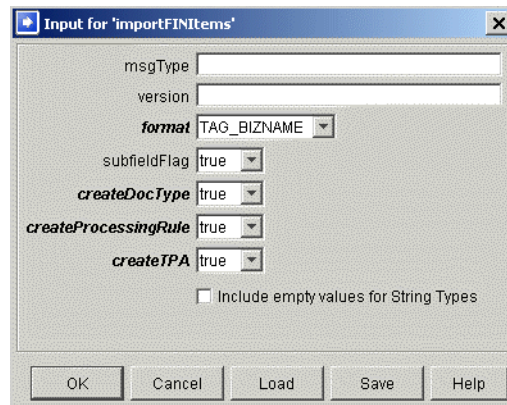■ Network validation rules (if available)

■ Market Practice rules (if available)

■ TN document type

■ Trading Partner Agreement (in "Proposed" status)

■ Processing Rule to process an inbound SWIFT message

The SWIFT message parsing template and DFD is copied from the WmFINDev package into the WmFIN package. For example, for nov03 version, the parsing template and DFD files from `WmFINDev\import\nov03` are copied into `WmFIN\config\dfd\nov03`. The IS document types are stored in the WmFINMessages package in the appropriate version folder and message category (for example, `wm.fin.doc.nov03.cat5`).

To run the wm.fin.dev:importFINItems service

1 In the Developer, navigate to the wm.fin.dev:importFINItems service located in the WmFINDev package, and then click ▶. The Input dialog box appears.

Input dialog box

2   Complete the following fields, and then click **OK**:

Field	Description
msgType	FIN message type. For example, 564.
version	FIN version. For example, nov03.
format	The format of the generated blocks and fields for the input FIN message. Valid values:

format (continued):

■ **TAG_BIZNAME** (default). SWIFT Message tag followed by the business name specified in the message DFD. For example, 23G_Function of the Message. This format provides the best balance between readability and performance. It provides DFD business names while retaining the field tag. This format causes half of the performance penalty of **BIZNAMEONLY** because lookups are used only when receiving a message.

■ **TAGONLY**. SWIFT Message tag only. For example, 23G:. This is the simplest format and is suited to those already familiar with SWIFT and specific messages. This format provides the best performance.

■ **BIZNAMEONLY**. Business name specified in the message DFD only. For example, Function of the Message. Although this format is the most readable, it also carries the largest performance penalty.

■ **XMLTAG**. XML-compatible tag name. For example, F23G. This format cannot contain colons or tags that begin with a number. This format enables you to take advantage of the XML-specific services and functionality provided by the Integration Server, such as pub.xml:documentToXMLString, etc.

Field	Description
subfieldFlag	Indicates whether the IS Document Type generated for this FIN Message is parsed to the field or subfield level. Valid values:

■ **true**. Parse to the subfield level.

■ **false**. Parse to the field level.

Note: For inbound SWIFT FIN messages, setting the **subfieldFlag** variable to true causes the webMethods SWIFT FIN Module to automatically parse messages to the subfield level and then remove the SWIFT delimiter (/) from between subfields. For outbound SWIFT FIN messages, setting the **subfieldFlag** variable to true causes the webMethods SWIFT FIN Module to add the SWIFT delimiter (/) between subfields.

Field	Description
createDocType	Indicates whether to create and insert a TN document type for this message. The TN document type is used to recognize an incoming message. ■ **true** (default). Create and insert a TN document type. ■ **false**. Do not create and insert a TN document type. **Important!** You should always set this field to **true**.
createProcessing Rule	Indicates whether to create a Trading Networks Processing Rule for this message. After the message is recognized, the Processing Rule specifies how the message should be processed. ■ **true** (default). Create a Processing Rule. ■ **false**. Do not create a Processing Rule. **Important!** If you will be using the Processing Rules to manage the execution of SWIFT FIN messages, you should always set this field to **true**.
createTPA	Indicates whether to create a Trading Networks trading partner agreement (TPA) for this message. A TPA is a set of parameters that you use to govern how SWIFT FIN messages are exchanged between two trading partners. ■ **true** (default). Create and insert a TPA. ■ **false**. Do not create and insert a TPA. **Important!** You should always set this field to **true**.

Creating Validation Rules

webMethods provides network validation rules for a number of commonly used message types. In addition to these rules, the webMethods SWIFT FIN Module enables you to create network validation rules for additional messages as well as create usage validation rules.

Note: Although Market Practice rules function in much the same way as validation rules, the configuration of Market Practice rules differs from that of validation rules. For more information about Market Practice rules, see Chapter 14, "Working with Market Practices" in this guide.

Creating Network Validation Rules

When the webMethods SWIFT FIN Module sends and receives a SWIFT message, it validates the message at the individual field level or across the fields using network validation rules as specified by SWIFT. The webMethods SWIFT FIN Module sends a message only when the message structure, syntax, and validation rules are applied.

webMethods provides network validation rules for version nov03, as flow services, for you to use with the SWIFT FIN messages that you import. When you create a message record, the corresponding network rule (as a flow service) is imported into the Integration Server and will be placed in the WmFINMessages package along with the message record.

After the webMethods SWIFT FIN Module syntactically validates a message, it executes the corresponding network rule. Any validation errors will be aggregated and reported in the webMethods Integration Server and process run time (PRT) error logs, if you are using process models.

webMethods provides network validation rules for the following SWIFT FIN messages:

- *MT 103 Single Customer Credit Transfer*
- *MT 202 General Financial Institution Transfer*
- *MT 300 Foreign Exchange Confirmation*
- *MT 502 Order to Buy or Sell*
- *MT 515 Client Confirmation of Purchase or Sale*
- *MT 535 Statement of Holdings*
- *MT 536 Statement of Transactions*
- *MT 537 Statement of Pending Transactions*
- *MT 540 Receive Free*
- *MT 541 Receive Against Payment*
- *MT 542 Deliver Free*
- *MT 543 Deliver Against Payment*

- *MT 545 Receive Against Payment Confirmation*
- *MT 547 Deliver Against Payment Confirmation*
- *MT 548 Settlement Status and Processing Advice*
- *MT 564 Corporate Action Notification*
- *MT 565 Corporate Action Instruction*
- *MT 566 Corporate Action Confirmation*
- *MT 567 Corporate Action Status and Processing Advice*
- *MT 568 Corporate Action Narrative*
- *MT 900 Confirmation of Debit*
- *MT 910 Confirmation of Credit*
- *MT 940 Customer Statement Message*
- *MT 950 Statement Message*

You can create additional network validation rules for particular messages by writing individual services based on the message documentation (pdf) provided by SWIFT. To use

a new validation rule, you must specify the service you created in the **ValidationRule** parameter in the TPA for the particular SWIFT message.

For more information about TPAs, see Chapter 9, "Customizing Trading Partner Agreements" in this guide.

Creating Usage Validation Rules

Usage rules exist only for certain messages when being exchanged between two specific partners. webMethods does not provide built-in usage rules because they vary by trading partner pairs, but you can create usage rules for particular messages by writing individual services based on the message documentation (pdf) provided by SWIFT. To implement a usage rule, you must specify the service you created in the **UsageRule** parameter in the TPA for the particular SWIFT message.

For more information about TPAs, see Chapter 9, "Customizing Trading Partner Agreements" in this guide.

Defining Trading Partner Profiles and TN Document Types

Overview

Trading partner profiles help define how you and your trading partners exchange SWIFT FIN messages. TN document types enable webMethods Trading Networks to identify a type of business document and specify what to extract from the business document.

This chapter provides you with information about defining trading partner profiles and TN document types 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 SWIFT FIN 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, used in conjunction with trading partner agreements (TPAs), and process models, define how you and your trading partners exchange SWIFT FIN messages. For example, a process model defines what actions your company takes in certain transactions, as well as the actions you expect your trading partners to perform during those transactions. In fact, the concise definition of profiles, the configuration of process models, and the application of TPAs are what enable you to interact successfully with your trading partners.

For the webMethods SWIFT FIN Module, you define a single trading partner profile for yourself (**My Enterprise**) and then create a process model to support each role you will perform in exchanging messages with your trading partners. You also must define a trading partner profile for each trading partner with whom you will be exchanging messages. When you create a process model, you will use the various trading partner profiles to help define the sender and receiver of the message.

For information about creating process models, see Chapter 11, "Creating or Modifying a Process Model" in this guide. For general information about process models, see the *webMethods Modeler User's Guide*.

Defining Your Enterprise Profile

Before defining your trading partner profiles in Trading Networks and exchanging messages with your trading partners, 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. The following section provides you with the fields that you are required to complete when defining your enterprise profile for use with the webMethods SWIFT FIN Module.

Required Profile Fields

Profile information is displayed on the Trading Networks Console Profile Assistant **Corporate** tab. The following table lists and describes the required fields you must complete when defining your enterprise profile for use with the webMethods SWIFT FIN Module.

Required Profile Field for My Enterprise	Description
Corporation Name	The name of your enterprise.
External ID Type	**BIC**
External ID Type Value	Your enterprise's BIC.

Note: The BIC **External ID Type** is added to the webMethods Trading Networks database when you start the webMethods SWIFT FIN Module for the first time.

For descriptions of other fields you must complete when you defining your enterprise profile, see the *webMethods Trading Networks--Building Your Trading Network* guide.

Defining Your Trading Partners' Profiles

Each trading partner with whom you want to exchange SWIFT FIN messages 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. The following section provides you with the fields that you are required to complete when defining your trading partner profiles for use with the webMethods SWIFT FIN Module.

Required Profile Fields

Profile information displays on the Trading Networks Console Profile Assistant **Corporate** tab. The following table lists and describes the required fields you must complete when defining a trading partner profile.

Required Profile Field for Trading Partner	Description
Corporation Name	The name of the trading partner.
External ID Type	**BIC**
External ID Type Value	Your trading partner's BIC.

For descriptions of other fields you must complete when you define a trading partner profile, see the *webMethods Trading Networks--Building Your Trading Network* guide.

Defining TN Document Types

TN document type*s* are definitions that tell webMethods Trading Networks how to identify a type of business document and specify the attributes that Trading Networks is to extract from the business document.

When you run the wm.fin.dev:importFINItems service, which is located in the WmFINDev package, to create a record for a particular SWIFT message DFD, you can specify that the service also create the external TN document types for this message. For more information about creating message records, see Chapter 7, "Creating Message Records and Validation Rules" in this guide.

When the webMethods SWIFT FIN Module receives a message, it invokes a Trading Networks service to recognize the type of business document by using the TN document types that were created along with your message records or that you created yourself. When Trading Networks recognizes the TN document type of the business document, Trading Networks extracts specific pieces of information from the business document based on the attributes specified in the TN document type.

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

Defining Your Own Internal TN Document Types

You might want to create your own internal TN document types or customize one of the TN document types created using the wm.fin.dev:importFINItems service in the WmFINDev package.

> ⚠️ **Important!** Do not modify any of the provided external TN document types. If you do, the incoming SWIFT message will not join the business process (also called a conversation).

When you define an internal TN document type, you specify the root tag from within the SWIFT message that the TN document type is to match, as well as the XML queries that Trading Networks uses to extract the SenderID and ReceiverID attributes from the message.

To define an internal TN document type

1   Start the Trading Networks Console.

2   In the Selector Panel, click **My Enterprise**.

3   Select **View ▸ Document Types**.

4   Select **Types ▸ New**.

5   In the **Create New DocType** dialog box, select the **XML** document type category from the list, and click **OK**.

6   In the **Document Type Details** screen, in the **Name** field, type the name you want to give to the internal TN document type (for example, `Internal Corporate Action`).

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

8   In the **Root Tag** field, type the value of the root tag of your internal document (for example, `InternalCorporateAction`).

9   Click **OK**.

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

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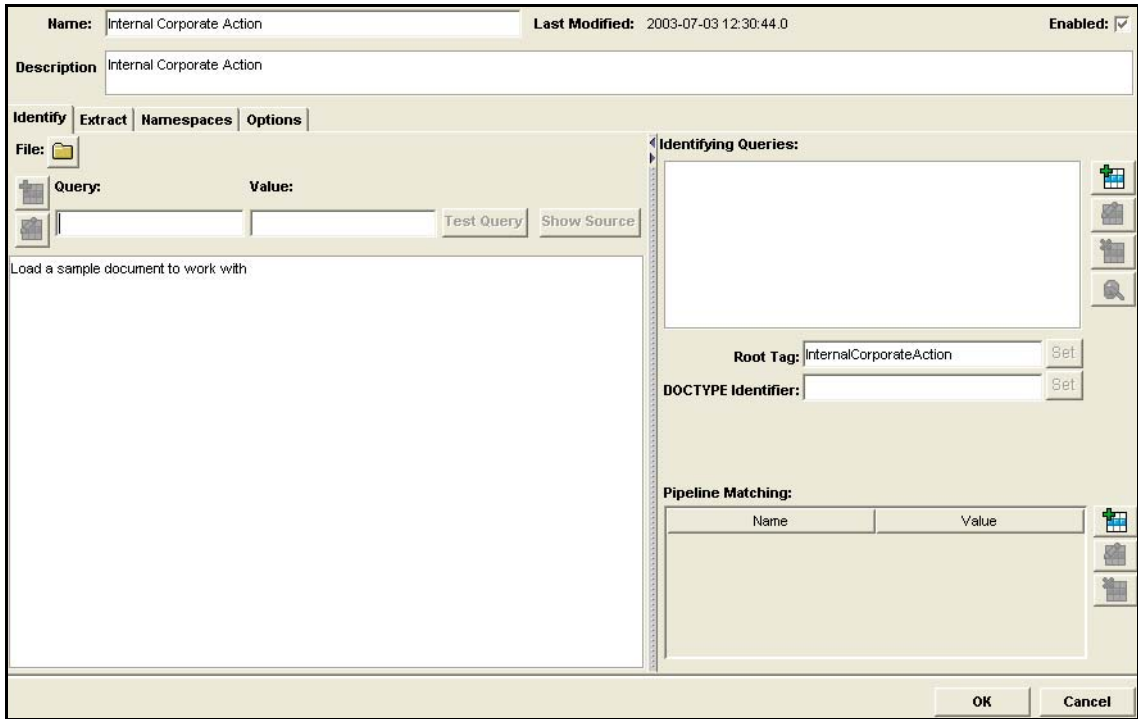
> **Note:** You can tell whether a TN document type is internal or external because an external TN document type *always* has a pipeline matching variable of *processVersion*. The TN document type in the preceding figure has *no* pipeline matching variable, so it is an internal TN document type.

---

10 Trading Networks uses the extracted attributes to determine who is sending the message and to whom to send the message. On the **Extract** tab, specify *SenderID* as one of the attributes to extract.

   a Click **Add an Attribute** 🔲.

   b In the **Add an Attribute** dialog box, in the **Name** list, select **SenderID**.

   c Select the **Required** check box.

   d In the **Description** field, enter a description for the attribute.

   e In the **Query** field, enter the query to extract the SenderID attribute, which is illustrated in the following figure.

Completed Add Attribute Dialog Box

f   In the **Detail** list, select **BIC**.

g   Click **OK**.

11   Repeat step 10 for the *ReceiverID*.

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

Completed Extract Tab of the Document Type Details Screen

Customizing Trading Partner Agreements

Understanding Trading Partner Agreements

A *Trading Partner Agreement (TPA)* is a set of parameters that you use to govern how SWIFT FIN messages are exchanged between two trading partners. TPAs also enable you to define Market Practice requirements for individual markets. Using Trading Partner Agreements (TPAs), the webMethods SWIFT FIN Module supports customization of SWIFT FIN messages based on specific trading partner pairs.

You modify and view TPAs in the Trading Networks Console **Agreement Details** screen. For detailed information about working with TPAs in the Trading Networks Console, see the *webMethods Trading Networks--Building Your Trading Network* guide.

How Does the webMethods SWIFT FIN Module Identify a TPA?

Every SWIFT message in the webMethods SWIFT FIN Module is associated with a TPA. Every TPA is uniquely identified by a Sender, Receiver, and Agreement ID. During a business process between trading partners, the webMethods SWIFT FIN Module uses this information to retrieve the TPA for a specific sender-receiver pair and to process the messages exchanged.

When using a business process to manage the execution of SWIFT FIN messages, the Agreement ID is *always* the first TN document type name used to start the process. For the sender of a SWIFT message in a business process, the first TN document type name is typically the TN document type name representing the internal back-end document, such as `InternalCorporateAction`. For the receiver of a SWIFT message in a business process, the first TN document type name is typically the TN document type name of the first SWIFT message received, such as `MT541`.

Important! To process messages according to a TPA, both sender and receiver TPAs must have an Agreement Status of "Agreed".

Modifying TPAs

When the webMethods SWIFT FIN Module creates a message record, it automatically creates the TPA for the particular SWIFT message. You can view and modify the TPA by double-clicking to open the selected "Proposed" TPA on the Trading Networks **Agreement** tab to display the **Agreement Details** screen.

Important! The **SubfieldFlag** input parameter in the TPA supports the parsing of SWIFT messages to the subfield level. If you change the value of the **SubfieldFlag** after creating the message records, you must delete the message records pertaining to the TPA and recreate them.

In the **Agreement Details** screen, the **Sender** and **Receiver** fields initially display a default value of "Unknown".

In the SWIFT TPA for each message that you created, you must specify the **Sender**, **Receiver**, and **Agreement ID**. The **Agreement ID** is a placeholder for the TN document type name of the internal business document received from the back-end system (for example, "InternalCorporateAction") or of the SWIFT message received from a partner (for example, "MT541"). After the webMethods SWIFT FIN Module recognizes and associates an incoming SWIFT message with a particular TN document type, it uses the TN document type name in the business document, such as "MT541", to find a TPA matching that sender-receiver pair. Although your TPA identifies the business names of the sender and receiver, the webMethods SWIFT FIN Module identifies the sender and receiver using their BIC.

Agreement Details Screen

Sender, Receiver, and Agreement ID

Double-click a heading to view the parameters that fall under that heading.

In the preceding figure, the **Agreement ID** value of MT541 indicates that the TPA is for a receiver because the starting business document for a receiver's TPA is always a SWIFT message. If this TPA were for a sender, the Agreement ID would be the name an internal business document from a back-end system such as InternalCorporateAction.

Important! The webMethods SWIFT FIN Module does **not** support a TPA in which either the Sender is known (for example *Company1*) and the Receiver is *Unknown* or in which the Sender is *Unknown* and the Receiver is known (for example, *Company2*). The webMethods SWIFT FIN Module supports only those TPAs in which either both the Sender and Receiver are known (for example, *Company1* and *Company2*, respectively) or *Unknown*.

Agreement Details Field Descriptions

The following table lists and describes the basic TPA information at the top of the **Agreement Details** screen that you need to understand.

TPA Information	Description
Sender	The name of the trading partner that has the sender role in the TPA. You select the sender from the profiles defined on your Trading Networks system, including your own profile.
Receiver	The name of the trading partner that has the receiver role in the TPA. You select the receiver from the profiles defined on your Trading Networks system, including your own profile.
Agreement ID	An application-specific field that uniquely identifies the type of agreement between two partners. For example, MT541 or *InternalCorporateAction*.
IS Document Type	The IS document type wm.fin.doc:UserParameters specifies the parameters that you define for a SWIFT TPA. It is located in the WmFINTransport package.

For step-by-step instructions about how to create or modify a TPA, see the *webMethods Trading Networks--Building Your Trading Network* guide.

TPA SWIFT-Specific Input Parameters

To modify TPA input parameters, click 🐾 on the **Agreement Details** screen.

TPA Input Screen

The following table describes the TPA input parameters for a SWIFT-specific TPA that you need to complete.

TPA Section	Parameter	Description
FINProcessInfo	**Transport**	Name of the transport protocol used to send and receive SWIFT FIN messages. Select one of the following values: ■ **MQ** (Default). Use webMethods MQ Adapter to transport SWIFT FIN messages. ■ **CASmf**. Use CASmf to transport SWIFT FIN messages. ■ **AFT**. Use Automated File Transfer to transport SWIFT FIN messages. ■ **TEST**. Enables you to test the processing of your FIN message without sending the message to SWIFT.
	MessageType	SWIFT FIN message type identifier, such as 541, which identifies MT 541, Receive Against Payment.
	ISDocumentName	The IS document type name for this particular message. You can find the IS document type for each message record you have created in the WmFINMessages package. For example, wm.fin.doc.nov03.cat5:MT541.
	Version	Version number of the SWIFT message record being used. For example nov03.
	MessageFormat	The format of the generated blocks and fields for the input FIN message. Valid values: ■ **TAG_BIZNAME** (default). SWIFT Message tag followed by the business name specified in the message DFD. For example, 23G_Function of the Message. ■ **TAGONLY**. SWIFT Message tag only. For example, 23G:. ■ **BIZNAMEONLY**. Business name specified in the message DFD. For example, Function of the Message. ■ **XMLTAG**. XML-compatible tag name. This format cannot contain colons or tags that begin with a number. For example, F23G.

TPA Section	Parameter	Description
	SubfieldFlag	Indicates whether to parse SWIFT messages to the field or subfield level. Select one of the following values: ■ **true** (default). Parse to subfield level. ■ **false**. Parse to field level.
	InboundProcessingRuleService	**Optional** - Indicates whether to manage the execution of this SWIFT message using a Processing Rule you have created. To do so, type the name of the service you created. For more information, see "Receiving Inbound Messages Using a Processing Rule" on page 49 of Chapter 4, "Sending and Receiving SWIFT Messages" in this guide.
	ValidateContent	**Optional** - Indicates whether to validate the SWIFT message.
	ValidateBICPlus	**Optional** - Indicates whether to validate the BIC+ information included in the message.
	ValidateNetworkRules	**Optional** - Indicates whether to validate the SWIFT message against the network rules for this message.
	NetworkValidationService	**Optional** - Name of the network validation service to validate the network rules for this message.
	ValidateMarketPracticeRules	**Optional** - Indicates whether to validate the SWIFT message against the Market Practice rules for this message.
	MarketPracticeRulesService	**Optional** - Name of the Market Practice Rules service to validate the Market Practice rules for this message.
	ValidateUsageRules	**Optional** - Indicates whether to validate the SWIFT message against the usage rules for this message.
	UsageRulesService	**Optional** - Name of the usage validation service to validate the usage rules for this message.
Message Header	This section contains information to be populated in blocks {B1}, {B2}, and {B3} of the outbound SWIFT message.	
	Logical Terminal	The logical terminal identifier defined in SAA.

TPA Section	Parameter	Description
	ApplicationIdentifier	Identifies the application within which the message is being sent or received. Select one of the following values: ■ **F**. SWIFT FIN - Use this setting for all FIN user-to-user, FIN system, and FIN service messages. ■ **A**. GPA - Use for GPA system and GPA service messages. ■ **L**. GPA - Use for certain GPA service messages. For example, `LOGIN`, `LAK`, `ABORT`.
	ServiceIdentifier	Identifies the type of data being sent or received. Select one of the following values: ■ **01**. Identifies user-to-user messages. ■ **21**. Identifies message acknowledgements.
	Priority	Indicates the priority with which the message is being delivered to the receiver. Select one of the following values: ■ **N**. Delivered with Normal priority. ■ **U**. Delivered with Urgent priority. ■ **S**. Delivered with System priority.
	DeliveryMonitoring	**Optional** - Enables the sender to request one of the following levels of delivery monitoring: ■ **None**. Do not perform delivery monitoring. ■ **1**. Non-delivery warning - Requesting automatic MT010 non-delivery warning if message is not delivered within the obsolescence period, which is 15 minutes for U **Priority**, 100 minutes for N **Priority**. ■ **2**. Delivery notification - Requesting automatic MT011 delivery notification after the message is delivered. ■ **3**. Both - Requesting both automatic non-delivery warning and delivery notification.
	FINCopyServiceIdentifier	**Optional** - (Used with FINCopy messages only.) Three-character system ID used to support access to multiple services with the same CBT.

TPA Section	Parameter	Description
	BankingPriority	**Optional** - Four-character field agreed upon by two or more parties to indicate priority.
	ValidationFlag	**Optional** - Flag indicating whether special validation needs to be completed at SWIFT. For more information, see SWIFT documentation.
	AddresseeInformation	**Optional** - Information from the central institution to the receiver of the payment message. This information is used in the input of *MT097, FIN Copy Message Authorization Refusal Notification*.
	Training	Indicates whether a message is sent to or received from a test and training logical terminal.
MQSeriesInfo	**putMessageHandlerService**	The name of the service that you generated when creating a message handler service for your IS-to-WebSphere MQ transport. For more information, see "Using the MQSeries Adapter to Communicate with SWIFT" on page 52 of Chapter 5, "Configuring Transport Protocols" in this guide.
AFT	**folder**	Fully qualified path of the folder you want the File Polling Listener to poll for this SWIFT message. For example, `c:\folder\bic.dat`.
		Important! This folder must be accessible by both Integration Server and SAA.
	extension	**Optional** - File extension of the files to be received from the AFT **folder**. The default is **inp**.

Mapping a SWIFT FIN Module Process

What Is "Mapping" a Message?

"Mapping" a message refers to the process of assigning the structure, values, or content of the message to a new message, that is, mapping the values, data, and information into a new message. You need to map messages because, typically, your back-end systems have different message formats than that of SWIFT.

> **Note:** To configure the webMethods SWIFT FIN Module to parse to the subfield level, set the *subfieldFlag* variable to `true` in the wm.fin.dev:importFINItem, wm.fin.dfd:convertTagFormat, or wm.fin.dfd:convertBizNameFormat services. For information about these services, see Appendix A, "webMethods SWIFT FIN Module Services" in this guide. For inbound messages, if you want to manually parse messages to the subfield level (without using subfield option), you must manually remove the SWIFT delimiter (/) from between subfields. For outbound messages, if you want to manually identify subfields (without using the subfield option), you must add the SWIFT delimiter.

Why Do You Create an Outbound Mapping Service?

You create an outbound mapping service to translate an outbound back-end proprietary message format (for example, Oracle Financials) to SWIFT message format. Elements of the proprietary message need to be mapped to corresponding elements of a SWIFT message (for example, *MT 541, Receive Against Payment*). Extra elements in the back-end message are ignored, as long as values are mapped to **all** elements in the SWIFT message.

Examples of outbound mapping services are located in the WmFINSample package.

Why Do You Create an Inbound Mapping Service?

You create an inbound mapping service to map each element of an inbound SWIFT message to a corresponding element in your back-end proprietary message format. For example, if you use Oracle Financials and you want to receive a SWIFT message via the webMethods SWIFT FIN Module, you create an inbound mapping service that maps each element of the SWIFT message to a corresponding element in the Oracle Financials format.

Examples of inbound mapping services are located in the WmFINSample package.

Example of Mapping a Message

The following figure illustrates the process of mapping a message. For further explanation, see the text that follows the figure.

Example of Mapping a Message

Trading Partner A   **Trading Partner B**
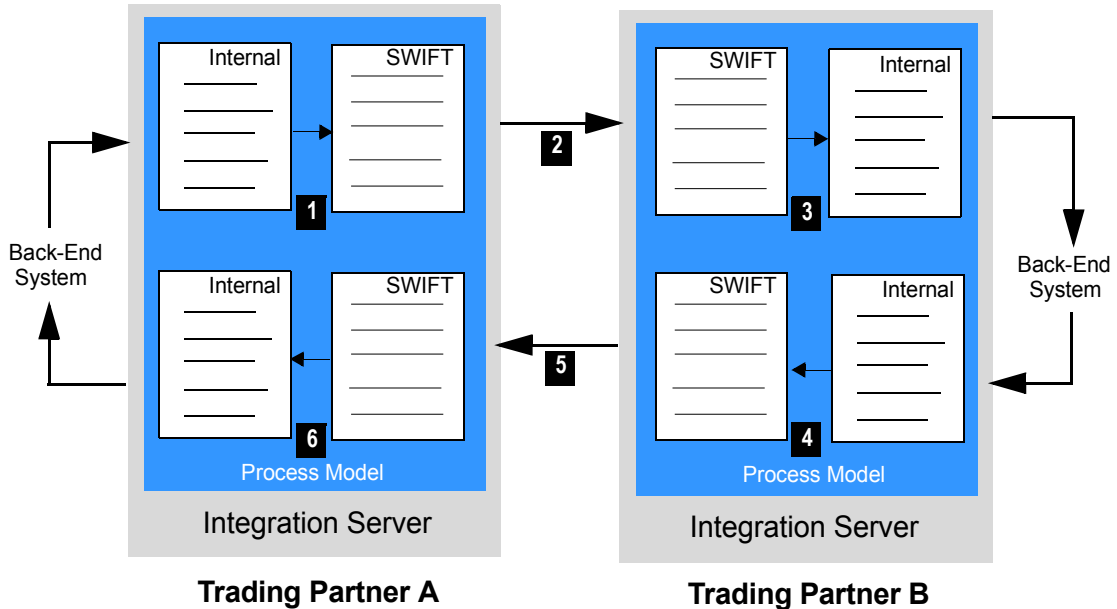
Step	Description
1	Trading Partner A uses an outbound mapping service to map an internal message from a back-end format to SWIFT format.
2	Trading Partner A sends the SWIFT message to Trading Partner B.
3	Trading Partner B receives the SWIFT message and uses an inbound mapping service to map the SWIFT message to an internal message. After the internal message is mapped, it is in a format that Trading Partner B's back-end system can process.
4	Trading Partner B responds by using an outbound mapping service to map an internal message from a back-end format to SWIFT format.

Step	Description
5	Trading Partner B sends the SWIFT message to Trading Partner A.
6	Trading Partner A receives the SWIFT message and uses an inbound mapping service to map the SWIFT message to an internal message. After the SWIFT message is mapped, it is in a format that Trading Partner A's back-end system can process.

Creating an Outbound Mapping Service

In webMethods Developer, you create an outbound mapping service by creating a service that contains one or more MAP entities, which do the actual mapping from your back-end message, through any desired intermediate steps, to the IS document type for the appropriate outbound SWIFT message.

Input/Output to Use

The input to the outbound mapping service is a TN document type representing your back-end document in a variable named *bizEnv*.

> **Note:** For information about retrieving your back-end document from the *bizEnv* variable, refer to the outbound mapping services in the WmSamples package.

The output of the outbound mapping service is the SWIFT message in the *documents\payload* IData object in the pipeline. For example, if you are mapping your back-end message to the MT 564, the output of the mapping service would be *MT564* in the *documents\payload* IData object. See the figure in "Example of an Outbound Mapping Service" on page 98 in this chapter.

The output of the outbound mapping service **must** be placed in the *documents\payload* IData object. Because the webMethods SWIFT FIN Module has to convert an IData object into an XML string before sending the SWIFT message to the trading partner, it must know the precise location of the IData object.

Also, you **must** place the Agreement ID for this TPA in the *documents\tpaAgreementID* IData object. For more information about Agreement IDs, see Chapter 9, "Customizing Trading Partner Agreements" in this guide.

> **Note:** If the documents for your back-end system have DTDs, you can automatically import an external DTD in webMethods Developer to provide a starting point for mapping. To do so, create an external record and specify the source as *XML, DTD,* or *XML Schema.*

Flow Operations to Use

In the flow service, you insert a MAP operation and use the service pipeline to map elements of the IS document type from your back-end message to all elements of the IS document type for the appropriate SWIFT message. Built-in IS document types for all versions of the SWIFT FIN messages that you imported are located in the WmFINMessages Package in wm\fin\doc*version**category* folders as illustrated in the following figure.

Records Available for Mapping
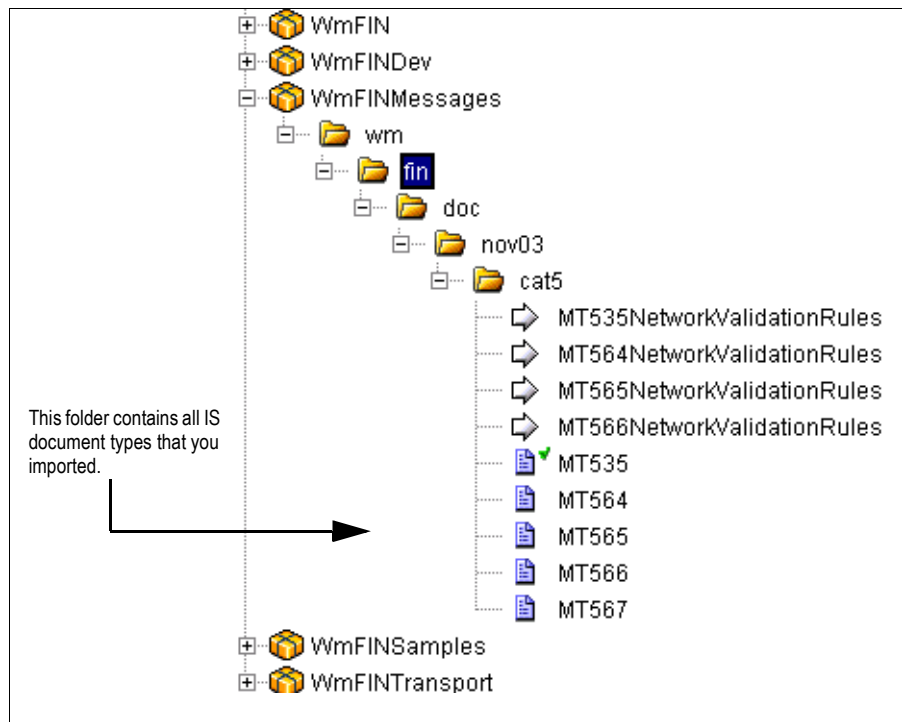
Example of an Outbound Mapping Service

If you are the sender in a business process and are implementing a version of MT 564, an outbound mapping service converts your back-end business document to a DFD and then to a SWIFT message. One of these outbound mapping services might look like the following figure. For further explanation, see the text that follows the figure.

Outbound Mapping Service
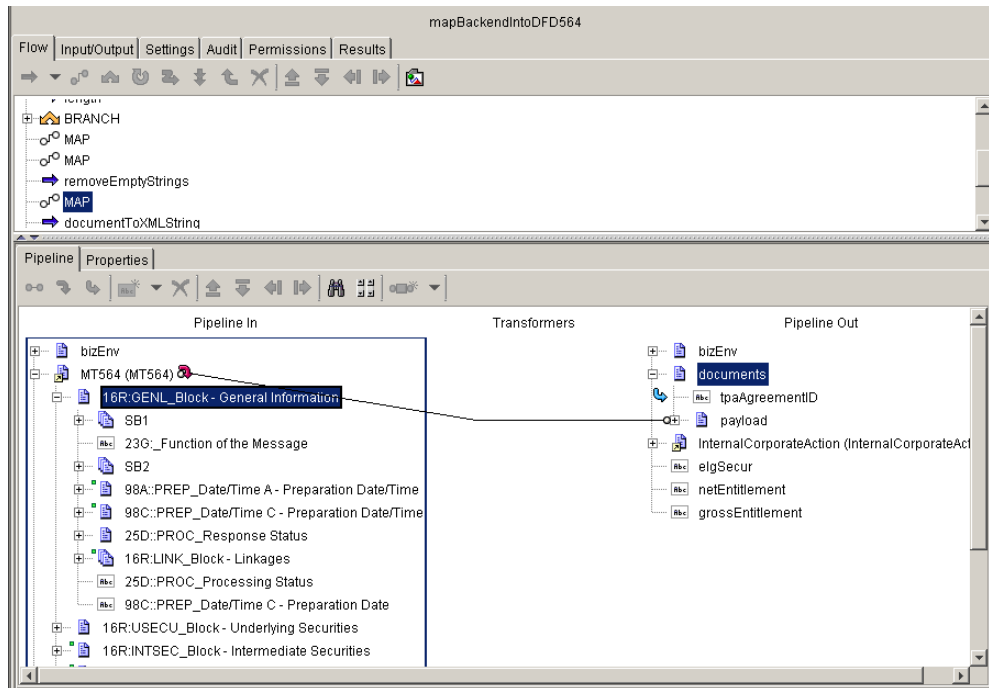
Under the **Pipeline Out** heading, in the *documents\payload* IData object, the *MT564* not in parentheses is the IData object, or SWIFT message, being sent to the trading partner and the *MT564* in parentheses is the IS document type that defines the SWIFT message. The name of the IData object is case sensitive.

Creating an Inbound Mapping Service

In webMethods Developer, just as with outbound mapping services, you create an inbound mapping service by creating a new service that contains one or more MAP entities, which do the actual mapping from the received SWIFT FIN messages, through any intermediate steps, to the format of your back-end messages. The inputs to any inbound mapping service include the following variables:

■ *finIData*. This is the message in TAG format.

■ *convertedFinIData*. This is the format specified in the message TPA (for example, TAG_BIZNAME).

Example of an Inbound Mapping Service

If you are the receiver in a business process and are implementing a version of MT 564, the inbound mapping service might look like the following figure.

Inbound Mapping Service
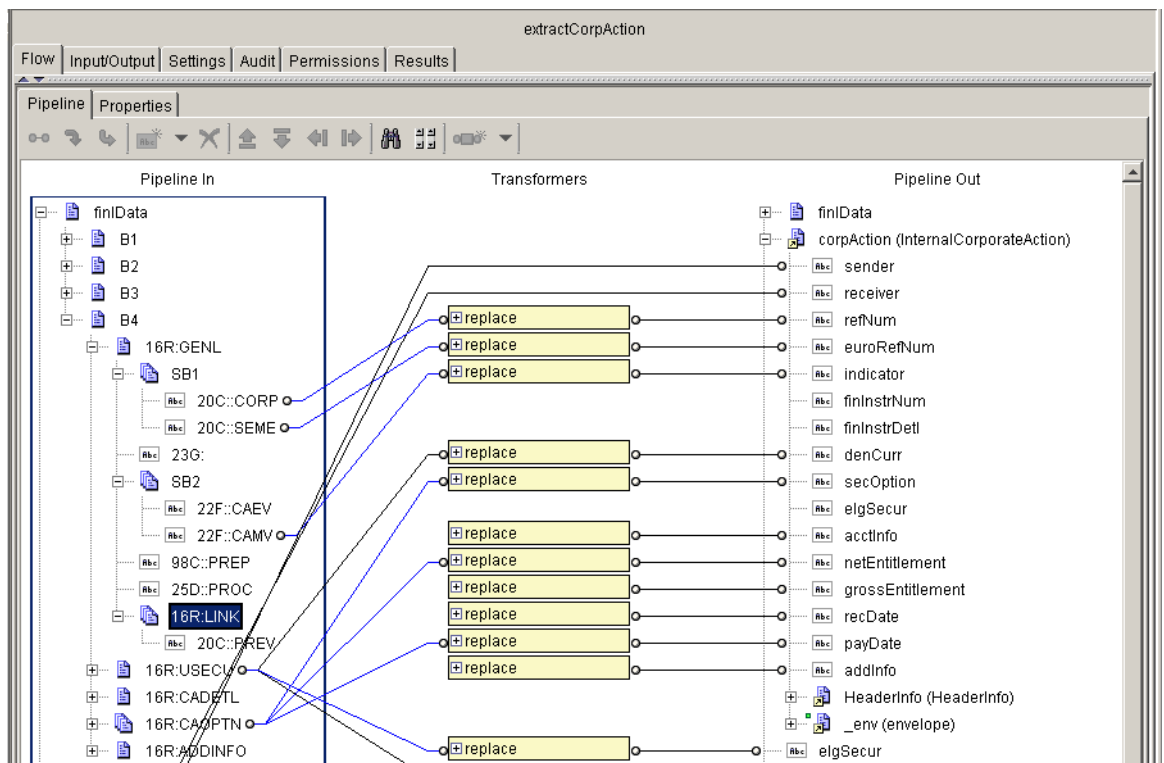
> **Note:** The *finIData* object that contains information from the inbound SWIFT message is mapped to the *InternalCorporateAction* IData object representing your back-end document.

Reusing Mapping Services

In the webMethods SWIFT FIN Module, you can reuse mapping services for trading partners that submit the same business document format. For example, you can use the same mapping services for Trading Partner A and Trading Partner B if they both always submit business documents in the same document format to the webMethods SWIFT FIN Module. As the receiver of those documents, you need to define only one inbound mapping service for both trading partners because the message format versions are the same.

Creating or Modifying a Process Model

What Is a Process Model?

A process model is a diagram that represents a business process. The following figure illustrates a sample process model. For further explanation, see the text that follows the figure.

Sample Process Model

Steps represent an automated process performed by a computer.

Transitions are lines between steps that indicate the flow of execution.

External groups box steps that are performed outside of an enterprise.
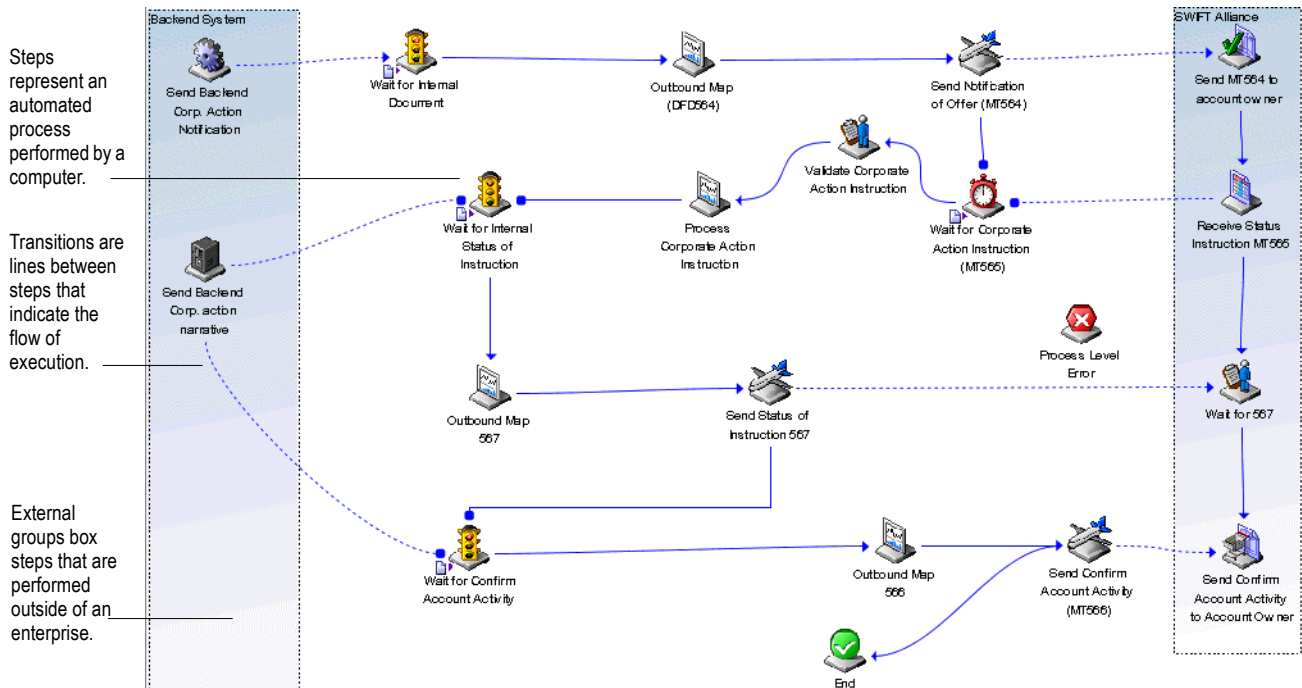
A process model consists of:

■ **Steps.** The basic unit of work in a process model.

■ **Transitions.** The lines between steps that indicate the execution order of steps within the process model.

■ **Groups.** Clustering steps to represent different organizational boundaries within a process model.

■ **Annotations (notes or text).** The labels, notes, and explanatory text in a process model.

For more information about steps, transitions, groups, and annotations, see the *webMethods Modeler User's Guide*.

Working with Process Models

The steps in a process model determine how the process run time (PRT) conducts a business process, which includes how the process run time processes a SWIFT message. For information about the process run time, see "Run-Time Architecture/Components" on page 23 in this guide and to the *webMethods Modeler User's Guide*.

webMethods provides process model samples that you can use to create your own process models. You specify how the process model is to interact with your back-end systems by editing the services that are invoked by the steps of the process model, identifying the TN document types that you have created for steps that are waiting for messages, and specifying inbound and outbound mapping services. When creating a process model, you also assign TN roles to TN document types and a focal role to the process model.

You use webMethods Modeler to create the process models based on the messages you are using. For more information about process models and Modeler, see the *webMethods Modeler User's Guide*.

Note: You can use webMethods Trading Networks Processing Rules instead of the Process Run Time to manage the execution of SWIFT FIN messages. For information about creating Processing Rules, see Chapter 4, "Sending and Receiving SWIFT Messages" in this guide.

What Is a Role?

In webMethods Modeler, you assign roles to TN document types that are associated with steps that wait to receive a message, such as an *MT 564*.

TN Roles

When you identify a TN document type that you created for a particular wait step, Modeler prompts you to enter the **Sender Role** and **Receiver Role** in the **Set Roles** dialog box. For example, if you are identifying a TN document type for a particular wait step with the name of a *MT 564* that you are sending to a trading partner, you would enter "Bank" as the **Sender Role** and "Broker" as the **Receiver Role**.

Focal Role

When you create a process model, you also assign a focal role to the process model. A *focal role* specifies the role of the user for a particular business process. You assign a focal role for your enterprise in the **Properties** dialog box in Modeler.

Using Process Model Samples

webMethods provides the following process model samples to assist you in creating your process models:

■ CorpAction_AccountOwner.model

■ CorpAction_AccountServices.model

You can find the process model samples in the following location:

webMethods6\IntegrationServer\packages\WmFINSample\data

where *webMethods6\IntegrationServer* is the directory in which the webMethods Integration Server is installed.

Creating or modifying a process model consists of such tasks as specifying the TN document type for a particular wait step, assigning inbound and outbound mapping services to invoke during the respective mapping steps, and so on. When you modify or create a process model, you must generate and update your process model in webMethods Modeler to create a business process, and then enable that business process in webMethods Monitor.

To see these process models used in a sample, see Appendix C, "webMethods SWIFT FIN Module Sample" in this guide. For more information about creating process models, see the *webMethods Modeler User's Guide*.

Monitoring a Process

Why Monitor a Business Process?

You monitor a webMethods SWIFT FIN Module business process to track the state of a particular SWIFT transaction. For example, suppose that you send a message to a trading partner. The next day, the trading partner calls to say that they did not receive your message. In the webMethods SWIFT FIN Module, you can view the activity of the business process to determine what the current status is and what activity has occurred during the progress of the business process. Following the same example, a reason why the trading partner did not receive the message could be because the transaction encountered an error, such as a timeout error. By monitoring the business process, you can determine whether this was the cause. You can see whether your message was resent, how many times, and with what success. By viewing the current status of and the progress of a transaction, you can take appropriate action, which might include retrying the business process, editing your trading partner profiles, or editing your process models.

Finding Business Process Information

To monitor business processes, you have a number of sources from which you can draw information to determine the status of your business processes and the activity of the various involved software entities. The following table lists and describes the primary sources.

Source	Description
webMethods Monitor	Use this tool to monitor and manage business processes. For information about Monitor, see the next section, "Using Monitor", and the *webMethods Monitor User's Guide*.
webMethods Trading Networks **Transaction Analysis** screen	Use this log to query and analyze results of documents that are sent or received by Trading Networks. For information about transaction analysis, see the *webMethods Trading Networks--Building Your Trading Network* guide.
webMethods Integration Server error log	Use this log to retrieve server-related error and exception messages that occur during an invocation of a service directly or indirectly from a business process.

Using Monitor

Monitor is a web-based user interface that you can use to examine instances of your process models. Monitor displays information about instances of your process models by accessing data from the Process Logging Database. For information about the Process Logging Database, see "Run-Time Architecture/Components" on page 23 of Chapter 1, "Concepts", in this guide.

Using Monitor, you can:

- Search for a business process by name, status, or date range.

- Search specifically for a business process that ended in error.

- See a graphical overview of a business process.

- Examine information about a business process and its execution, for example, status of the business process, status and iteration of process model steps, and so on.

- View services used in the business process.

- Perform control tasks to affect the state of a business process:

  - Suspend and resume a business process.

  - Enable and disable a business process.

  - Start and stop a business process.

  - Edit and resubmit process model steps.

For more information about Monitor and for procedures to perform the aforementioned tasks, see the *webMethods Monitor User's Guide*.

Repairing and Resubmitting Messages

Overview

Using the **Repair/Resubmit Messages** tool on the SWIFT home page in the Server Administrator, you can manually repair invalid messages and then resubmit those messages to SWIFT.

SWIFT Home Page
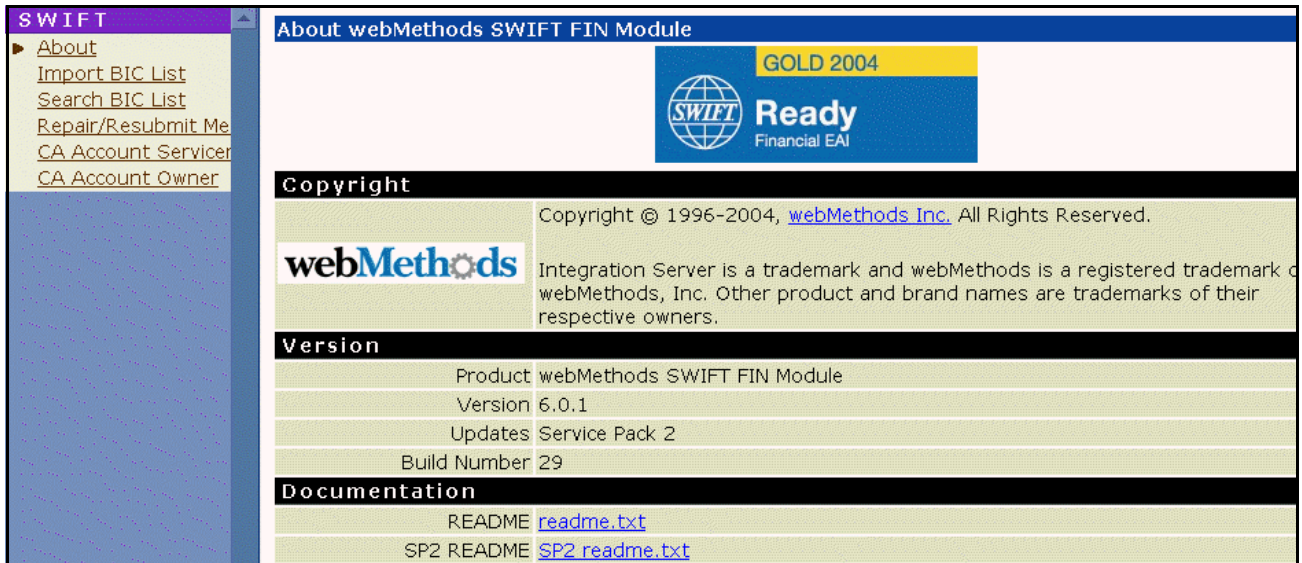
You click **Repair/Resubmit Messages** to display a list of all errors that have occurred in the webMethods SWIFT FIN Module. When you click a particular error, the details of this error display and you now can correct the error and resubmit the transaction to SWIFT.

Repairing and Resubmitting Messages to SWIFT

To repair and resubmit a SWIFT message

1   In the Server Administrator, on SWIFT home page, click **Repair/Resubmit Messages**. The Repair Messages screen appears listing recently failed processes.

Repair Messages Screen

2   Click the **ID** of the error you want to repair. A detail screen appears displaying information about the message and listing the blocks in error. To view the specific errors, click the **+** signs for each block.

Repair Message Detail Screen

A red message indicates the specific error.

Repair Message Detail Screen with Error

3   Correct the information as necessary, and then click **Update** for the corrected fields.

4 At the bottom of the screen, click **Resubmit**. Integration Server resumes sending the message from wherever the error occurred during the message.

5 You are returned to the **Repair/Resubmit Messages** screen. The message you repaired is listed under **Resubmit Processes**.

Working with Market Practices

Overview

Market Practices are specific requirements for individual markets. Using Trading Partner Agreements (TPAs), the webMethods SWIFT FIN Module supports customization of SWIFT FIN messages based on specific trading partner sender-receiver pairs. For example, two partners trading within France might have different processing requirements for their SWIFT FIN messages than two trading partners within Austria.

SWIFT FIN messages that are exchanged between two partners may have additional fields and/or a subset of key words. The webMethods SWIFT FIN Module enables you to maintain multiple versions of a given message that conform to different Market Practices.

Creating Market Practices

You create a Market Practice by creating an alternate version of the SWIFT message based on an original message record. Doing so maintains the original content of the message record.

To create a Market Practice

1 On your file system, create identically named folders (for example, `FrenchMarket`) in the following directories:

- *webMethods6\IntegrationServer*\packages\WmFINDev\import directory

- *webMethods6\IntegrationServer*\packages\WmFIN\config\dfd

2 Copy `dfd000.xml` from `WmFIN\config\dfd\novXX` to `WmFIN\config\dfd\FrenchMarket`, where `novXX` is the SWIFT message version, for example `nov03`.

3 Copy the `dfd*.xml` file (for example, `dfd541.xml`) for your message from `WmFINDev\import\novXX` to `WmFINDev\import\FrenchMarket`.

4 Open your `French Market\dfd*.xml` and edit as necessary by deleting or changing existing information.

Sample of dfd.541.xml File

```xml
<?xml version="1.0" ?>
<constraints>
  <!--   Text Block for MT541   -->
  <field name="11A::DENO" type="PatternType">
    <bizName>Currency A - Currency of the Denomination</bizName>
    <pattern>//<CUR></pattern>
  </field>
```

5 In the webMethods Developer, run the wm.fin.dev:importFINItems service for the message. The **Input** screen appears.

importFINItems Service Input Screen

6 On the **Input** screen for the importFINItems service, set the **version** field to the name of the new folder (for example, FrenchMarket). Set the remaining fields as desired.

7 In the Trading Networks Console, click the **Agreements** tab, open the TPA for the particular SWIFT message, and then click 📇 on the **Agreement Details** screen to display the **Inputs** screen.

8 On the **Input** screen, set the **ISDocumentName** parameter to the location the new message record (for example, wm.fin.doc.FrenchMarket.cat1:MT103).

9 Set the **Version** parameter to a new Market Practice version name (for example, FrenchMarket).

10 Set the **MarketPracticeRulesService** parameter to the Market Practice rule for this SWIFT message.

Completed Market Practice TPA Input Screen

For more information about TPAs, see Chapter 9, "Customizing Trading Partner Agreements" in this guide.

Creating Market Practice Rules

The webMethods SWIFT FIN Module provides 16 common Market Practice rules for Category 5 SWIFT FIN messages. You can create additional Market Practice rules by writing services based on message documentation (pdf) provided by SWIFT.

To use a new Market Practice rule, you must specify the service you created in the **MarketPracticeRulesService** parameter in the TPA for the particular SWIFT message. For more information about TPAs, see Chapter 9, "Customizing Trading Partner Agreements" in this guide.

Migrating Messages

Overview

As SWIFT introduces new messages to handle the increasingly complex financial transactions carried out across the SWIFT network, messages are retired to make way for improved and more flexible messages. For example, changes were required for the main securities transaction messages *MT520 Receive Free*, *MT521 Receive Against Payment*, *MT522 Deliver Free*, and *MT523 Deliver Against Payment*. These ISO 7775 messages were superseded by the ISO 15022 messages *MT540*, *MT541*, *MT542*, and *MT543* respectively. However, there may be a need for some SWIFT message users to convert between the old and new standard messages to allow time for old systems to be upgraded.

The wm.fin.migration:templateToMap service converts XML templates, representing migrations or mappings of fields between ISO 7775 and ISO 15022 messages, into Integration Server services with identical migration maps.

What Are Migration Templates?

A migration template is required to map each SWIFT message. For example, a migration template is required to convert from *MT520* to *MT540*, and another is required to convert from *MT540* back to *MT520*. All migration templates are stored in a single directory that is defined in the webMethods SWIFT FIN Module configuration. The wm.fin.migration:templateToMap migration service in the WmFINDev package reads the required migration template when it is executed, creating a service.

The name of the migration template is based on the names of the message types it converts. Each message type has a unique ID, which is usually a three-digit number. Typically, the name of a migration template file follows the following:

Format of Migration Template Name	Used for…
finConvMT*xxx*ToMT*yyy*.xml	Map source message MT*xxx* to MT*yyy*.

Each migration template file consists of lines representing either the mapping of a single source field to another within the target message (`lineAttribute` lines), or the setting of a target message field to a default value (`defaultAttribute` lines). An example of a migration template appears below:

Sample Migration Template File

```
<lineAttribute businessElemName="Related Message Reference" fromTag="21"
fromCode="" fromQualifier="" toTag=":20C:" toCode="" toQualifier="TRRF"
toSyntax=":4!c//16z" fieldMapFn=""
directToMapField="B4.MGENL.MLINK[0].M20CRELA"/>
```

Understanding Migration Template Syntax

Line Attribute Syntax of a Migration Template

The `lineAttribute` elements in the migration template define a mapping from a single source field to one or many target fields, possibly converting the value using mapping functions. The syntax of a `lineAttribute` directive is shown below. (Note that optional parameters are shown in square braces [ ].)

```
<lineAttribute businessElemName="name" fromTag="tag" fromCode="code"
fromQualifier="qualifier" toTag="toTag" toCode="toCode"
toQualifier="toQualifier" [toSyntax="fieldSyntax"]
fieldMapFn="functionName[:parameters][,functionName]"
directToMapField="directMapField"/>
```

The full set of parameters are described below.

Element	Values	Description
businessElemName	field name	The business element name for this field.
fromTag	tag	The SWIFT tag for the field to be mapped. This value will match the field's ID value in the SWIFT message parsing template. For example, 103, B2, or 88D.
fromCode	code value	The code associated with the tag. This may be left empty if no code is needed for mapping. This is used only for ISO 7775 fields.
fromQualifier	tag qualified	The qualifier associated with the SWIFT tag. This may be left empty if no qualifier is needed for mapping. This is used only for ISO 15022 fields.
toTag	tag	The SWIFT tag for the field to be mapped into in the target message. The value will match the field's tag value from the SWIFT message parsing template, that is, '103', 'B2', or '88D'.
toCode	code	The code associated with the tag to be mapped to. This may be left empty if no code is needed for mapping. This is used only for ISO 7775 fields.

Element	Values	Description
toQualifier	qualifier	The qualifier associated with the tag to map to. This may be left empty if no qualifier is needed for mapping. This is used only for ISO 15022 fields.
toSyntax	SWIFT syntax format	**Optional** - The SWIFT syntax for the field structure. This is for documentation purposes. For example, [ISIN1!e12!c][4*35x].
fieldMapFn	List of functions and parameters	The list of mapping functions to be called on the value that is being mapped. List of functions, are separated by commas. Parameters for functions are separated by colons. The value is passed to each mapping function in the order they are listed in (see "Syntax of fieldMapFn Element" on page 34).
directToMapField	Path	Fully-specified path of the field in the target message record. This field allows the toTag, toCode, and toQualifier element values to be overridden, and to write the mapping value directly to a field in the target message record.

fieldMapFn Element Syntax

The fieldMapFn element is used to specify one or many functions that will be applied to the value that is being mapped from source to target messages. If multiple functions are used, the result from each function is passed to the input of the next function in line. The list of available functions may be found in the wm.fin.mappingFunctions folder of the WmFIN package.

Examples of the use of this element are shown in the table below.

Example	Description
InsertSlashes	Single mapping function call.
StripPartyIdentifier,InsertSlashes	Multiple mapping function called. Mapping functions are separated by commas.
truncate:30	Mapping function call with parameter. Parameters are separated by a colons.

Default Attribute Syntax of a Migration Template

The `defaultAttribute` elements in the SWIFT message template define the fields to which a default value will be written. The syntax of a `block` directive is shown below. (Note that optional parameters are shown in square braces [ ].)

```
< defaultAttribute target="targetField" value="value" function="function"
set="placeValue" />
```

Element	Values	Description
target	Path	Fully-specified path of the field in the target message record in which to set the default value
value	Value to use on target	The SWIFT tag for the field to be mapped. The value will tie up to the field's ID value from the SWIFT message parsing templates.
function	List of functions and parameters	The code associated with the tag. Lists of functions are separated by commas. Parameters for functions are separated by colons. This may be left empty if no code is needed for mapping. This is used only for ISO 7775 fields.
set	start | end	Indicates whether to set the default value before all mapping occurs (`start`), or after (`end`).

Working with Migration Services

The webMethods SWIFT FIN Module fully supports the migration of the following four SWIFT nov03 version messages from ISO 7775 to ISO 15022 and ISO 15022 to ISO 7775:

- *MT 520* to *MT 540*

- *MT 521* to *MT 541*

- *MT 522* to *MT 542*

- *MT 523* to *MT 543*

The four migration map services for these messages are available for nov03 version messages only:

- *finConvMT520ToMT540.xml*

- *finConvMT521ToMT541.xml*

- *finConvMT522ToMT542.xml*

- *finConvMT523ToMT543.xml*

In the same folder, webMethods supplies partial migration templates for many more SWIFT FIN messages. You can modify these templates as needed as well as create additional migration services.

Creating Additional Migration Services

You migrate SWIFT nov03 version messages from ISO 7775 to ISO 15022 using map templates supplied by webMethods in the `webMethods6\IntegrationServer\packages\WmFINDev\import\mapTemplates` directory.
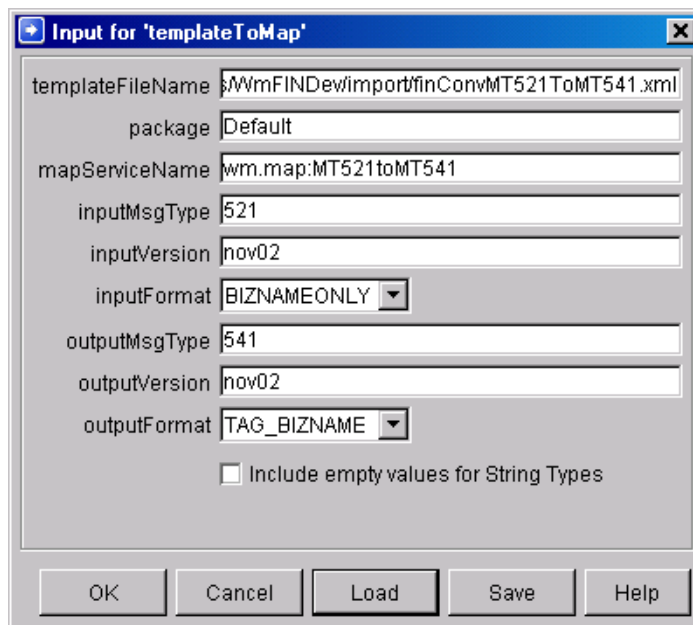
You create one service for the migration of each message from ISO 7775 to ISO 15022 and from ISO 15022 to ISO 7775. You then can reference those services in your inbound and outbound maps.

To create a migration service

1   In the webMethods Developer, navigate to and select the
wm.fin.dev.migration:templateToMap service, and then click ▶ to run the service. The **Input** screen appears.

Input Screen

2 Complete the following inputs for the service:

Input Variable	Description
templateFileName	The absolute path and name of the migration template. For example, `c:\webMethods6\IntegrationServer\packages\WmFINDev\import\mapTemplates\finConvMT521ToMT541.xml`.
package	The development package in which you want to store the service you are creating. You should save the service in the package that contains your inbound and outbound maps. For example, `Default`.
mapServiceName	Name of the migration service you are creating. For example, `wm.map:MT521toMT541`.
inputMsgType	SWIFT message type identifier of the message to be migrated. For example, `MT521`.
inputVersion	Version number of the SWIFT message record message to be migrated. For example, *nov03*.
intputFormat	The format of the generated blocks and fields for the input FIN message. Valid values:
	■ `TAG_BIZNAME` (default). SWIFT Message tag followed by the business name specified in the message DFD.
	■ `TAGONLY`. SWIFT Message tag only.
	■ `BIZNAMEONLY`. Business name specified in the message DFD.
	■ `XMLTAG`. XML-compatible tag name. This format cannot contain colons or tags that begin with a number. For example, `F23G`.
outputMsgType	SWIFT message type identifier of the message after migration. For example, `MT541`.

Input Variable	Description
outputVersion	Version number of the SWIFT message record message when migrated. For example, `nov03`.
outputFormat	The format of the generated blocks and fields for the output FIN message. Valid values: ■ `TAG_BIZNAME` (default). SWIFT Message tag followed by the business name specified in the message DFD. ■ `TAGONLY`. SWIFT Message tag only. ■ `BIZNAMEONLY`. Business name specified in the message DFD. ■ `XMLTAG`. XML-compatible tag name. This format cannot contain colons or tags that begin with a number. For example, `F23G`.

webMethods SWIFT FIN Module Services

WmCASmf Package

The WmCASmf Package contains support services used to send and receive messages from CASmf.

wm.casmf.init

The services in this folder perform initialization routines for CASmf.

wm.casmf.init:shutdown

This service unregisters the application with CASmf.

wm.casmf.init:startup

This service registers the application with CASmf Input Name.

wm.casmf.trp

This folder contain services to send and receive messages from CASmf.

wm.fin.transport.casmf:SendReceiveSchedule

This is the service to be run as a scheduled job by the user. This service will first send all the outbound messages to CASmf using the value specified for `wm.casmf.send.mapid` in `wmcasmf.cnf file` located in *webMethods6\IntegrationServer*\packages\WmCASmf\config folder. Then incoming messages are retrieved from CASmf using the value specified for wm.casmf.receive.mapid in wmcasmf.cnf file. The messages received are then published to the Integration Server/Broker for processing by wm.fin.trp:receive service in WmFIN package.

wm.casmf.trp:processOutboundMessage

This service is invoked when publishing an outbound message with the **Transport** parameter in the message TPA is set to CASmf. This service writes the SWIFT message to a unique file name located in the folder specified by the *wm.casmf.send.message.folder* property in the file *webMethods6\IntegrationServer* \packages\WmCASmf\config\wmcasmf.cnf.

Input Variables	Type	Description
wm.fin.doc:FIN OutboundMessage	IData	Document subscribed by this service when the **Transport** parameter in the message TPA is set to **CASmf**.

wm.casmf.trp:sendAndReceive

Service to send and receive messages from CASmf. This service is invoked by the
wm.casmf.trp:casmfSendReceiveSchedule service. Messages received from CASmf are accumulated into a
String [ ].

Output Variables	Type	Description
receivedFINMessages	String [ ]	List of messages received from CASmf.

wm.casmf.trp:CASmfOutboundTrigger

Trigger to process outbound SWIFT FIN messages to sent via CASmf. This trigger then invokes
wm.casmf.trp:processOutboundMessage to process the outbound SWIFT message.

wm.casmf.util

This folder contains utility services for retrieving property values specified in
webMethods6\IntegrationServer\packages\WmCASmf\config\wmcasmf.cnf file.

wm.casmf.util:getOutboundMessageFolder

This service retrieves the value for the *wm.casmf.send.message.folder* property specified in
webMethods6\IntegrationServer\packages\WmCASmf\config\wmcasmf.cnf file. This is the folder to
which all outbound SWIFT FIN messages to CASmf are stored prior to actually sending them to CASmf.

Output Variables	Type	Description
folder	String	Value specified for the *wm.casmf.send.message.folder* property in the wmcasmf.cnf file.

WmFIN Package

The WmFIN Package contains services used to implement and support the SWIFT FIN-compliant functionality of the webMethods SWIFT FIN Module.

wm.fin.bic

This folder contains BIC-related services. They are used in inserting and retrieving BIC information.

wm.fin.bic:getBICInfo

Retrieves BIC information from database based on specified criteria.

Input Variables	Type	Description
code	String	BIC code of the financial institution. You may specify a partial String specified using `%partial string%`.
bicKey	String	BIC+ key of the financial institution.
institution	String	Name of the financial institution.
branch	String	Name of the financial institution's branch.
city	String	City in which the financial institution is located.
modFlag	String	BIC modifier flag for the financial institution, which identifies additions, updates, and deletions of BIC records since the last update. Valid Values: ■ A. Addition. ■ U. Unchanged. ■ M. Modified.
location	String	Location of the financial institution.
countryName	String	Country in which the financial institution is located.

Output Variables	Type	Description
count	String	Specifies the number of BIC records returned in the search.
bicInfo	IData [ ]	BIC records specifying the search criteria.

wm.fin.bic:insertBICList

Imports BIC list into database.

Input Variables	Type	Description
filename	String	Fully qualified path and file name of the BIC list you want to import. For example, `c:\folder\bic.dat`.
bicInfo	String	Specify the type of BIC list you are importing. Valid values: `BIC` or `BIC+`.

Output Variables	Type	Description
errorMessage	String	Specifies the error message if one occurs.
error	String	Specifies whether an error occurred. Valid values: `yes` and `no`.

wm.fin.bic:BICInfo

Record structure identifying BIC record retrieved from the database. This specifies the result of the wm.fin.bic.getBICInfo service.

wm.fin.dfd

This folder contains services related to the loading and use of the FIN Data Field Dictionary (DFD).

wm.fin.dfd:convertBizNameFormat

Converts FIN IData from specified format to TAGONLY format and merges subfields into a FIN field.

Input Variables	Type	Description
finIData	IData	FIN IData in the format specified in the *fromFormat* input string.
msgType	String	SWIFT message type identifier. For example, `541`.
version	String	Version number of the SWIFT message record being used. For example, `nov03`.

Input Variables	Type	Description
fromFormat	String	The format of the generated blocks and fields for the input SWIFT message. Valid values:
		■ TAG_BIZNAME (default). SWIFT Message tag followed by the business name specified in the message DFD.
		■ TAGONLY. SWIFT Message tag only.
		■ BIZNAMEONLY. Business name specified in the message DFD.
		■ XMLTAG. XML-compatible tag name. This format cannot contain colons or tags that begin with a number. For example, F23G.
subfieldFlag	String	Specifies whether the fields in the input FIN IData are parsed to the subfield level. Valid values:
		■ true (default). Parse to the subfield level.
		■ false. Parse to the field level.

Output Variables	Type	Description
convertedFinIData	IData	Converted FIN IData in the format of TAGONLY.

wm.fin.dfd:convertTagFormat

Converts FIN IData from TAGONLY to the specified format and parses whole fields into subfields.

Input Variables	Type	Description
finIData	IData	FIN IData in the format specified in the *fromFormat* input string.
msgType	String	SWIFT message type identifier. For example, 541.
version	String	Version number of the SWIFT message record being used. For example, nov03.

Input Variables	Type	Description
toFormat	String	The format of the generated blocks and fields for the output FIN IData. Valid values:

■ TAG_BIZNAME (default). SWIFT Message tag followed by the business name specified in the message DFD.

■ TAGONLY. SWIFT Message tag only.

■ BIZNAMEONLY. Business name specified in the message DFD.

■ XMLTAG. XML-compatible tag name. This format cannot contain colons or tags that begin with a number. For example, F23G.

subfieldFlag	String	Specifies whether the generated fields in the output FIN IData are parsed to the subfield level. Valid values:

■ true (default). Parse to the subfield level.

■ false. Parse to the field level.

Output Variables	Type	Description
convertedFinIData	IData	Converted FIN IData in the format specified in the *fromFormat* input string.

wm.fin.dfd:getDFDList

Displays a list of DFDs loaded into the system.

Output Variables	Type	Description
dfdList	String [ ]	List of DFDs loaded into the system in <dfd name>_<dfd version> format. For example, 541_nov03.

wm.fin.dfd:loadDFD

Loads a FIN DFD into memory.

Input Variables	Type	Description
msgType	String	SWIFT message type identifier. For example, 541.
version	String	Version number of the SWIFT message record being used. For example, nov03.

wm.fin.dfd:unloadDFD

Unloads a FIN DFD from memory.

Input Variables	Type	Description
msgType	String	SWIFT message type identifier. For example, 541.
version	String	Version number of the SWIFT message record being used. For example, nov03.

wm.fin.dfd:unloadDFDs

Unloads all FIN DFDs from memory.

wm.fin.doc

This folder contains the document structures used to represent particular sections of SWIFT FIN messages, such as the header and trailer structures and their fields. Also within this folder are the generic structure definitions for incoming and outgoing SWIFT FIN messages, where the data record structure (known as Block 4 in SWIFT FIN messages) is left as an open record. These are generated by the wm.fin.dev:importFINItems service in the WmFINDev package.

wm.fin.doc:FINIData_Input

Record structure defining the fields of an incoming SWIFT message. B4 is left as open record and can be created based on the particular message type and version.

wm.fin.doc.header:FINIData_Outgoing

Record structure defining the fields of an outgoing SWIFT message. B4 is left as open record and can be created based on the particular message type and version.

wm.fin.doc.catF

The record definitions within this folder describe the record structures used to represent the body of the FIN Acknowledgement

wm.fin.doc.catF:MTF21

Record structure defining the fields of the body of the FIN Acknowledgement (F21).

wm.fin.doc.header

The record definitions within this folder describe the record structures used to represent the three header sections of a SWIFT message; the Basic Header (known as Block 1 in SWIFT FIN messages), the Application Header (Block 2, in both incoming and outgoing message format), and User Header (Block 3).

wm.fin.doc.header:ApplicationHeader_Input

Record structure defining the fields of the Application Header (Block 2) on an incoming SWIFT message.

wm.fin.doc.header:ApplicationHeader_Outgoing

Record structure defining the fields of the Application Header (Block 2) on an outgoing SWIFT message.

wm.fin.doc.header:BasicHeader

Record structure defining the fields of the Basic Header (Block 1) of a SWIFT message.

wm.fin.doc.header:UserHeader

Record structure defining the fields of the User Header (Block 3) of a SWIFT message.

wm.fin.doc.trailer

The record definitions within this folder describe the record structures used to represent the trailer section of a SWIFT message, known as Block 5.

wm.fin.doc.trailer:Trailer

Record structure defining the fields representing the Trailer section (Block 5) of a SWIFT message.

wm.fin.format

This folder contains format-related services. They are used in converting formats such as a SWIFT message format into a FIN IData.

wm.fin.format:conformFINIData

Conforms (rearranges) FIN IData into correct structure based on B4 IS Document.

Input Variables	Type	Description
inputFINIData	IData	Input *bound* FIN IData. (Must include B4 block.)
isDocument	String	Fully-qualified is document name to conform inputFINIData B4 block to. For example, `wm.fin.doc.nov03.cat5:MT502`.

Output Variables	Type	Description
outputFINIData	IData	Output *bound* conformed FIN IData.

wm.fin.format:conformIData

Conforms (rearranges) IData into correct structure.   Incorrect IData structure will not yield conformed data.

Input Variables	Type	Description
finIData	IData	Input *bound* FIN IData.
isDocument	String	Fully-qualified IS document name to conform input *finIData* to. For example, `wm.fin.doc.nov03.cat5:MT502`.

Output Variables	Type	Description
finIData	IData	Output *bound* conformed IData.

wm.fin.format:convertFINToIData

Converts a SWIFT Message into a FIN IData. This will load a 'parse' template into memory to create the correct structure.

Input Variables	Type	Description
finMsg	String	Valid SWIFT message.
msgType	String	SWIFT message type identifier. For example, `541`.
version	String	Version number of the SWIFT message record being used. For example, `nov03`.

Output Variables	Type	Description
finIData	IData	FIN IData in the format specified.

wm.fin.format:convertIDataToFIN

Converts a FIN IData into a SWIFT Message.

Input Variables	Type	Description
finIData	IData	FIN IData in the format specified.

Output Variables	Type	Description
FINmsg	IData	Output SWIFT message.

wm.fin.format:flushTemplateCache

Clears 'parse' templates from memory.

wm.fin.format:xmlToIData

Converts a XML-formatted SWIFT message into an IData.

Input Variables	Type	Description
xmlString	String	XML string.

Output Variables	Type	Description
outputIData	IData	Output FIN IData.

wm.fin.init

The services found in this folder are used to either initialize or de-initialize FIN packages on startup and shutdown of webMethods Integration Server.

wm.fin.init:startup

Initializes DSP user interface and resource bundles and configures the WmFIN package for run-time.

wm.fin.map

Services found in this folder provide easy frameworks for creating the header and trailer sections of SWIFT FIN messages, for use in outbound (to be sent to SWIFT) messages. All services have as their input the mandatory fields required in each appropriate header or trailer section.

wm.fin.maps.outbound:mapApplicationHeader

Maps the input variables into a default FIN Application Header.

Input Variables	Type	Description
userVariable	IData	FIN transport user variables.
TPA	IData	The TPA for SWIFT message.

Output Variables	Type	Description
B2	IData	Application header IData.

wm.fin.maps.outbound:mapBasicHeader

Maps the input variables into a default Basic Header.

Input Variables	Type	Description
userVariable	IData	FIN transport user variables.
TPA	IData	The TPA for SWIFT message.

Output Variables	Type	Description
B2	IData	Basic header IData.

wm.fin.maps.outbound:mapTrailer

Creates a Trailer record. When creating an outbound SWIFT message, this record does not need to be populated, so this service currently creates an empty Trailer record.

Input Variables	Type	Description
userVariable	IData	FIN transport user variables.

Output Variables	Type	Description
B5	IData	Trailer IData.

wm.fin.maps.outbound:mapUserHeader

Maps a default User Header.

Input Variables	Type	Description
userVariable	IData	FIN transport user variables.
TPA	IData	The TPA for SWIFT message.

Output Variables	Type	Description
B3	IData	User header IData.

wm.fin.mappingFunctions

The services found within this folder provide mapping functions that are used in SWIFT message migration; that is, conversion of a SWIFT message from one ISO message type to another (in this case, between ISO 7775 and ISO 15022). The functions are used in the services created by the wm.fin.migration:templateToMap service to transform field values during the mapping process.

wm.fin.mappingFunctions:AccruedInterestRemoveDays

Remove days from Accrued Interest Field.

Input Variables	Type	Description
value	String	Input string.

Output Variables	Type	Description
output	String	Output string.

wm.fin.mappingFunctions:BookValueToDealPriceIn15022

Convert from ISO 7775 Book Value field to ISO 15022 Deal Price (place //ACTU at the start of the value).

Input Variables	Type	Description
value	String	Input string.
param	String	Input variable. This is the fully-qualified name of the field within the target message where the party identifier will be placed.

Output Variables	Type	Description
output	String	Output string. This is the remainder section of the input string.

wm.fin.mappingFunctions:CrestStripPartyIdentifier

Strips party identifier from the value where the party identifier begins with '/CREST ', and returns both the party identifier and the remainder.

Input Variables	Type	Description
value	String	Input string.
param	String	Input variable. This is the fully-qualified name of the field within the target message where the party identifier will be placed.

Output Variables	Type	Description
output	String	Output string. This is the remainder section of the input string.

wm.fin.mappingFunctions:FullStripPartyIdentifier

Strips party identifier and returns both the party identifier and the remainder.

Input Variables	Type	Description
value	String	Input string.
param	String	Input variable. This is the fully-qualified name of the field within the target message where the party identifier will be placed.

Output Variables	Type	Description
output	String	Output string.

wm.fin.mappingFunctions:InsertSlashes

Insert two slashes at the beginning of a string.

Input Variables	Type	Description
value	String	Input string.

Output Variables	Type	Description
output	String	Output string. This is "//" followed by the input string.

wm.fin.mappingFunctions:QuantityOfSecurities

Converts to new Security Quantity string format (Starts with FMT, UNT, or other to FAMT//, UNIT// or AMOR// respectively.

Input Variables	Type	Description
value	String	Input string.

Output Variables	Type	Description
output	String	Output string. This is the converted *QuantityOfSecurities* value.

wm.fin.mappingFunctions:RemoveSlashes

Removes the slashes at the beginning of a string.

Input Variables	Type	Description
value	String	Input string.

Output Variables	Type	Description
output	String	Output string. This is input string without leading slashes.

wm.fin.mappingFunctions:StripPartyIdentifier

Strips party identifier and returns remainder.

Input Variables	Type	Description
value	String	Input string.

Output Variables	Type	Description
output	String	Output string. This is input string minus the party identifier (if present).

wm.fin.mappingFunctions:TaxesAdded

Returns only the Taxes Added subsection of the field.

Input Variables	Type	Description
value	String	Input string.
param	String	**Optional** - Input variable.

Output Variables	Type	Description
output	String	Output string.

wm.fin.mappingFunctions:call

Used primarily in conversion maps, this service will take the first available input, invoke the appropriate services, and return an output string.

Input Variables	Type	Description
value	IData	Input fields providing mapping information.

Output Variables	Type	Description
output	String	Results of input.

wm.fin.mappingFunctions:dateAndPlaceMapTo15022

Converts date format from YYMMDD to YYYYMMDD and maps place value.

Input Variables	Type	Description
value	String	Input string.

Output Variables	Type	Description
output	String	Output string.
TRADDET.94B:: TRAD	String	Mapped place value (migration service will map to correct IS Document format).

wm.fin.mappingFunctions:dateMapTo15022

Converts date format from YYMMDD to YYYYMMDD.

Input Variables	Type	Description
value	String	Input string.

Output Variables	Type	Description
output	String	Output string. This is the converted date.

wm.fin.mappingFunctions:dateMapTo7775

Converts date format from YYYYMMDD to YYMMDD.

Input Variables	Type	Description
value	String	Input string.

Output Variables	Type	Description
output	String	Output string. This is the converted date.

wm.fin.mappingFunctions:replaceMT

Replace the three-character string describing message type with a new message type. Input string is assumed to be the Application Header (Block 2) raw string, without '{2:' at the start.

Input Variables	Type	Description
value	String	Input string.
param	String	Input variable.

Output Variables	Type	Description
output	String	Output string.

wm.fin.mappingFunctions:strip

Strips specified leading characters from string.

Input Variables	Type	Description
value	String	Input string.
param	String	Input variable. This is the number of characters to strip from the beginning of the input string.

Output Variables	Type	Description
output	String	Output string. This is the stripped input string.

wm.fin.mappingFunctions:truncate

Truncates string to specified size.

Input Variables	Type	Description
value	String	Input string.
param	String	Input variable. This is the number of characters to truncate the input string to.

Output Variables	Type	Description
output	String	Output string. This is the truncated input string.

wm.fin.resubmit

The services found in this folder provide mapping functions that are to resubmit a failed SWIFT message on the **Repair/Resubmit** screen in the SWIFT screen of the Integration Server Administrator.

wm.fin.resubmit:editIData

Used in the **Repair/Resubmit** screen to edit SWIFT message

Input Variables	Type	Description
finPayload	String	Modified payload.

wm.fin.resubmit:getFailedMessages

Used in the **Repair/Resubmit** screen to return failed SWIFT FIN messages.

Output Variables	Type	Description
recentlyFailed	IData [ ]	Recently failed FIN documents.

wm.fin.resubmit:getRec

Used in the **Repair/Resubmit** screen to retrieve variables and documents from step pipeline

Input Variables	Type	Description
IDataEditIn	IData	Step pipeline.

Output Variables	Type	Description
params	IData	Contains the TPA for the SWIFT message.
docs	IData	Contains the FIN payload.

wm.fin.resubmit:getStepErrors

Used in the **Repair/Resubmit** screen to return step errors occurring for process.

Input Variables	Type	Description
processID	String	Process ID.

wm.fin.resubmit:getStepPipeline

Used in the **Repair/Resubmit** screen to return step errors occurring for process.

Input Variables	Type	Description
IDataEditIn	String	Step pipeline.

wm.fin.rules

The services found within this folder provide utility functions that are used in the implementation of network validation rules.

wm.fin.rules:checkCodeOrder

Specifies whether codes are in correct order.

Input Variables	Type	Description
codeList	String [ ]	Input code list.
codeOrder	String [ ]	Correct order of codes.

Output Variables	Type	Description
isCodeOrderValid	String	Specifies whether the code list is valid. Valid values: `true` or `false`.

wm.fin.rules:contains

Specifies whether key is contained in code list.

Input Variables	Type	Description
codeList	String [ ]	Input code list.
key	String	Key that may be in the code list.

Output Variables	Type	Description
keyExists	String	Specifies whether the key exists in the code list. Valid values: `true` or `false`.

wm.fin.rules:getDuplicateCodeList

Returns all codes that are duplicates in code list.

Input Variables	Type	Description
codeList	String [ ]	Input code list.

Output Variables	Type	Description
duplicateCodeList	String [ ]	All duplicate codes in the input *codeList*.

wm.fin.rules:setErrorDocument

Returns correct error document from specified variables.

Input Variables	Type	Description
key	String	Error message key. This is usually a FIN error message code.
path	String	Path of error in message. For example, `B4/SBB/57D:`.
data	String	Data where error occurs.

Output Variables	Type	Description
errors	IData [ ]	Error array with error appended to end.

wm.fin.trp

These two core services are used in conjunction with Trading Networks, to provide single-point access to send and receive SWIFT FIN messages.

wm.fin.trp:receive

Triggered by the *FINInboundMessageTrigger,* this service receives an incoming FINInboundMessage IData, parses it into a record, and sends it to Trading Networks to be recognized and routed.

Input Variables	Type	Description
FINInboundMessage	IData	IData containing raw SWIFT message.to be processed.

wm.fin.trp:send

Formats an IData into a SWIFT message and send it to the appropriate delivery service.

Input Variables	Type	Description
TPA	IData	IP TPA containing specified configuration variables.
documents	IData	IData containing FIN IData to process.
bizEnv	IData	TN business envelope containing document.

wm.fin.trp:FINInboundMessageTrigger

This Trigger subscribes to the wm.fin.doc:FINInboundMessage service. When a document is received, the wm.fin.trp:receive service is invoked.

wm.fin.utils

The services found within this folder are generic utility services providing various functionality.

wm.fin.utils:getFINMessageAndIDs

From a raw SWIFT message, this service will recognize and extract the sender, receiver and message type.

Input Variables	Type	Description
rawFINMessage	String	Raw SWIFT message.

Output Variables	Type	Description
finMessage	IData	Contains SWIFT Message and whether message is an acknowledgement.
internalSenderID	String	Trading Networks Internal Sender ID.
internalReceiverID	String	Trading Networks Internal Receiver ID.
msgType	String	SWIFT message type.

wm.fin.validation

This folder contains validation-related services. They are used to facilitate the validation of a SWIFT message.

wm.fin.validation:getErrorMessage

Return appropriate SWIFT message for that key.

Input Variables	Type	Description
key	String	Error key.

Output Variables	Type	Description
errorMessage	String	FIN error message.

wm.fin.validation:validationFinMsg

Parses and validates a SWIFT message.

Input Variables	Type	Description
bizdoc	IData	Trading Networks BizDocEnvelope containing SWIFT message.

Output Variables	Type	Description
finIData	IData	SWIFT message as an IData in TAGONLY format.
convertedFinIData	IData	SWIFT message as an IData in specified format.

wm.fin.validation:validateIData

Provides Content Validation, Network Rule Validation, Market Practice Rule Validation, and Usage Rule validation of a FIN IData.

Input Variables	Type	Description
finIData	IData	FIN IData.
userVariables	IData	User variables providing configuration information for message.

Output Variables	Type	Description
isValid	String	Specifies whether FIN IData passes validation according to the message configuration.
errorArray	IData	Errors occurring if FIN IData doesn't pass validation.

wm.fin.validation:validateIDataUtil

Validates content and structure of a FIN IData.

Input Variables	Type	Description
finIData	IData	FIN IData.
userVariables	IData	User variables providing configuration information for message.
msgType	String	SWIFT message type identifier. For example, 541.
version	String	Version number of the SWIFT message record being used. For example, nov03.

Input Variables	Type	Description
isDocument	String	Fully-qualified name of the TN document type to validate FIN IData structure. For example, `wm.fin.doc.nov03.cat5:MT502`.
fromFormat	String	The format of the generated blocks and fields for the input SWIFT message. Valid values:
		■ `TAG_BIZNAME` (default). SWIFT Message tag followed by the business name specified in the message DFD.
		■ `TAGONLY`. SWIFT Message tag only.
		■ `BIZNAMEONLY`. Business name specified in the message DFD.
		■ `XMLTAG`. XML-compatible tag name. This format cannot contain colons or tags that begin with a number. For example, `F23G`.
mapServiceName	String	The migration map's fully-qualified service name. For example, `wm.fin.migration.maps:map520To540`.
validateHeaders	String	Specifies whether service should validate headers. Valid values: `true` or `false`.

Output Variables	Type	Description
isValid	String	Specifies whether FIN IData passes validation according to the message configuration.
errors	IData	Errors occurring if FIN IData does not pass validation.

WmFINDev Package

The WmFINDev Package contains the records and services that enable users to create message records, TPAs, TN document types, validation rules, and Processing Rules, as well as migrate SWIFT FIN messages.

wm.fin.dev

This folder contains design-time services. They are used in the install and configuration of new SWIFT FIN messages.

wm.fin.dev:importFINItems

Imports, configures and creates all items needed in a SWIFT Message transaction. This includes the IS document, DFD, parse template, TN doc type, TN Processing Rule, and TN TPA.

Input Variables	Type	Description
msgType	String	SWIFT message type. For example, `502`.
version	String	FIN version. For example, `nov03`.
format	String	The format of the generated blocks and fields for the input SWIFT message. Valid values: ■ `TAG_BIZNAME` (default). SWIFT Message tag followed by the business name specified in the message DFD. For example, `23G_Function of the Message`. ■ `TAGONLY`. SWIFT Message tag only. For example, `23G:`. ■ `BIZNAMEONLY`. Business name specified in the message DFD. For example, `Function of the Message`. ■ `XMLTAG`. XML-compatible tag name. This format cannot contain colons or tags that begin with a number. For example, `F23G`.
subfieldFlag	String	Specifies whether the fields in the IS Document Type generated for this FIN Message are parsed to the subfield level. Valid values: ■ `true` (default). Parse to the subfield level. ■ `false`. Parse to the field level.

Input Variables	Type	Description
createDocType	String	Indicates whether to create and insert a TN document type for this message. The TN document type is used to recognize an incoming message. Valid values: `true` or `false`.
createProcessing Rule	String	Indicates whether to create a Trading Networks Processing Rule for this message. After the message is recognized, the Processing Rule specifies how the message should be processed.Valid values: `true` or `false`.
createTPA	String	Indicates whether to create a Trading Networks TPA for this message. This specifies specific variables used in WmFIN for processing and validation.Valid values: `true` or `false`.

wm.fin.migration

The services in this folder are used in the migration of SWIFT FIN messages from ISO 7775 to ISO 15022 and ISO 15022 to ISO 7775.

wm.fin.migration:mapIDataToMap

This service is for internal use only.

wm.fin.migration:templateToMap

Creates a webMethods Flow map from a template that can be used in the migration of ISO 7775 messages to ISO 15022 and ISO 15022 to ISO 7775.

> **Note:** IS Documents for the input/output MUST be created first.

Input Variables	Type	Description
templateFileName	String	Path of migration template. For example, `packages/WmFINDev/import/templateName`.
package	String	IS package in which to generate the migration map. For example, `Default`.
mapServiceName	String	Fully qualified IS service name of the migration map. For example, `wm.fin.migration.maps: map520To540`.
inputMsgType	String	SWIFT message type of the input SWIFT message. For example, `502`.

Input Variables	Type	Description
inputVersion	String	FIN version of the input SWIFT message. For example, `nov03`.
inputFormat	String	The format of the generated blocks and fields for the input SWIFT message. Valid values: ■ `TAG_BIZNAME` (default). SWIFT Message tag followed by the business name specified in the message DFD. ■ `TAGONLY`. SWIFT Message tag only. ■ `BIZNAMEONLY`. Business name specified in the message DFD. ■ `XMLTAG`. XML-compatible tag name. This format cannot contain colons or tags that begin with a number. For example, `F23G`.
outputMsgType	String	SWIFT message type of the output SWIFT message. For example, `502`.
outputVersion	String	SWIFT version of the output SWIFT message. For example, `nov03`.
outputFormat	String	For the output SWIFT message. The format of the generated blocks and fields. Valid values: ■ `TAG_BIZNAME` (default). SWIFT Message tag followed by the business name specified in the message DFD. ■ `TAGONLY`. SWIFT Message tag only. ■ `BIZNAMEONLY`. Business name specified in the message DFD. ■ `XMLTAG`. XML-compatible tag name. This format cannot contain colons or tags that begin with a number. For example, `F23G`.

wm.fin.migration:templateToMapIData

This service is for internal use only.

WmFINMarketPractice Package

The WmFINMarketPractice Package contains 16 common services that support Market Practices for some Category 5 messages. The services in this package are for internal use only.

WmFINTransport Package

This package contains the services needed to exchange messages with SWIFT using Automated File Transfer (AFT) and MQSeries.

wm.fin.doc

This folder contains publishable IS document types that are used to send and receive SWIFT FIN messages. It also contains IS document types which will be used to populate values for a given TPA.

Document	Publishable	Description
FINInboundMessage	Yes	SWIFT FIN messages received via AFT or MQ Series are mapped into this document. This document is then published to the Broker or the Integration Server where it is processed by the wm.fin.trp:FINInboundMessageTrigger service and the wm.fin.trp:receive service.
FINOutboundMessage	Yes	SWIFT FIN messages to be sent via AFT or MQSeries are mapped into this document and published to the Broker or the Integration Server. If `wm.fin.doc:FINOutboundMessage/Transport = MQ`, this document is subscribed and processed by the wm.fin.transport.MQSeries:MQSeriesPutTrigger. If `wm.fin.doc:FINOutboundMessage/Transport = AFT`, this document is subscribed and processed by the wm.fin.transport.AFT:AFTOutboundTrigger service.
MessageHeader	No	Data used to populate to header blocks (B1,B2,B3 and B5) in the outgoing SWIFT message.
UserParameters	No	TPA information to be used while sending and receiving SWIFT FIN messages.

wm.fin.transport.AFT

This folder contains services to send and receive messages using Automated File Transfer.

wm.fin.transport.AFT:generateUniqueFileName

This service generates a unique file name.

Input Variables	Type	Description
folder	String	The folder in which the file needs to be created.
extension	String	**Optional** - The extension to use for the generated file name. If no extension has been specified, the default extension is ".inp".

Output Variables	Type	Description
fileName	String	Generated unique file name.

wm.fin.transport.AFT:processInboundFile

Flat File Listener invokes this service to process incoming SWIFT FIN messages received via AFT.

Input Variables	Type	Description
ffdata	java.io.InputStream	InputStream to the file received via AFT.

wm.fin.transport.AFT:processIncomingMessage

This service parses incoming SWIFT FIN messages separated with special characters and outputs the SWIFT FIN messages as a String array with the special characters stripped.

Input Variables	Type	Description
ffdata	java.io.InputStream	InputStream to the file received via AFT.

Output Variables	Type	Description
finMessage	String [ ]	A string list containing the individual SWIFT FIN messages with the special characters stripped.

wm.fin.transport.AFT:processOutboundFile

Generates a unique file name and writes the outbound SWIFT message to the file to the folder specified in the TPA.

Input Variables	Type	Description
ffdata	java.io.InputStream	InputStream to the file received via AFT.
wm.fin.doc:FIN OutboundMessage	IData	Document subscribed by this service when the **Transport** parameter in the message TPA is set to AFT.

wm.fin.transport.AFT:AFTOutboundTrigger

Trigger to subscribe for outbound SWIFT FIN messages when the **Transport** parameter in the message TPA is set to "AFT". The trigger then invokes the wm.fin.transport.AFT:processOutboundFile service to process the outbound SWIFT message.

wm.fin.transport.MQ

This folder contains services to send and receive SWIFT FIN messages from MQ Series.

wm.fin.transport.MQSeries:getListenerService

This is the service to be specified by the user when getting SWIFT FIN messages from a specified MQSeries queue. This service strips out extraneous information in the SWIFT message and publishes the actual SWIFT message to be processed further by services in WmFIN package. More specifically, wm.fin.trp:receive service subscribes to, processes, and validates the message, after which the service will either pass the resulting TN document type to the process run time or to the specified Trading Networks Processing Rule. The user must specify this service as the **Message Service** when creating the **WebSphere MQ-to-IS** message handler.

Input Variables	Type	Description
msgbody	String	SWIFT message retrieved off the specified MQ Series queue.

wm.fin.transport.MQSeries:put

This service puts the outbound SWIFT message in a MQ Series queue by invoking the 'put' message handler service created by the user and specified in the message TPA.

Input Variables	Type	Description
wm.fin.doc:FIN OutboundMessage	IData	Document subscribed to by this service when the **Transport** parameter in the message TPA is set to "MQ".

wm.fin.transport.MQSeries:MQSeriesPutTrigger

Trigger to subscribe to outbound SWIFT FIN messages when the **Transport** parameter in the message TPA is set to "MQ". The trigger then invokes the wm.fin.transport.MQSeries:put service to put the outbound SWIFT message into the specified MQ Series queue.

wm.fin.transport.Test

This folder contains services, triggers and publishable documents to be used with WmFINSamples package.

wm.fin.transport.Test:processFinMsg

Service to receive an outbound SWIFT message and simulate a round-trip by publishing the same message as an inbound SWIFT message. This service is invoked by the wm.fin.transport.Test:FINSampleOutboundMessageTrigger service when the **Transport** parameter in the message TPA is set to "Test".

Input Variables	Type	Description
wm.fin.doc:FIN OutboundMessage	IData	Document subscribed to by this service when the **Transport** parameter in the message TPA is set to "Test".

wm.fin.transport.Test:FINSampleInboundMessageTrigger

Trigger to subscribe to the `FINSampleInboundMessage` document published by the wm.fin.transport.Test:processFinMsg service. This trigger then invokes the wm.fin.sample:receive service to process the incoming SWIFT message.

wm.fin.transport.Test:FINSampleOutboundMessageTrigger

Trigger to subscribe to outbound SWIFT FIN messages when the **Transport** parameter in the message TPA is set to "Test". The trigger then invokes the wm.fin.transport.Test:processFinMsg service to process the outbound SWIFT message.

wm.fin.transport.Test:FINSampleInboundMessage

Publishable IS Document representing an inbound SWIFT message to be used WmFINSamples package.

wm.fin.transport.property

This folder contains services to retrieve properties defined for publishing SWIFT FIN messages.

wm.fin.transport.property:getProperty

Outputs the property value specified in *webMethods6\IntegrationServer*\packages\ WmFINTransport\config\finTransport.cnf file.

Input Variables	Type	Description
propertyName	String	Property name specified in finTransport.cnf file

Output Variables	Type	Description
value	String	Value of the property specified in finTransport.cnf file

wm.fin.transport.property:listProperties

Outputs all the properties specified in *webMethods6\IntegrationServer*\packages\ WmFINTransport\config\finTransport.cnf file.

Output Variables	Type	Description
properties	Document	List of all the properties and their values specified in finTransport.cnf file

WmIPCore Package

The WmIPCore Package contains generic services for using the webMethods SWIFT FIN Module with the webMethods Integration Server.

wm.ip.bizdoc

The services in this folder are used to deal with Trading Networks bizdoc envelope.

wm.ip.bizdoc:addErrorContentPart

Add/update errors content part of the bizdoc and set the User Status of the bizdoc

Input Variables	Type	Description
errors	Document [ ]	Error to be added as content part to the bizdoc.
bizdoc	Document	Bizdoc envelope to which errors must be added as content part.
errorType	String	User status to be set for the bizdoc.

Output Variables	Type	Description
bizdoc	Document	Updated bizdoc envelope.

wm.ip.bizdoc:decodeErrorContentPart

Decodes a byte [ ] into an document [ ] object.

Input Variables	Type	Description
bytes	byte [ ]	Bytes to be decoded into a document [ ] object.

Output Variables	Type	Description
errors	Document [ ]	Errors returned.

wm.ip.bizdoc:getBizDocFromEvent

Retrieve latest bizdoc envelope submitted to PRT (process run time).

Input Variables	Type	Description
variables	Document	An IData object that contains the variables related to the current business process.
document	Document	An IData object that contains the document related to the current business process.
lastEvents	String	**Optional** - The events associated with the BizDocEnvelopes that you want to retrieve.
getContent	String	Whether you want to retrieve the content. valid values: `true` or `false`.

Output Variables	Type	Description
bizenv	Document	Bizdoc envelope retrieved from TN database.

wm.ip.cm

This folder contains services used to manage processes, such as start a process, unhandled documents, etc.

wm.ip.cm.handlers:defaultHandler

This service is a place holder for the service that is invoked by a handler step in a process model. You should replace this service with a handler service that you code, and update the process model to reflect your service.

Input Variables	Type	Description
ProcessData	Document	An IData object that contains run-time process information for the process.

wm.ip.cm.handlers:done

Sets the current business process to a done (completed) state.

Input Variables	Type	Description
ProcessData	Document	An IData object that contains run-time process information for the process.

wm.ip.cm.handlers:done

Sets the current business process to an error state.

Input Variables	Type	Description
ProcessData	Document	An IData object that contains run-time process information.

wm.ip.cm:getConversationID

Construct a Conversation ID in the format <OriginalReceiverID>-<UniqueIdentifier>.

Input Variables	Type	Description
bizdoc	Document	Bizdoc envelope associated with the current business process.
cid	String	Unique identifier to be used as part of the generated Conversation ID.

Output Variables	Type	Description
convID	String	Generated Conversation ID.

wm.ip.cm:getConversationScript

Finds the first matching process run-time script for a given bizdoc (a BizDocEnvelope). Generates the Conversation ID for the process run-time script that is going to be initiated.

Input Variables	Type	Description
bizdoc	Document	Bizdoc envelope used to find matching process run-time.
finID	String	A unique identifier to be used as part of the Conversation ID.
direction	String	Whether the message is inbound or outbound. Valid values: Inbound or Outbound.

Output Variables	Type	Description
name	String	Specifies the first matching process run-time script for a given bizdoc.

wm.ip.cm:processDocument

This service performs the following tasks:

◼ Recognizes the incoming document.

◼ Finds the matching process run-time script if the value you specify for cid is null.

◼ Saves the incoming document to the Trading Networks database.

◼ Submits the recognized bizdoc to PRT.

Input Variables	Type	Description
node	Object	**Optional** - XML node to be recognized as a bizdoc.
input	Document	**Optional** - IData object to be recognized as a bizdoc to be eventually used to start a new business process or rejoin an existing business process. Either node or input must be present in the pipeline.
nsDecls	Document	**Optional** - Namespace prefixes to use for the conversion.
cid	String	**Optional** -The Conversation ID of the business process that the document, which is specified in input, is joining, if one exists. If the document is not currently part of a business process, leave *cid* null.

wm.ip.cm:startConversation

Starts a process model and passes it a matching process run-time script and assigns the business process the specified Conversation ID.

Input Variables	Type	Description
matchResult	Object	The process run-time script that matches the given bizdoc.
cid	String	The Conversation ID to assign to the business process you are starting.

wm.ip.cm:waitStepInit

Any step in the process that waits for a document can invoke this service. This service:

■ Retrieves the BizDocEnvelope that is associated with the current document event.

■ Retrieves the TPA that is associated with the business process if it does not already exist.

Input Variables	Type	Description
ProcessData	Document	The data associated with the current business process, such as the process instance ID, roles, etc.

Input Variables	Type	Description
documents	Document	An IData object that contains the documents associated with the business process.
variables	Document	An IData object that contains the variables associated with the business process.
roles	Document	The role information for the sender and receiver of the document.
bizEnv	Document	The BizDoc envelope that is associated with the most recent document event.
TPA	Document	An IData object that contains the trading partner profile information of the partners involved in the business process and process specific variables.

wm.ip.profile

This folder contains services that are used to retrieve information about trading partners and the retrieves the internal identifier of the sender and receiver for a business process.

wm.ip.profile:createCertChainList

Creates a certificate chain list with the user certificate as the first certificate in the list followed with the CA certificate chain.

Input Variables	Type	Description
userCertBytes	Byte [ ]	User certified bytes.
certChainBytes	Byte [ ]	CA certificate chain bytes.

Output Variables	Type	Description
certChain	Object [ ]	User certified and CA certificate chain bytes.

wm.ip.profile:getInternalIDs

Retrieves the internal identifier of the sender and receiver for a business process.

Input Variables	Type	Description
variable	Document	An IData object that contains the variables related to current business process.
focalRole	String	The focal role associated with the current business process.

Output Variables	Type	Description
senderID	String	The Trading Networks internal identifier of the sender.
receiverID	String	The Trading Networks internal identifier of the receiver.
tpaSenderID	String	The Trading Networks internal ID of the partner that is associated with the focal role of the business process.
tpaReceiverID	String	The Trading Networks internal ID of the partner that is associated with the second role of the business process.
secondRole	String	The second role associated with the current business process.

wm.ip.profile:getTPA

This service retrieves the TPA for a given sender, receiver and agreement ID combination.

Input Variables	Type	Description
docTypeName	String	Agreement ID of the TPA.
tpaSenderID	String	The Trading Networks internal identifier of the sender.
tpaReceiverID	String	The Trading Networks internal identifier of the receiver.

Output Variables	Type	Description
tpaData	Document	IData object containing the retrieved TPA data.

wm.ip.profile:getTPAInfo

Retrieves an IData object that represents the trading partner profile and process information related to the current business process.

Input Variables	Type	Description
senderID	String	The Trading Networks internal identifier of the sender.
receiverID	String	The Trading Networks internal identifier of the receiver.
docTypeName	String	Name of the TN document type that was received.
tpaSenderID	String	The Trading Networks internal ID of the partner that is associated with the focal role of the business process.
tpaReceiverID	String	The Trading Networks internal ID of the partner that is associated with the second role of the business process.

Output Variables	Type	Description
TPA	Document	IData object containing sender, receiver profile information and process information related to the current business process.

wm.ip.rec

This folder contains the IS document types used when retrieving TPA information.

IS Document Type	Description
Address	Address information for a trading partner.
Contact	Contact information for trading partner.
Corporation	Company entity information for trading partner.
Delivery	Delivery method information for trading partner.
ExternalID	An external ID for trading partner.

IS Document Type	Description
TPAInfo	Trading partner profile information and process information.
TPInfo	Trading partner information for a trading partner.

wm.ip.ui

The services in this folder add submenus' to the specified link under Adapters in the Server Admin UI.

wm.ip.ui:addSubmenu

Add the specified submenu under the specified link in Adapters section of the Server Admin UI.

Input Variables	Type	Description
name	String	The name of the adapter link in Adapters section of the Server Admin UI to which the submenus need to be specified.
tabs	Document [ ]	Information containing the submenus to be added

wm.ip.ui:removeSubmenu

Remove the specified submenu under the specified link in Adapters section of the Server Admin UI.

Input Variables	Type	Description
name	String	The name of the adapter link in Adapters section of the Server Admin UI to remove.
tabs	Document [ ]	Information containing the submenus to be removed.

wm.ip.util

This folder contains utility services used by other packages such as generating unique ID, writing to a file, etc.

wm.ip.util:createFinID

Generates a unique 16 character identifier

Output Variables	Type	Description
finID	String	Generated unique 16 character identifier

wm.ip.util:formatErrorMessage

Formatted string representation of error IData object.

Input Variables	Type	Description
errorMessage	Document	IData object containing the error information.
tabs	Document [ ]	Information containing the submenus to be removed.

Output Variables	Type	Description
formattedError	String	Formatted string representation of error IData object.

wm.ip.util:getLastDocuments

Retrieves the bizdoc envelope(s) specified in the lastEvents variable from TN database.

Input Variables	Type	Description
variables	Document	IData object containing the variables related to the current business process.
documents	Document	IData object containing the documents related to the current business process.
getContent	String	**Optional** - Whether you want to retrieve the content for the bizdoc. Valid values: `true` or `false`.
lastEvents	String [ ]	IDs of the bizdoc envelopes to be retrieved.

Output Variables	Type	Description
recordCount	String	Number of bizdoc envelopes retrieved from TN database.
events	Document [ ]	IData[ ] containing the retrieved bizdoc envelope.

wm.ip.util:init

Startup service for WmIPCore package to perform needed initialization routines.

wm.ip.util:invokeService

Invoke the specified service synchronously or asynchronously.

Input Variables	Type	Description
serviceName	String	Service to be invoked.
threadInvoke	String	**Optional** - Whether service is invoked asynchronously.
		`true`. Invoked asynchronously.
		`false`. Invoked synchronously in the same thread.
input	Document	IData object containing data to be passed to the service being invoked.

Output Variables	Type	Description
events	Document	IData object containing the output from the service invoked synchronously.

wm.ip.util:removeEmptyStrings

Recursively remove any string variables that either have a null value or zero length in the given IData object.

Input Variables	Type	Description
Record	Document	IData object from which empty string variables need to be removed.

Output Variables	Type	Description
Record	Document	IData object from which empty string variables have been removed

wm.ip.util:writeLog

Write a message to Integration Server Log.

Input Variables	Type	Description
message	String	The detailed message to write to the Integration Server Log.
briefMessage	String	**Optional** - Brief message to write to the Integration Server Log.

Input Variables	Type	Description
relatedConversation Id	String	**Optional** - The business process to which the log message relates.
TNLevel	String	**Optional** - Type of activity log entry to create. Valid values: OFF, MESSAGE, WARNING, or ERROR.

wm.ip.util:writeToFile

Writes the specified string or the specified bytes to the local file system.

Input Variables	Type	Description
FileName	String	Name of the file to which to write the specified the string or bytes. Either specify a file name relative to the Integration Server home folder or specify a fully-qualified path name.
cursdata	String	**Optional** - String (in UTF-8 encoding) to write to the file system. You must specify either userdata or bytesIn.
bytesIn	Object	**Optional** - Bytes to be write to the file system. You must specify either userdata or bytesIn.

Output Variables	Type	Description
errorStatus	String	**Optional** - Error message when writing to the local file system.

XML Parsing Templates

Overview

> **Important!** XML parsing templates are used only when receiving messages in Integration Server.

webMethods provides XML parsing template files to define the structure of SWIFT FIN messages. Each parsing template describes the message using an XML syntax, and each parsing template defines a unique SWIFT message. The webMethods SWIFT FIN Module uses the parsing template when it receives a message of that type.

To fully define the entire set of SWIFT FIN messages, a parsing template is required for each type of SWIFT message. The parsing templates are installed in the appropriate category and version folder. The webMethods SWIFT FIN Module reads a parsing template as needed at run time.

The name of each parsing template is based on the definition of the message type it contains. Each message type has a unique ID, which is usually a three-digit number. Typically, the name of a parsing template follows a convention that indicates the MT defined in the parsing template. The following table shows the format used for the names of parsing templates:

Format of Parsing Template Name	Used for…
`swiftmtF21.xml`	Incoming ACK/NACK messages returned by the SWIFT system. Any service message will follow this file name format.
`swiftmtnnn.xml` where *nnn* is the unique id for the message type.	All other incoming messages types and all outgoing message. The webMethods SWIFT FIN Module looks for a specific parsing template file for the specific type of message. For example, `swiftmt101.xml`.

The parsing templates that webMethods provides are a mixture of messages that conform to the older SWIFT message standard (ISO 1775) and the new standard (ISO 15022). The webMethods SWIFT FIN Module can support both standards because of the flexible parsing template syntax. As a result, when SWIFT issues an update of their message standards, you can define new parsing templates for new SWIFT message formats or update existing parsing templates for updated SWIFT message formats.

The wm.fin.trp:receive service converts messages that are received from SWIFT into Integration Server document structures. Likewise, the wm.fin.trp:send service converts Integration Server documents into SWIFT FIN messages, so that the messages can be sent to SWIFT.

SWIFT Message Data

The following message is a sample of *MT 101*. Users unfamiliar with the SWIFT format should take time to study this data. Alternate blocks and repeating sequences within block 4 have been highlighted for clarity.

```
{1:F01PASOBEB0AXXX0071007172}{2:O1011509010306LRLRXXXX4A00000009622301
03061609N}
{3:{108:MT101 005 OF 007}}{4:
:20:00054
:50H:/12345-67891
WALT DISNEY COMPANY
:30:000228
:
:21:DP951101TRSGB
:32B:USD132546,93
:50L:WALT DISNEY PRODUCTION HOLLYWOOD CA
:57A:TESTGBVT
:59:/0010499
TRISTAN RECORDING STUDIOS
35 SURREY ROAD
BROMLEY, KENT
:71A:OUR
:21:WDC951101RPCUS
:32B:USD377250,
:50L:WALT DISNEY COMPANY LOS ANGELES, CA
:57A:TESTUSVT
:59:/26351-38947
RIVERS PAPER COMPANY
37498 STONE ROAD
SAM RAMON, CA
:71A:OUR
-}{5:{MAC:711DDD87}{CHK:A66AB15C6E3F}{TNG:}}{S:{SAC:}{COP:P}}
```

Sample SWIFT Message Definition

The following tables give the definition of block 4 in SWIFT message 101. Blocks 1, 2, 3 and 5 are not shown because they have a fixed definition for all message types.

Block 4, Mandatory Sequence A

Field Name	Mandatory
21R	No
50L	No
50H	No

Field Name	Mandatory
52A *or* 52C	No
51A	No
30	Yes
25	No

Block 4, Mandatory Repetitive Sequence B

Field Name	Mandatory	Notes
21	Yes	
21F	No	
23E	Yes	Field can repeat multiple times.
32B	Yes	
50L	No	
50H	No	
52A *or* 52C	No	
56A *or* 56C *or* 56D	Yes	
57A *or* 57C *or* 57D	No	
59	Yes	
70	No	
77B	No	
33B	No	
71A	Yes	
25A	No	
36	No	

Parsing Template Structure

All SWIFT FIN messages are essentially a sequence of fields that are contained within blocks. The message syntax allows for the fact that blocks and fields can be optional, and that blocks can be nested to any level, can repeat, or embed submessages. Every SWIFT message consists of one to five blocks as shown in the following table.

Block ID	Block Name	Mandatory	Description
1	Basic Header	Yes	Contains fixed length, untagged fields.
2	Application Header	No	Contains fixed length, untagged fields.
3	User Header	No	Contains tagged, delimited fields that are mapped to individual fields.
4	Text	No	Contains tagged, delimited fields that are mapped to individual fields within a sub structure. This block can contain nested blocks of fields that are mapped into further sub-structures. It also can contain repeating sequences of fields, which are mapped to a sequence of structures.
5	Trailers	No	Contains delimited, tagged fields that are mapped to individual fields within a sub structure.

Each of these five basic blocks is enclosed in braces {….} and is identified by a single digit. Although defined as an optional block, in practice block 4 is always present because it contains the actual message text.

Despite these complexities, the parsing templates use just two main elements: *block* and *lineAttribute*.

Sample Parsing Template

A sample parsing template is illustrated below:

Sample Parsing Template

```
<?xml version="1.0"?>
<block id="101" isMandatory="true" isList="false">
    <lineAttribute id="1:"   isMandatory="true"  extractHint="BR,{,},T,S"
idHint="FL,0,2" B2BMap="" EAImap="B1" />
    <lineAttribute id="2:"   isMandatory="false"  extractHint="BR,{,},T,S"
idHint="FL,0,2" B2BMap="" EAImap="B2" />
```

```
    <block id="3:"   isMandatory="true" isList="false"  termString=""
extractHint="BR,{,},T,S" idHint="FL,0,2" EAImap="B3">
        <lineAttribute id="103:"   isMandatory="false"
extractHint="BR,{,},T,S" idHint="FL,0,4" B2BMap="" EAImap="O103" />
        <lineAttribute id="113:"   isMandatory="false"
extractHint="BR,{,},T,S" idHint="FL,0,4" B2BMap="" EAImap="O113" />
        <lineAttribute id="108:"   isMandatory="false"
extractHint="BR,{,},T,S" idHint="FL,0,4" B2BMap="" EAImap="O108" />
        <lineAttribute id="119:"   isMandatory="false"
extractHint="BR,{,},T,S" idHint="FL,0,4" B2BMap="" EAImap="O119" />
        <lineAttribute id="115:"   isMandatory="false"
extractHint="BR,{,},T,S" idHint="FL,0,4" B2BMap="" EAImap="O115" />
    </block>
    <block id="4:\r\n"   isMandatory="false" isList="false"
termString="\r\n" extractHint="BR,{,-},T,S" idHint="FL,0,4" EAImap="B4">
        <lineAttribute id=":20:" isMandatory="true"
extractHint="DL,:,T,S" idHint="FL,0,4" B2BMap="" EAImap="M20" />
        <lineAttribute id=":21R:" isMandatory="false"
extractHint="DL,:,T,S" idHint="FL,0,5" B2BMap="" EAImap="O21R" />
        <lineAttribute id=":50L:" isMandatory="false"
extractHint="DL,:,T,S" idHint="FL,0,5" B2BMap="" EAImap="O50L" />
        <lineAttribute id=":50H:" isMandatory="false"
extractHint="DL,:,T,S" idHint="FL,0,5" B2BMap="" EAImap="O50H" />
        <lineAttribute id=":52A:,:52C:" isMandatory="false"
extractHint="DL,:,T,S" idHint="FL,0,5" B2BMap="" EAImap="O52A,O52C" />
        <lineAttribute id=":51A:" isMandatory="false"
extractHint="DL,:,T,S" idHint="FL,0,5" B2BMap="" EAImap="O51A" />
        <lineAttribute id=":30:" isMandatory="true"
extractHint="DL,:,T,S" idHint="FL,0,4" B2BMap="" EAImap="M30" />
        <lineAttribute id=":25:" isMandatory="false"
extractHint="DL,:,T,S" idHint="FL,0,4" B2BMap="" EAImap="O25" />
        <block id="B4B" isMandatory="true" isList="true" termString="\r\n"
EAImap="B4B">
            <lineAttribute id=":21:" isMandatory="true"
extractHint="DL,:,T,S" idHint="FL,0,4" B2BMap="" EAImap="M21" />
            <lineAttribute id=":21F:" isMandatory="false"
extractHint="DL,:,T,S" idHint="FL,0,5" B2BMap="" EAImap="O21F" />
            <block id="B423E" isMandatory="false" isList="true"
termString="\r\n" EAImap="B423E">
                <lineAttribute id=":23E:" isMandatory="true"
extractHint="DL,:,T,S" idHint="FL,0,5" B2BMap="" EAImap="O23E" />
            </block>
            <lineAttribute id=":32B:" isMandatory="true"
extractHint="DL,:,T,S" idHint="FL,0,5" B2BMap="" EAImap="M32B" />
            <lineAttribute id=":50L:" isMandatory="false"
extractHint="DL,:,T,S" idHint="FL,0,5" B2BMap="" EAImap="O50L" />
            <lineAttribute id=":50H:" isMandatory="false"
extractHint="DL,:,T,S" idHint="FL,0,5" B2BMap="" EAImap="O50H" />
            <lineAttribute id=":52A:,:52C:" isMandatory="false"
extractHint="DL,:,T,S" idHint="FL,0,5" B2BMap="" EAImap="O52A,O52C" />
```

```
            <lineAttribute id=":56A:,:56C:,:56D:" isMandatory="false"
extractHint="DL,:,T,S" idHint="FL,0,5" B2BMap="" EAImap="O56A,O56C,O56D"
/>
            <lineAttribute id=":57A:,:57C:,:57D:" isMandatory="false"
extractHint="DL,:,T,S" idHint="FL,0,5" B2BMap="" EAImap="O57A,O57C,O57D"
/>
            <lineAttribute id=":59:" isMandatory="true"
extractHint="DL,:,T,S" idHint="FL,0,4" B2BMap="" EAImap="M59" />
            <lineAttribute id=":70:" isMandatory="false"
extractHint="DL,:,T,S" idHint="FL,0,4" B2BMap="" EAImap="O70" />
            <lineAttribute id=":77B:" isMandatory="false"
extractHint="DL,:,T,S" idHint="FL,0,5" B2BMap="" EAImap="O77B" />
            <lineAttribute id=":33B:" isMandatory="false"
extractHint="DL,:,T,S" idHint="FL,0,5" B2BMap="" EAImap="O33B" />
            <lineAttribute id=":71A:" isMandatory="true"
extractHint="DL,:,T,S" idHint="FL,0,5" B2BMap="" EAImap="M71A" />
            <lineAttribute id=":25A:" isMandatory="false"
extractHint="DL,:,T,S" idHint="FL,0,5" B2BMap="" EAImap="O25A" />
            <lineAttribute id=":36:" isMandatory="false"
extractHint="DL,:,T,S" idHint="FL,0,4" B2BMap="" EAImap="O36" />
        </block>
    </block>
    <block id="5:"  isMandatory="false" isList="false"  termString="\r\n"
extractHint="BR,{,},T,S" idHint="FL,0,2" EAImap="B5">
        <lineAttribute id="MAC"   isMandatory="true"
extractHint="BR,{,},T,S" idHint="FL,0,3" B2BMap="" EAImap="MMAC" />
        <lineAttribute id="CHK"   isMandatory="true"
extractHint="BR,{,},T,S" idHint="FL,0,3" B2BMap="" EAImap="MCHK" />
        <lineAttribute id="TNG"   isMandatory="false"
extractHint="BR,{,},T,S" idHint="FL,0,3" B2BMap="" EAImap="OTNG" />
        <lineAttribute id="PDE"   isMandatory="false"
extractHint="BR,{,},T,S" idHint="FL,0,3" B2BMap="" EAImap="OPDE" />
        <lineAttribute id="SYS"   isMandatory="false"
extractHint="BR,{,},T,S" idHint="FL,0,3" B2BMap="" EAImap="OSYS" />
        <lineAttribute id="PDM"   isMandatory="false"
extractHint="BR,{,},T,S" idHint="FL,0,3" B2BMap="" EAImap="OPDM" />
        <lineAttribute id="DLM"   isMandatory="false"
extractHint="BR,{,},T,S" idHint="FL,0,3" B2BMap="" EAImap="ODLM" />
        <lineAttribute id="PAC"   isMandatory="false"
extractHint="BR,{,},T,S" idHint="FL,0,3" B2BMap="" EAImap="OPAC" />
        <lineAttribute id="MRF"   isMandatory="false"
extractHint="BR,{,},T,S" idHint="FL,0,3" B2BMap="" EAImap="OMRF" />
    </block>
</block>
```

Block Syntax of a Parsing Template

The `block` elements in the SWIFT parsing template define the blocks in the SWIFT message. The syntax of a `block` directive is shown below. (Note that optional parameters are shown in square braces [ ].)

```
<block id="id" isMandatory="true|false" isList="true|false"
[termString="string"] [ EAImap="string"] [[extractHint="hint"
idHint="hint"] [ loadBlockHint="hint" blockPointer="pointer"]]>
```

For the first (outer most) block directive, only the mandatory parameters are supplied. The first block directive identifies the entire message, which translates to the top level Enterprise document type. The syntax of the first block directive always takes the following syntax:

```
<block id="nnn" isMandatory="true" isList="false">
```

where *nnn* is the SWIFT message type number. This number must match the number in the parsing template file name, for example, for *MT 101*, the value of *nnn* must be 101 in the file `swiftmt101.xml`.

For all subsequent blocks, the `block` directive requires optional parameters. The full set of parameters are described below.

Elements	Values	Description
id	block id	The block number in the SWIFT message, including delimiting characters. For the first block directive this contains the message type id.
isMandatory	true | false	**false** indicates that the block is optional and does not need to be present for the message to be considered valid. **true** indicates that the block must be present in an incoming message or the message will fail validation.
isList	true | false	**false** indicates that only one instance of the block can occur. **true** indicates that the block can repeat one or more times.
termString	\r (carriage return) \n (line feed)	Characters that occur at the end of the block. Some blocks are terminated with a carriage return + line feed, while other blocks. For example blocks 1, 2 and 3, are not terminated with a carriage return + line feed and the value of termString should be an empty string (termString="").
EAImap	document structure name	Represents the name of the structure in the document that the SWIFT message block is mapped to or from. By convention, the EAImap value for blocks is prefixed with 'B'.

Elements	Values	Description
extractHint	Parameter list	he first member of which must be **BR**, block is enclosed in braces. The syntactical clue used to identify the beginning and end of blocks. ■ For an incoming message, extractHint is used to strip block markers from the raw data. ■ For an outgoing message, extractHint specifies the padding characters to be applied to form a syntactically correct message block. **Note**: The 'braces' used to enclose blocks can comprise *any characters*, such as **{……***blockdata***…...-}** or **:16R:TRADE**……*blockdata*…...**:16S:TRADE** For a full explanation of this element, see "Hint Processing" on page 182 in this chapter.
IdHint	Parameter list	The first member of which must be **FL**, tag is fixed length or **EH,** tag is derived from the extractHint. Syntactical clue used to extract the tag that identifies the block extracted using the extractHint. ■ **FL** is used in most cases, and is used to strip the first few characters from the remaining raw data. ■ **EH** is used to extract the first few characters from the block marker stripped by the extract hint. This is used where the block delimiters are themselves a string, such as, 16R:TRADE. For the webMethods SWIFT FIN Module to identify the block correctly, the text returned by idHint *must match* the value in the block id element. For a full explanation of this element, see "Hint Processing" on page 24.

Elements	Values	Description
loadBlockHint	Parameter list	The first member of which must be **FL**, tag is fixed length. Used for embedded messages, such as in n92, n95 and n96. These messages embed block 4 of a message that was previously processed. The embedded message can be any of the SWIFT message types.
		loadBlockHint is used to determine the sub-template that must be embedded in the current parsing template whenever an embedded message is received. It extracts the message type number (100, 101, 521 etc.) from the data returned by a preceding lineAttribute.
		For a full explanation of this element, see "Hint Processing" on page 24.
		Note: loadBlockHint must be used in conjunction with blockPointer. If these elements are specified, extractHint and idHint are omitted.
blockPointer	4:\r\n	Used in conjunction with loadBlockHint.
		Specifies that block 4 is to be included from the embedded parsing template.

Line Attribute Syntax of a Parsing Template

The lineAttribute elements typically define a single field in the message, although it also is used in the generic parsing template to include an entire block of fields. The syntax of the lineAttribute directive is shown below.

Note that optional parameters are shown in square braces [ ].

```
<block id="id" isMandatory="true|false"
EAImap="string" extractHint="hint" idHint="hint"
[ blockLoadField="true"]>
```

Elements	Values	Description
id	field id or field id list	The field tag in the SWIFT message, including delimiting characters. Mutually exclusive fields are depicted as a comma separated list (see fields 52, 56 and 57 in the example parsing template above).
isMandatory	true / false	**false** indicates the *lineAttribute* is optional and need not be present for the message to be considered valid. **true** indicates the field represented by the lineAttribute must be present in an incoming message, or the message will fail validation. For an outgoing message, the corresponding field in the Enterprise document (specified by the EAImap element) must be populated, or the message will fail validation.
EAImap	document field name or document field name list	The name of the field in the Enterprise document that the lineAttribute is mapped to or from. By convention, the EAImap value for lineAttributes are prefixed with 'O' for optional fields or 'M' for mandatory fields. Mutually exclusive fields are depicted as a comma separated list that must match up with the field ID list.
extractHint	Parameter list	The first member of which must be **BR**, field is enclosed by braces, **DL**, field is delimited, or **CK**, field will contain the entire block data in a single chunk. --- **Note: CK** is used for the generic parsing template only. --- Syntactical clue used to identify the beginning and end of fields. For an incoming message, extractHint is used to identify the end of the field data and to strip any braces from the raw data. For a full explanation of this element, see "Hint Processing" on page 182 in this chapter.

Elements	Values	Description
idHint	Parameter list	The first member of which must be **FL**, ID is fixed length, or **CK**, ID will not be extracted as the field contains the entire block in a single chunk. Syntactical clue used to identify and strip the ID of the field extracted using the extractHint. The idHint is applied to the raw data line and typically uses a fixed length parameter to return the first few characters of the data. For the webMethods SWIFT FIN Module to identify the field correctly, the text returned by idHint using **FL** must match the value in the lineAttribute ID element. For a full explanation of this element, see "Hint Processing" on page 182 in this chapter.
blockLoadField	true (if specified)	Optional element that causes the lineAttribute to be referenced by a subsequent loadBlockHint directive. blockLoadField must precede loadBlockHint in the parsing template.

Hint Processing

ExtractHint, idHint, and loadBlockHint are used in block and lineAttribute directives. The full syntax is provided in the following sections.

Braced fields

Used where the data is enclosed in one or more bracing characters BR, <open brace characters>, <close brace characters>, <tag flag>, <tag position>.

Delimited fields

Used where fields are delimited by a single character DL, <delimiting character>, <tag flag>, <tag position>

Chunk data

Used where data is treated as a single chunk, without further parsing CK, <tag flag>, <tag position> or extractHint CK[, <from position>, <to position>] for idHint.

Fixed length

Used where data occupies a fixed number of characters FL, <from position>, <to position>.

Extract hint

Used to extract a fixed number of characters from the extract hint EH, <from position>, <to position>.

The remaining parameters are given in the following table:

Parameter	Values	Description
open brace character	any sequence of characters	The character(s) used to identify the beginning of the block or field
close brace character	any sequence of characters	The character(s) used to identify the end of the block or field
delimiting character	any character	A single character that separates fields
from position	numeric character	Starting character position used to extract fixed length data from the raw data after the block characters have been stripped off. First character is position 0.
to position	numeric character	Last character position plus one used to extract fixed length data.
tag flag	T or N	**T** (tagged) – the field tag is included with the data. **N** (not tagged) – data does not include the tag field
tag position	S or E	**S** (start) – data tag precedes the data. **E** (end) – data tag follows the data.

Miscellaneous Notes

The following notes should be read and understood before attempting to maintain the webMethods SWIFT FIN Module parsing templates.

■ While SWIFT defines block 1 as the only mandatory block, in general the parsing templates define blocks 1 through 4 as mandatory.

■ Individual optional repeating fields must be defined as a mandatory field within an optional block.

■ When `loadBlockHint` / `blockPointer` is used, `isMandatory` must be true and `isList` must be false.

■ If `extractHint` uses chunk parameter (CK), `idHint` must also be CK.

- ■ `loadBlockField` and associated `loadBlockHint` / `blockPointer` can only occur once in the parsing template.

- ■ Block 3 must include field 108. This is required to correctly process ACK/NACK messages received from the SWIFT network.

webMethods SWIFT FIN Module Sample

Overview

This document provides a real-world example of sending SWIFT FIN messages using the webMethods SWIFT FIN Module. In this example, two financial institutions (trading partners) join in a SWIFT *business process* (also called a conversation) that adheres to the SWIFT standard. You will use various webMethods interfaces to simulate this business process. This document assumes that you are familiar with the webMethods Integration Server, the Server Administrator, webMethods Developer, webMethods Trading Networks, the Trading Networks Console, webMethods Modeler, and webMethods Monitor.

Who Are the Trading Partners?

The sender is EuroClear. EuroClear performs the role of the *initiating party* in this sample. The receiver is UBS Warburg. UBS Warburg performs the role of the *executing party* in this sample.

What Will Be Accomplished?

EuroClear, as the initiating party, will implement the SWIFT FIN messages *MT 564*, *568*, and *566*, with UBS Warburg, the executing party. The process is as follows:

■ **EuroClear sends an *MT 564, Corporate Action Notification*, to UBS Warburg**. This is used to provide an account owner with the details of a corporate action event along with the possible elections or choices available to the account owner.

■ **UBS Warburg sends an *MT 565, Corporate Action Instruction*, to EuroClear.** This message is used to provide the custodian with instructions on how the account owner wishes to proceed with a corporate action event. Instructions include investment decisions regarding the exercise of rights issues, the election of stock or cash when the option is available, and decisions on the conversion or tendering of securities.

■ **EuroClear sends an *MT 568, Corporate Action Narrative*, to UBS Warburg.** This message is used to provide complex instructions or narrative details relating to a corporate action event.

■ **EuroClear sends an *MT 566, Corporate Action Confirmation*, to UBS Warburg.** This message is used to confirm to the account owner that securities and/or cash have been credited/debited to an account as the result of a corporate action event.

Process Overview
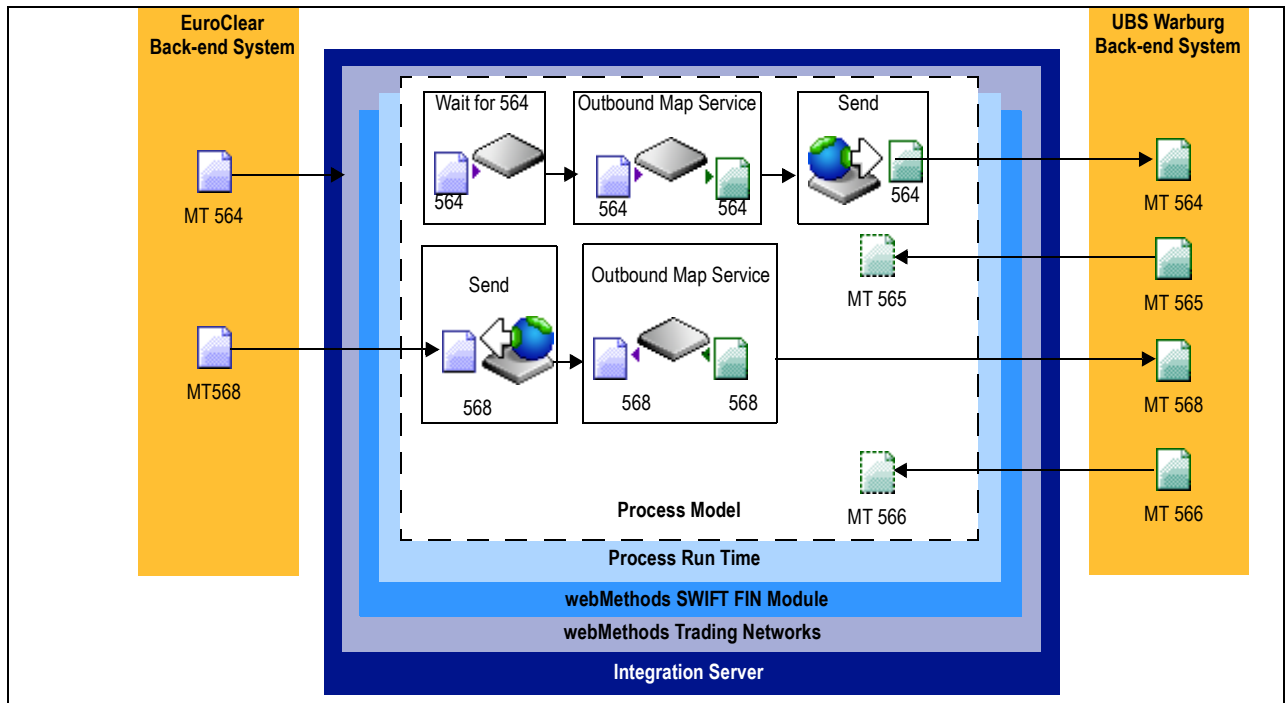
Before You Begin

In this sample scenario, you will use the IS document types and services in the various folders of the packages that make up the webMethods SWIFT FIN Module. This guide assumes that you are familiar with the Integration Server, Trading Networks, Modeler, and Monitor.

Make sure that you have:

■ Installed the Integration Server, Trading Networks, Modeler, and Monitor.

  For information about installing these webMethods components, see the *webMethods Integration Platform Installation Guide.*

■ Installed the webMethods SWIFT FIN Module and the nov03 Category 5 messages.

  For information about installing the webMethods SWIFT FIN Module, see Chapter 2, "Installing the webMethods SWIFT FIN Module" in this guide.

> **Note:** You can run the entire sample on one computer running a single installation of the webMethods6\IntegrationServer with Trading Networks, Modeler, Monitor, and the SWIFT FIN Module. While you can set up two separate machines to simulate an over-the-network connection to a trading partner, you do not need to do so to run the sample. This guide directs you to run the entire sample on a *single* machine.

How Do I Run the Sample?

> **Important!** These steps assume that you already have created database tables for you BIC information and imported you BIC or BIC+ list. For more information, see Chapter 6, "Working with BIC and BIC+ Lists" in this guide.

To run the sample, you will need to perform the following steps after installation:

■   **Step 1: Set Up Partner Profiles**

■   **Step 2: Import TN Document Types and TPAs**

■   **Step 3: Run ImportFINItems for Each TN Document Type**

■   **Step 4: Import the Sample BIC List Database**

■   **Step 5: Import, Generate, and Enable the Process Models**

■   **Step 6: Run the Business Process**

■   **Step 7: View the Business Process**

> **Note:** For more information about installation requirements to run the webMethods SWIFT FIN Module sample, see the previous section, Before You Begin.

Step 1: Set Up Partner Profiles

Perform the following procedures to set up the trading partner profiles that the sample uses. You define in Trading Networks Console the profiles for EuroClear (**My Enterprise**) and UBS Warburg.

In the sample, set up the **My Enterprise** profile for EuroClear (the initiating party in this sample) and a partner profile for UBS Warburg (the executing party in this sample).

Create the My Enterprise Profile for EuroClear

Perform the following steps to set up EuroClear as the **My Enterprise** profile in the Trading Networks Console.

To create the My Enterprise profile for EuroClear

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

2 In the Trading Networks Console, select **Tools ▸ Profile Assistant**. The Trading Networks Console displays the first page of the **Profile Assistant**.

3 Select **Create my profile**, if it is not already selected, and then click **Next**. The Trading Networks Console displays the next page of the **Profile Assistant**, which you use to supply the name of the corporation.

   Note that the Trading Networks Console highlights required fields in blue and also marks the required fields with an asterisk.

4 Type `EuroClear` in the **Corporation Name** field. You also can enter information for the **Unit Name** and import an image file for the corporate logo, but you are only required to fill in the **Corporation Name**.

5 Click **Next** twice. The Trading Networks Console displays the next page of the **Profile Assistant**, which you use to supply the corporation's external ID, which is the BIC for EuroClear.

6 Click **Add New External ID** 📇 , select **BIC** from the **External ID Type** column, and type `MGTCBEBEECL` in the **Value** column.

> **Note:** To run the sample using your own profile, enter your enterprise BIC in this field.

7 Click **Next** until you reach the last page of the Profile Assistant.

8 Click **Finish**.

Verifying and Enabling the EuroClear Profile

To verify that the **My Enterprise** profile for EuroClear is configured correctly for the sample, view the profile. You also must enable the profile. Perform the following to verify and enable the profile.

To verify and enable the profile for EuroClear is configured correctly

1   In the Trading Networks Console, in the Selector panel, select the **My Enterprise** radio button, if it is not already selected.

2   Select **View ▶ Profile**.

3   Click through the tabs of the profile to verify that the profile for EuroClear is configured as shown in the following figure, or using your Enterprise information.

My Enterprise Profile

4   To enable the profile, click **Enable** 💡 to activate the EuroClear profile.

Create a Partner Profile for UBS Warburg

Perform the following steps to set up the partner profile for UBS Warburg in the Trading Networks Console.

To create a partner profile for UBS Warburg

1   In the Trading Networks Console, select **Tools ▶ Profile Assistant**. The Trading Networks Console displays the first page of the **Profile Assistant**.

2   Select **Create partner profile** and click **Next**. The Trading Networks Console displays the next page of the **Profile Assistant.**

Note that Trading Networks Console highlights required fields in blue and also marks the required fields with an asterisk.

3   Type UBS Warburg in the **Corporation Name** field. You also can enter information for the **Unit Name** and import an image file for the corporate logo, but you are only required to fill in the **Corporation Name**.

4   Click **Next** twice. The Trading Networks Console displays the next page of the **Profile Assistant**, which you use to supply the corporation's external ID, which is the BIC for UBS Warburg.

5 Click **Add New External ID** , select **BIC** from the **External ID Type** column, and type UBSWUS3NXXX in the **Value** column.

> **Note:** To run the sample using your another partner, enter your partner's BIC in this field.

6 Click **Next** until you reach the last page of the Profile Assistant.

7 Click **Finish**.

Verifying and Enabling the UBS Warburg Profile

To verify that the **Partner** profile for UBS Warburg is configured correctly for the sample, view the profile. You also must enable the profile. Perform the following to verify and enable the profile.

To verify and enable the profile for UBS Warburg is configured correctly

1 In the Trading Networks Console, in the Selector panel, select the **Partner** radio button, if it is not already selected.

2 Select **View ▸ Profile**.

3 Click through the tabs of the profile to verify that the profile for UBS Warburg is configured as shown in the following figure, or using your Enterprise information.

My Enterprise Profile

4 To enable the profile, click **Enable** to activate the UBS Warburg profile.

> **Note:** The default **External ID Type** is DUNS. You must set the **External ID Type** to BIC.

Step 2: Import TN Document Types and TPAs

You must import the TN document types and TPAs for the messages that the sample will be processing.

Import the TN Document Types

To import the TN Document Types

1   In the Trading Networks Console, click **File ▶ Import**.
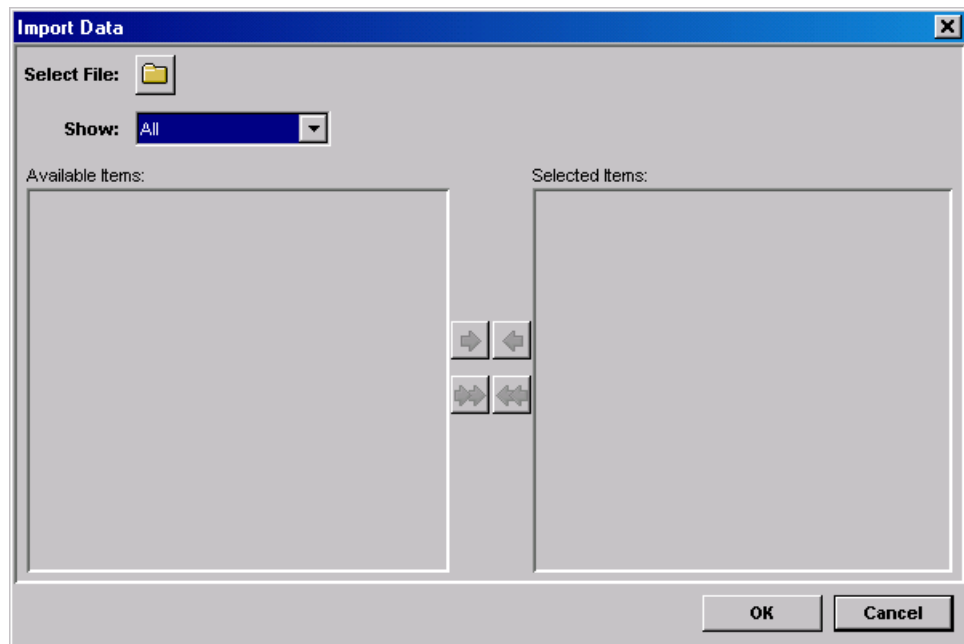
2   To select the TN document type file, click ▢.

3   Navigate to `webMethods6\IntegrationServer\packages\WmFINSamples\data\CorporateActionDocTypes.xml`, and click **Open**. The TN document types are listed on the **Import Data** screen.

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

Import the TPAs

To import the TPAs

1   In the Trading Networks Console, click **File ▶ Import**.
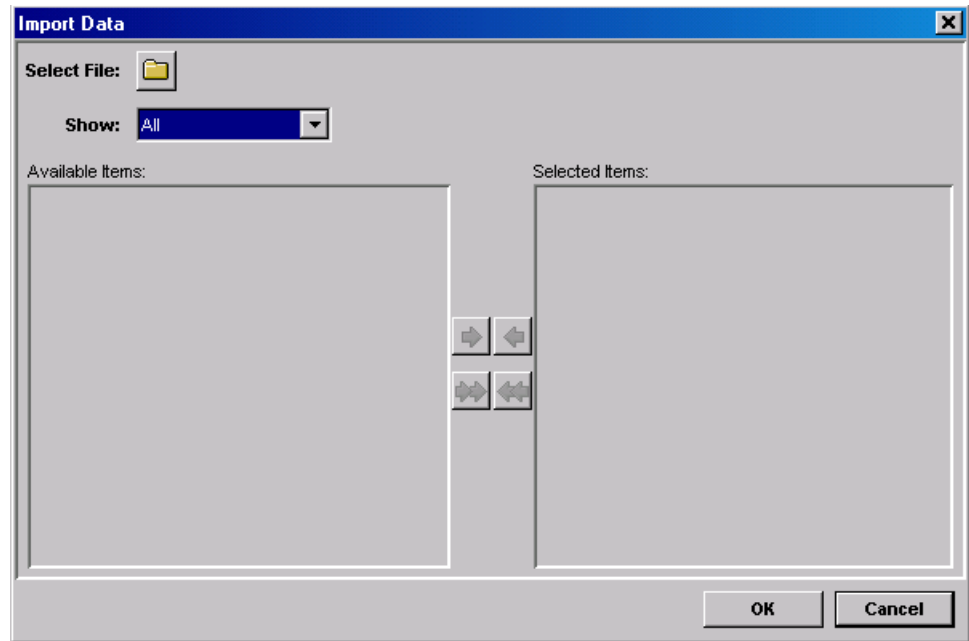
2   To select the TPA file, click 📁.

3   Navigate to `webMethods6\IntegrationServer\packages\WmFINSamples\` `data\CorporateActionTPA.xml`, and click **Open**. The TPAs are listed on the **Import Data** screen.

4   Click ⏩ to select all of the TPAs and then click **OK**. The TPAs are imported.

Step 3: Run ImportFINItems for Each TN Document Type

> ⚠ **Important!** This step assumes that you copied the nov03 Category 5 DFDs when installing the webMethods SWIFT FIN Module.

You must run the wm.fin.dev:importFINItems service for the following message DFDs:

- MT 564
- MT 565
- MT 566
- MT 568
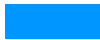
To run the wm.fin.dev:importFINItems service

1   In the Developer, navigate to the wm.fin.dev:importFINItems service, and then click ▶.
    The Input dialog box appears.

Input dialog box

2   Complete the following fields, and then click **OK**:

In this field...	Specify...
msgType	564
version	nov03
format	TAG_BIZNAME
subfieldFlag	false
createDocType	false

In this field...	Specify...
createProcessing Rule	**false**
createTPA	**false**

Note: The **createDocType**, **createProcessingRule**, and **createTPA** fields should be set to **false** because you already have imported the elements that you need.

3 Repeat the previous step for MT 565, 566, and 568.

4 Reload the WmFINMessages package to view the service outputs in Developer.

Note: Refresh Developer to see the WmFINMessages package.

The output is stored in the WmFINMessages package in the *webMethods6\IntegrationServer*\packages\WmFINMessages\nov03).

Step 4: Import the Sample BIC List Database

To import the BIC List database

1 In the Server Administrator Navigation panel, under **Adapters**, click **SWIFT**.

2 On the SWIFT home page, click **Import BIC List**. The **SWIFT Import BIC List** screen appears.

SWIFT Import BIC List screen

3 In the **BIC Type** list, select **BIC**.

4   Navigate to the *webMethods6\IntegrationServer*\packages\WmFINSamples\data directory, and select the sample BIC List database (bicplus.zip).

5   Click **Import BIC List**.

Step 5: Import, Generate, and Enable the Process Models

You will use the sample process models in webMethods Modeler to define (execute steps for) the sample business processes.

When you import, generate, and enable these process models, you are setting up the process that EuroClear, as the initiating party (Account Owner), would run on their installation, and that UBS Warburg, as the executing party (Account Servicer) would run on their installation. In a real life situation, each company would set up only the process model that would run on their installation. Because you are simulating the entire business process on one installation, you must set up *both* process models.

Import the Process Models

Before you can use the sample process models, you must import the following process models into webMethods Modeler:

■   *CorpActions_AccountOwner.model*

■   *CorpActions_AccountServicer.model*

For more details about importing process models, see the *webMethods Modeler User's Guide*.

Importing the Account Owner Process Model

Perform the following steps in Modeler to import the sample EuroClear Account Owner process model.

To import the EuroClear Account Owner process model

1   Start the Integration Server, Server Administrator, and Modeler, if they are not already running.

2   In Modeler, select **File ▸ Import**.

3   Locate the sample process model named *CorpActions_AccountOwner.model* in:

`webMethods6\IntegrationServer\packages\WmFINSample\data\ProcessModels`

where `webMethods6\IntegrationServer` is the directory in which the Integration Server is installed.

4   Click **Open** to import the process model.

5 When the message "**Imported items: Business Process: CorpActions_AccountOwner**" appears, click **OK**.

6 The process model appears in Modeler.

Account Owner Process Model
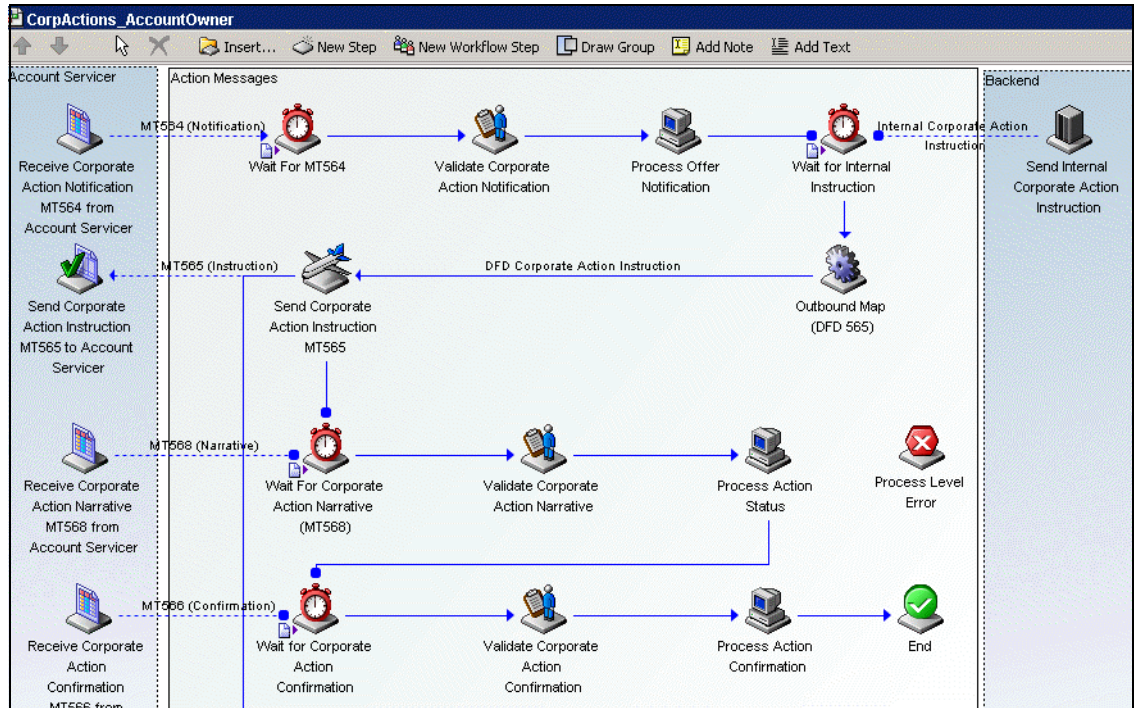
Importing the Account Servicer Process Model

Perform the following steps in Modeler to import the sample UBS Warburg Account Servicer process model.

To import the UBS Warburg Account Servicer process model

1 In Modeler, select **File ▸ Import**.

2 Locate the sample process model named *CorpActions_AccountServicer.model* in:

`webMethods6\IntegrationServer`\packages\WmFINSample\data\ProcessModels

where `webMethods6\IntegrationServer` is the directory in which the Integration Server is installed.

3 Click **Open** to import the process model.

4   When the message "**Imported items: Business Process: CorpActions_AccountServicer.**" appears, click **OK**.

5   The process model appears in Modeler.

Account Servicer Process Model
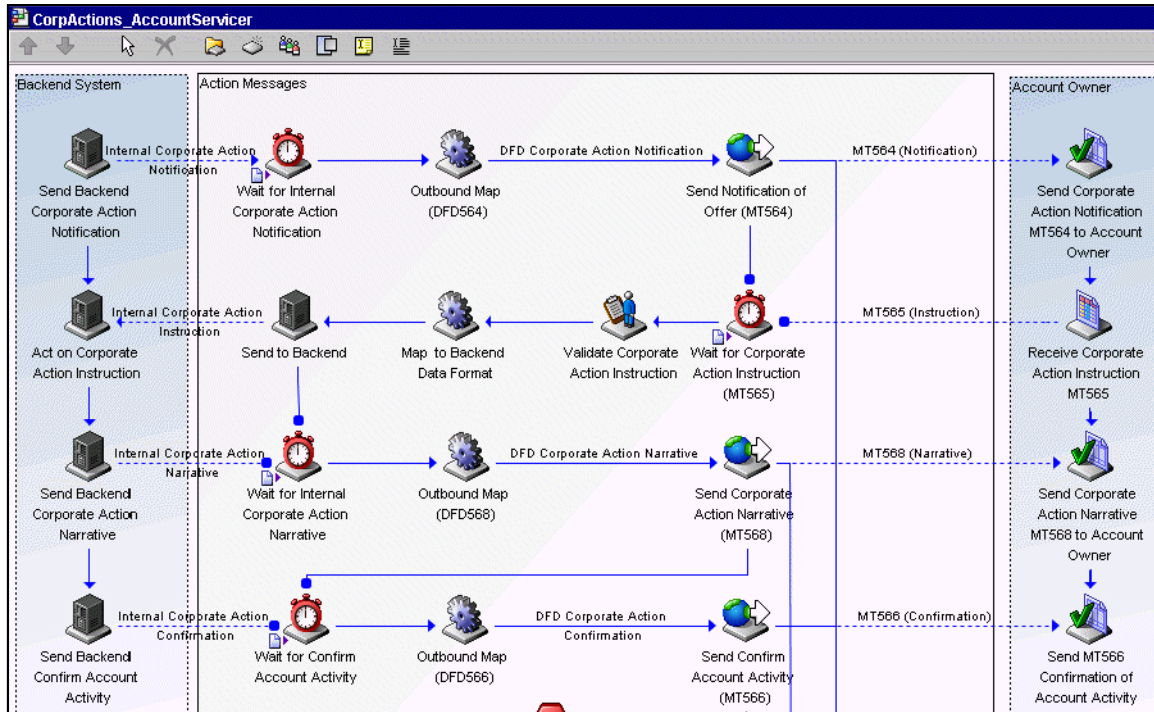
Generate the Process Models

After importing the process models into Modeler, use Modeler to generate the process models to create the run-time elements in the underlying webMethods platform. When you generate the process models, Modeler generates the run-time elements (that is, services, triggers, and process run-time fragments) that actually execute at run-time. Modeler displays messages in the **Implementation Generation Messages** dialog as it creates the run-time elements.

For more details about generating process models, see the *webMethods Modeler User's Guide*.

Generating the Account Owner Process Model

Perform the following steps in Modeler to generate the Account Owner process model.

To generate the Account Owner process model

1   In Modeler, select the Account Owner model.

2   Select **Tools ▶ Generate Business Process**. Modeler begins to generate the run-time elements. The **Implementation Generation Messages** dialog appears.

3   When the message "**CorpActions_AccountOwner Model generated successfully**" appears in the dialog, click **Close**.

4   Update the model for monitoring by selecting **Tools ▶ Update Model for Monitoring**. A dialog appears confirming the process model is ready for monitoring.

5   Click **OK**.

Generating the Account Servicer Process Model

Perform the following steps in Modeler to generate the Account Servicer process model.

To generate the Account Servicer process model

1   In Modeler, select the Account Servicer model.

2   Select **Tools ▶ Generate Business Process**. Modeler begins to generate the run-time elements. The **Implementation Generation Messages** dialog appears.

3   When the message "**CorpActions_AccountServicer Model generated successfully**" appears in the dialog, click **Close**.

4   Update the model for monitoring by selecting **Tools ▶ Update Model for Monitoring**. A dialog appears confirming the process model is ready for monitoring.

5   Click **OK**.

Enable the Process Models

After generating the process models and updating them for monitoring, the process models are disabled. The process run time will not use the process models until you enable both of them. You enable the process models using Monitor.

For more details about enabling process models, see the generating process model chapter in the *webMethods Modeler User's Guide* and the monitoring processes chapter in the *webMethods Monitor User's Guide*.

Enabling the Account Owner Process Model

Perform the following steps in Monitor to enable the Account Owner process model.

To enable the Account Owner process model

1   Open Monitor, if it is not already running.

2   In the Monitor Navigation panel, under the **Processes** heading, click **Process Models**.

3   Locate the **CorpActions_AccountOwner Model** from the list of unused process models. Monitor lists the process model under **Unused Process Models** because no instances of this process model have been executed.

4   In the **CorpActions_AccountOwner Model** row, the **Enabled** column, click **No**.

5   When prompted to confirm your action, click **OK,** and refresh the page to see the enabled status of the process model.

Enabling the Account Servicer Process Model

Perform the following steps in Monitor to enable the Account Servicer process model.

To enable the Account Servicer process model

1   In the Monitor Navigation panel, under the **Processes** heading, click **Process Models**.

2   Locate the **CorpActions_AccountServicer Model** from the list of process models. Monitor lists the process model under **Unused Process Models** because no instances of this process model have been executed.

3   In the **CorpActions_AccountServicer Model** row, the **Enabled** column, click **No**.

4   When prompted to confirm your action, click **OK**, and refresh the page to see the enabled status of the process model.

Step 6: Run the Business Process

In webMethods SWIFT FIN Module, to start a business process on the initiating party's side, the initiating party's back-end system calls wm.ip.cm:process Document. To start a business process on the executing party's side, a service on the initiating party's side sends a SWIFT message to wm.ip.fin:receive. After you start a service, you can monitor its progress using Monitor. For more information about running a business process, see Chapter 12, "Monitoring a Process" in this guide. For more information about monitoring process progress using Monitor, see the next section, "Step 7: View the Business Process" on page 207.

Perform the following procedures to kick off the business process between EuroClear and UBS Warburg.

To run the sample SWIFT business process

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

2   In the Server Administrator navigation panel, under the **Adapters** heading, click **SWIFT**. The Integration Server displays the SWIFT Management page in a new browser window.

3   In Navigation panel, under the **SWIFT** heading, click **CA Account Servicer.**

4   **EuroClear sends an *MT 564, Corporate Action Notification*, to UBS Warburg.**

    Click **Initiate New Corporate Action Notification**. The Corporate Action sample screen appears with the appropriate information already entered in the fields.

    Note: If you used different BICs to define your profiles for this sample, you must enter those BICs in the **Sender's BIC** and **Receiver's BIC** fields.

Corporate Action Notification Sample


```
FIN > Corporate Action Sample

Corporate Action Notification MT564 -- Cash Dividend

Trading Parties
Sender's BIC:  MGTCBEBEECL              Receiver's BIC:  UBSWUS3NXXX

General Information
Reference Number:  CA000002345630
Euroclear Reference Number:  CA000002345630
Mandatory/Optional Indicator:  Mandatory

Financial Instrument Details
Financial
Instrument   ISIN US8765692038                    Financial    TATA TEA LTD (REGS G
ISIN                                               Instrument
number:                                            Detail:
Denominated  USD                                   Options:     Cash
Currency:
Eligible     14300                                 Account      90000
Securities:                                        Information:
Net                                                Gross
Entitlement  0.1761                                Entitlement  0.1911
( USD per                                          ( USD per
share):                                            share):
Record Date:  May 5, 2001                          Payment      August 31, 2001
                                                   Date:
```

5   At the bottom of the page, click **Send**.

If the sample runs successfully, the Integration Server displays the following page.

Corporate Action Notification Confirmation Page


```
FIN > Corporate Action Account Servicer

• Sent Corporate Action Notification Order with reference number CA000002345630
```

6   To view the message details, click the reference number.

Message Details

Corproate Action Notification -- Cash Dividend

Trading Parties

Sender's BIC: MGTCBEBEECL   Receiver's BIC: UBSWUS3NXXX

General Information

Reference Number:	CA000002345630
Euroclear Reference Number:	CA000002345630
Mandatory/Optional Indicator:	MAND

Financial Instrument Details

Financial Instrument ISIN number:	ISIN US8765692038	Financial Instrument Detail:	TATA TEA LTD (REGS GDS)
Denominated Currency:	USD	Options:	CASH
Eligible Securities:	14300	Account Information:	90000
Net Entitlement(USD per share):	0.1761	Gross Entitlement(USD per share):	0.1911
Record Date:	May 5, 2001	Payment Date:	August 31, 2001
Additional Information:			

Corporate Action Notification SWIFT message

7   To view the actual SWIFT message, click the **Corporate Action Notification SWIFT message** link at the bottom of the page.

SWIFT MT 564 Message

564 SWIFT Message

Date: Sat Jun 28 20:23:29 EDT 2003

```
{1:F01MGTCBEBEXECL0000000000}{2:I564UBSWUS3NXXXXN2020}{3:{108:20000000nc2oe8ou}}{4:
:16R:GENL
:20C::CORP//CA000002345630
:20C::SEME//CA000002345630
:23G:NEWM
:22F::CAEV//DVCA
:22F::CAMV//MAND
:98C::PREP//20010901093444
:25D::PROC//COMP
:16R:LINK
:20C::PREV//000002345630
:16S:LINK
:16S:GENL
:16R:USECU
:35B:ISIN US8765692038
```

8   In Navigation panel, under the **SWIFT** heading, click **CA Account Owner**. You also can view the details for the MT 564 message by clicking the reference number of this screen.

9   **UBS Warburg sends an *MT 565, Corporate Action Instruction*, to EuroClear.**

Click **Send Corporate Action Instruction with reference number CA*xxxxxxxxxxxx***. The Corporate Action sample screen appears with the appropriate information already entered in the fields.

> **Note:** If you used different BICs to define your profiles for this sample, you must enter those BICs in the **Sender's BIC** and **Receiver's BIC** fields.

Corporate Action Instruction Sample

FIN > Corporate Action Sample > Corporate Action Narrative

Corproate Action Instruction MT565 -- Cash Dividend

Trading Parties

Sender's BIC: UBSWUS3NXXX   Receiver's BIC: MGTCBEBEECL

General Information

Reference Number: CA000002345630

Account Owner's Reference Number: 00000876344

Financial Instrument Details

Financial Instrument ISIN number: ISIN US8765692038

Financial Instrument Detail: TATA TEA LTD (REGS G

Account Information: 90000

Foreign Exchange transformation: GBP

Send

10   At the bottom of the screen, click **Send**.

If the sample runs successfully, the Integration Server displays the confirmation page.

11   To view the message details, click the reference number.

12   To view the actual SWIFT message, click the link at the bottom of the page.

13   In Navigation panel, under the **SWIFT** heading, click **CA Account Servicer**.

14 Click **Resume the Last Corporate Action Notification**, and then click **Refresh**.

15 **EuroClear sends an** *MT 568, Corporate Action Narrative*, **to UBS Warburg.**

Click **Send Corporate Action Narrative with reference number CA*xxxxxxxxxxxx***. The Corporate Action sample screen appears with the appropriate information already entered in the fields.

Note: If you used different BICs to define your profiles for this sample, you must enter those BICs in the **Sender's BIC** and **Receiver's BIC** fields.

Corporate Action Narrative Sample

FIN > Corporate Action Sample > Corporate Action Narrative

Corporate Action Narrative MT568 -- Cash Dividend

Trading Parties

Sender's BIC: MGTCBEBEECL   Receiver's BIC: UBSWUS3NXXX

General Information

Reference Number: CA000002345630

Euroclear Reference Number: CA000002345630

Financial Instrument Details

Financial Instrument ISIN number: ISIN US8765692038

Financial Instrument Detail: TATA TEA LTD (REGS G

Account Information: 90000

Additional Information:

Send

16 At the bottom of the page, click **Send**.

17 To view the message details, click the reference number.

18 To view the actual SWIFT message, click the link at the bottom of the page.

19 In Navigation panel, under the **SWIFT** heading, click **CA Account Servicer**.

20 Click **Resume the Last Corporate Action Notification**, and then click **Refresh**.

21   EuroClear sends an *MT 566, Corporate Action Confirmation*, to UBS Warburg.

Click **Send Corporate Action Confirmation with reference number CA*xxxxxxxxxxxx***. The Corporate Action sample screen appears with the appropriate information already entered in the fields.

Note: If you used different BICs to define your profiles for this sample, you must enter those BICs in the **Sender's BIC** and **Receiver's BIC** fields.

Corporate Action Confirmation Sample

22   At the bottom of the page, click Send.

23   To view the message details, click the reference number.

24   To view the actual SWIFT message, click the link at the bottom of the page.

Step 7: View the Business Process

To monitor business processes, you can determine the status of your business processes and the activity of the various involved software entities in a couple of ways:

■ **Monitor.** To view the activity and steps that occurred during the execution of the process models, audit the business process in Monitor. You can use this tool to monitor and manage business processes. For more information about Monitor, see the *webMethods Monitor User's Guide*

■ **Trading Networks Transaction Analysis screen.** To view the transactions of documents processed by webMethods SWIFT FIN Module, use the **Transaction Analysis** function in Trading Networks. You can use this log to query and analyze results of documents that are sent or received by Trading Networks. For more information about the Trading Networks Transaction Analysis screen, see the *webMethods Trading Networks-- Building Your Trading Network* guide.

For more information about monitoring business processes in the webMethods SWIFT FIN Module, refer to the chapter about monitoring business processes in the *webMethods SWIFT FIN Module Installation and User's Guide*.

View Activity on the Monitor

If the sample was sent successfully, you can use Monitor to find activity/audit log information for the business process. Monitor's **Recent Activity** area displays all business processes that have run within the last two weeks, organized into sections by status (**Recently Failed**, **Recently Completed**, **Recently Created**, and **Recently Suspended**). Each section lists up to 20 business processes, with the most recent processes listed at the top.

For more information about Monitor and viewing processes, see the *webMethods Monitor User's Guide*.

Perform the following steps in Monitor to audit the sample business process.

To monitor the business process

1 Open Monitor, if it is not already running.

2 In the Monitor Navigation panel, under the **Processes** heading, click **Recent Activity**. Monitor displays all recent processes.

Recent Activity Screen

3   In the **Recently Created Processes** section, click one the **ID** for one of the Corporate Action processes.

Recently Created Processes Screen

4 Monitor displays the business process, as shown below. A check mark (✔) next to individual steps on the page illustrates the progress of the business process.

Process Instance Status Screen

The bottom portion of this screen displays the status of individual steps, activity messages, and information about custom fields.

Bottom of Process Instance Status Screen

Name	Status	Process Iteration	Step Iteration	Time Ended
Step Status				
Wait forSignal	Waiting	1	1	2003-07-03 14:43:05.295 EDT
Wait For Corporate Action Narrative (MT568)	Waiting	1	1	2003-07-03 14:43:05.254 EDT
Send Corporate Action Instruction MT565	Completed	1	1	2003-07-03 14:43:03.111 EDT
Outbound Map MT565	Completed	1	1	2003-07-03 14:43:02.420 EDT
Wait for Internal Instruction	Completed	1	1	2003-07-03 14:43:01.830 EDT
Process Offer Notification	Completed	1	1	2003-07-03 14:42:47.709 EDT
Inbound Validation MT564	Completed	1	1	2003-07-03 14:42:47.188 EDT
Wait For MT564	Completed	1	1	2003-07-03 14:42:42.712 EDT

⊞ Activity Messages

⊞ Custom Fields

View Transactions via the Trading Networks Console

You can view transactions (that is, the individual documents in the webMethods SWIFT FIN Module business process) that occurred during the execution of the process by using Trading Networks. For more information about viewing transactions with the Trading Networks Console, see the *webMethods Trading Networks--Managing and Analyzing Your Trading Networks*.

To view transactions using the Trading Networks Console, perform the following steps.

To view transactions via the Trading Networks Console

1   Start Trading Networks, if it is not already running.

2   In the Trading Networks Console, complete one of the following steps:

 ■   Click the **My Enterprise** radio button to view the data that are associated with EuroClear, or

■ Click the **Partner** radio button to view the data that are associated with UBS Warburg.

3 Select **View ▶ Transaction Analysis**.

4 If the query panels are not displayed, select **Transactions ▶ Show/Hide Query** to display the query panels.

5 Using the query panels, specify the criteria you want to use to search for the documents within the business process. To use a criterion, you first must select the check box for the criterion, then fill in related fields. For example, you can search by selecting the **Conversation ID** criterion and then entering a valid Conversation ID number.

Example Transaction Analysis Query

Transaction Analysis: **EuroClear**				
Sent				
Basic Criteria	Custom Criteria	Detail View	Summary View	

☐	Sender:	EuroClear	☐	Document ID:
☐	Receiver:	EuroClear	☐	Group ID:
☐	Type:	Internal Corporate Action Confirmation	☐	Conversation ID:
☐	Processing Status:		☐	Date Received: Today
☐	User Status:			☐ Beg
				☐

Date Received	Document Type	Sender	Receiver	Processing Sta
2003-06-30 18:36:44.893	MT568	EuroClear	UBS Warburg	NEW
2003-06-30 18:36:39.607	Internal Corporate Action N...	EuroClear	UBS Warburg	NEW
2003-06-30 18:32:06.493	MT565	UBS Warburg	EuroClear	NEW

6 Select **Transactions ▶ Run Query**. Trading Networks updates the display in the bottom panel, as illustrated in the following figure.

Example Transaction Analysis Query Results

Date Received	Document Type	Sender	Receiver	Processing S
2003-06-28 20:37:01.447	Internal Corporate Action C...	EuroClear	UBS Warburg	NEW
2003-06-27 15:52:59.75	Internal Corporate Action C...	EuroClear	UBS Warburg	NEW

7   To see the contents of a document, click the row for the document, and select
Transactions ▶ View Document. Trading Networks displays the **Document Details** screen,
as illustrated in the following figure.

Document Details Screen

Index

■ ■ ■ **webMethods SWIFT FIN Module Installation and User's Guide Version 6.0.1 Service Pack 2**