Oracle® Java CAPS SWIFT Message Library User's Guide

Part No: 821–2582
December 2011

120126@25097

Contents

SWIFT Message Library

This document provides information on installing the SWIFT message library as well as instructions on using the library message validation features and some associated sample projects. The following sections provide installation, library, and validation information for SWIFT messages:

Overview of SWIFT Message Libraries

The Society for Worldwide Interbank Financial Telecommunication (SWIFT) Message Library contains template messages for use with the Oracle Java Composite Application Platform Suite (Java CAPS). These messages correspond to the SWIFT user-to-user message types employed by its SWIFT network. The library provides an individual object type definition for each SWIFT message type, as defined in the SWIFT standards documentation.

Each SWIFT message library represents a corresponding SWIFT message type. You can use these libraries to transfer SWIFT message data with the Oracle Java CAPS.

Library Versions and Access

SWIFT periodically revises their message types, adding to or subtracting from the total set of message types, and modifying the definitions of individual message types. New sets are identified with the year they are issued, such as 2009 and 2010. New SWIFT Message Libraries are released corresponding to each revised set of SWIFT message types. Java CAPS supports the message types for the current and previous year, so Java CAPS 6.3 supports the 2009 and 2010 libraries.

You must install each year's version via a separate `SAR` file.

Installing the SWIFT Message Library

The SWIFT Message Library is not installed as part of the standard Java CAPS installation, and needs to be installed after you install Java CAPS. For instructions on installing Repository-based components, see "Installing Additional Repository-Based Java CAPS Components" in *Installing Additional Components for Oracle Java CAPS 6.3*.

The annual SWIFT Message Libraries rely on the following components. The Adapters are installed in the standard Java CAPS installation, and you can install the SWIFT Message Library when you install the annual libraries.

- **File Adapter** (the File Adapter is used by most sample Projects)
- **Batch Adapter** (the Batch Adapter is required to run the MX validation sample project)
- **SWIFT Message Library** (this includes all versions of the library)

When you install the SWIFT libraries, install `SwiftOTDLibrary.sar` and one or both of the following files, depending on which versions of SWIFT messages you are using:

- `SwiftOTDLibrary2010.sar`
- `SwiftOTDLibrary2009.sar`

Increasing the Heap Size

Because of the size of the SWIFT Message Library, the heap size may need to be increased before using the library. If the heap size is not increased, you might receive OutOfMemoryError messages when you try to build, deploy, or run a SWIFT Message Library Project.

If you receive this message while building or deploying a SWIFT Project, increase the heap size and then retry the process. If you receive the message during runtime, increase the GlassFish server heap size. NetBeans automatically determines the heap size based on available memory, but if you receive the message in NetBeans, increase the heap size for NetBeans as well.

▼ To Increase the Heap Size for GlassFish

1. **In the GlassFish Admin Console, click Application Server in the left navigation panel.**

2. **Click the JVM Setting tab, and then the JVM Options subtab.**

3 **Change the** *–Xmx512m* **option to** *–Xmx768m*

▼ To Increase the Heap Size for NetBeans

1 **Navigate to** *JavaCAPS_Home*`\netbeans\etc` **and open the file** `netbeans.conf` **in a text editor.**

2 **Modify the** `netbeans_default_options` **property by changing the** *–Xmx512m* **option to** *–Xmx768m*.

Note – Your Xmx option might have a default value other than 512 depending on your available memory.

Using the SWIFT Message Library

This section explains, lists, and provides a cross-reference for the SWIFT Message Library message types.

- "SWIFT Message Type OTDs" on page 7
- "SWIFT Message Type Reference" on page 8

SWIFT Message Type OTDs

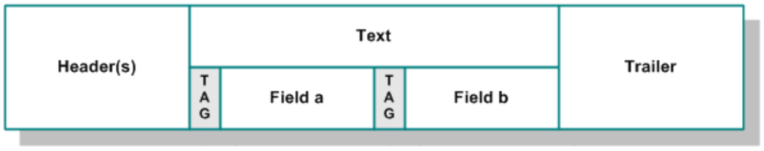
This section provides a general overview of the SWIFT message types and their OTDs.

- "SWIFT Message Structure" on page 7
- "Message Library and Collaboration Locations in NetBeans" on page 8

SWIFT Message Structure

Messages used by the SWIFT network have a maximum of five components (see "SWIFT Message Structure" on page 7), as follows:

- Basic header block
- Application header block
- User header block (optional)
- Text block
- Trailer block



Each field component in the text block is preceded by a field tag. There are no field tags in the header and trailer blocks. The one exception to this format is MT 121, EDIFACT FINPAY, which has a single text field with no field tag identifier.

Information about a field common to all message types in which that field is used is found in the *Standards - General Field Definitions* volume of the *SWIFT User Handbook*. Information about a field specific to its use with a particular message type is found in the field specifications section of the *Standards* volume of the *SWIFT User Handbook* for that message type.

Message Library and Collaboration Locations in NetBeans

The SWIFT Object Type Definitions (OTDs) and Collaborations are listed on the NetBeans Projects window under `CAPS Components Library\Message Library\Swift`. Each message library version is contained in its own folder specifying the year. This includes the MT Fund, Generic, and Acknowledgement OTDs and validation Collaborations.

The Validation Collaborations folder contains the Collaboration Definitions that enable the validation features of the SWIFT Message Library. See "SWIFT Message Library JAR Files" on page 23 for details. The Category 5 folder contains the SWIFT MT Funds message template OTDs. See "Parse Debug Level Message Example" on page 48 for details.

The `bic.jar` file allows you to update the BICDirService feature. See "SWIFT Message Library JAR Files" on page 23 for details.

SWIFT Message Type Reference

SWIFT groups message types into the following categories:

Customer Payments and Cheques

- See "Category 1 Messages" on page 9.

Financial Institution Transfers

- See "Category 2 Messages" on page 10.

Treasury Markets: Foreign Exchange and Derivatives

- See "Category 3 Messages" on page 11.

Collections and Cash Letters

- See "Category 4 Messages" on page 13.

Securities Markets

- See "Category 5 Messages" on page 14.

Treasury Markets: Precious Metals and Syndications

- See "Category 6 Messages" on page 17.

Documentary Credits and Guarantees

- See "Category 7 Messages" on page 18.

Travellers Cheques

- See "Category 8 Messages" on page 19.

Cash Management and Customer Status

- See "Category 9 Messages" on page 20.

The remainder of this section discusses these categories and the message types within each category. The 2009 and 2010 versions of the SWIFT Message Library are provided with the SWIFT Message Library. You must install each version from a separate `SAR` file. The tables in the following sections list each supported message type, though some are only supported in version 2009 or version 2010, but not both.

For explanations of the different versions, see the SWIFT Web site at `http://www.swift.com`.

Category 1 Messages

The table below lists the Category 1 message types, Customer Payments and Cheques, with the type designation MT 1xx.

TABLE 1 Customer Payments and Cheques

SWIFT Message Type	Description
MT 101	Request for Transfer
MT 102	Multiple Customer Credit Transfer
MT 102+(STP)	Multiple Customer Credit Transfer (STP)

TABLE 1 Customer Payments and Cheques *(Continued)*

SWIFT Message Type	Description
MT 103	Single Customer Credit Transfer
MT 103+ (REMIT)	Single Customer Credit Transfer (REMIT)
MT 103+ (STP)	Single Customer Credit Transfer (STP)
MT 104	Direct Debit and Request for Debit Transfer Message (STP)
MT 105	EDIFACT Envelope
MT 107	General Direct Debit Message
MT 110	Advice of Cheque(s)
MT 111	Request for Stop Payment of a Cheque
MT 112	Status of a Request for Stop Payment of a Cheque
MT 121	Multiple Interbank Funds Transfer (EDIFACT FINPAY Message)
MT 190	Advice of Charges, Interest, and Other Adjustments
MT 191	Request for Payment of Charges, Interest and Other Expenses
MT 192	Request for Cancellation
MT 195	Queries
MT 196	Answers
MT 198	Proprietary Message
MT 199	Free Format Message

Category 2 Messages

The table below lists the Category 2 message types, Financial Institution Transfers, with the type designation MT 2xx.

TABLE 2 Financial Institution Transfers

SWIFT Message Type	Description
MT 200	Financial Institution Transfer for its Own Account
MT 201	Multiple Financial Institution Transfer for its Own Account

TABLE 2 Financial Institution Transfers *(Continued)*

SWIFT Message Type	Description
MT 202	General Financial Institution Transfer
MT 202+(COV)	General Financial Institution Transfer – Cover
MT 203	Multiple General Financial Institution Transfer
MT 204	Financial Markets Direct Debit Message
MT 205	Financial Institution Transfer Execution
MT 207	Request for Financial Institution Transfer
MT 210	Notice to Receive
MT 256	Advice of Non-Payment of Cheques
MT 290	Advice of Charges, Interest and Other Adjustments
MT 291	Request for Payment of Charges, Interest and Other Expenses
MT 292	Request for Cancellation
MT 295	Queries
MT 296	Answers
MT 298	Proprietary Message
MT 299	Free Format Message

Category 3 Messages

The table below lists the Category 3 message types, Treasury Markets, Foreign Exchange, Money Markets, and Derivatives, with the type designation MT 3xx.

TABLE 3 Treasury Markets, Foreign Exchange, Money Markets, and Derivatives

SWIFT Message Type	Description
MT 300	Foreign Exchange Confirmation
MT 303	Forex/Currency Option Allocation Instruction
MT 304	Advice/Instruction of a Third Party Deal
MT 305	Foreign Currency Option Confirmation

TABLE 3 Treasury Markets, Foreign Exchange, Money Markets, and Derivatives *(Continued)*

SWIFT Message Type	Description
MT 306	Foreign Currency Option Confirmation
MT 307	Advice/Instruction of a Third Party FX Deal
MT 308	Instruction for Gross/Net Settlement of Third Party FX Deals
MT 320	Fixed Loan/Deposit Confirmation
MT 321	Instruction to Settle a Third Party Loan/Deposit
MT 330	Call/Notice Loan/Deposit Confirmation
MT 340	Forward Rate Agreement Confirmation
MT 341	Forward Rate Agreement Settlement Confirmation
MT 350	Advice of Loan/Deposit Interest Payment
MT 360	Single Currency Interest Rate Derivative Confirmation
MT 361	Cross Currency Interest Rate Swap Confirmation
MT 362	Interest Rate Reset/Advice of Payment
MT 364	Single Currency Interest Rate Derivative Termination/Recouponing Confirmation
MT 365	Single Currency Interest Rate Swap Termination/Recouponing Confirmation
MT 380	Foreign Exchange Order
MT 381	Foreign Exchange Order Confirmation
MT 390	Advice of Charges, Interest and Other Adjustments
MT 391	Request for Payment of Charges, Interest and Other Expenses
MT 392	Request for Cancellation
MT 395	Queries
MT 396	Answers
MT 398	Proprietary Message
MT 399	Free Format Message

Category 4 Messages

The table below lists the Category 4 message types, Collections and Cash Letters, with the type designation MT 4xx.

TABLE 4 Collections and Cash Letters

SWIFT Message Type	Description
MT 400	Advice of Payment
MT 405	Clean Collection
MT 410	Acknowledgment
MT 412	Advice of Acceptance
MT 416	Advice of Non-Payment/Non-Acceptance
MT 420	Tracer
MT 422	Advice of Fate and Request for Instructions
MT 430	Amendment of Instructions
MT 450	Cash Letter Credit Advice
MT 455	Cash Letter Credit Adjustment Advice
MT 456	Advice of Dishonor
MT 490	Advice of Charges, Interest and Other Adjustments
MT 491	Request for Payment of Charges, Interest and Other Expenses
MT 492	Request for Cancellation
MT 495	Queries
MT 496	Answers
MT 498	Proprietary Message
MT 499	Free Format Message

Category 5 Messages

The table below lists the Category 5 message types, Securities Markets, with the type designation MT 5xx.

TABLE 5 Securities Markets

SWIFT Message Type	Description
MT 500	Instruction to Register
MT 501	Confirmation of Registration or Modification
MT 502	Order to Buy or Sell
MT 502 (FUNDS)	Order to Buy or Sell (FUNDS)
MT 503	Collateral Claim
MT 504	Collateral Proposal
MT 505	Collateral Substitution
MT 506	Collateral and Exposure Statement
MT 507	Collateral Status and Processing Advice
MT 508	Intra-Position Advice
MT 509	Trade Status Message
MT 509 (FUNDS)	Trade Status Message (FUNDS)
MT 510	Registration Status and Processing Advice
MT 513	Client Advice of Execution
MT 514	Trade Allocation Instruction
MT 515	Client Confirmation of Purchase or Sale
MT 515 (FUNDS)	Client Confirmation of Purchase or Sale (FUNDS)
MT 516	Securities Loan Confirmation
MT 517	Trade Confirmation Affirmation
MT 518	Market-Side Securities Trade Confirmation
MT 519	Modification of Client Details
MT 524	Intra-Position Instruction
MT 526	General Securities Lending/Borrowing Message

TABLE 5 Securities Markets *(Continued)*

SWIFT Message Type	Description
MT 527	Triparty Collateral Instruction
MT 528	ETC Client-Side Settlement Instruction
MT 529	ETC Market-Side Settlement Instruction
MT 530	Transaction Processing Command
MT 535	Statement of Holdings
MT 535 (FUNDS)	Statement of Holdings (FUNDS)
MT 536	Statement of Transactions
MT 537	Statement of Pending Transactions
MT 538	Statement of Intra-Position Advice
MT 540	Receive Free
MT 541	Receive Against Payment
MT 542	Deliver Free
MT 543	Deliver Against Payment
MT 544	Receive Free Confirmation
MT 545	Receive Against Payment Confirmation
MT 546	Deliver Free Confirmation
MT 547	Deliver Against Payment Confirmation
MT 548	Settlement Status and Processing Advice
MT 549	Request for Statement/Status Advice
MT 558	Triparty Collateral Status and Processing Advice
MT 559	Paying Agent's Claim
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 569	Triparty Collateral and Exposure Statement

TABLE 5 Securities Markets *(Continued)*

SWIFT Message Type	Description
MT 574 (IRSLST)	IRS 1441 NRA (Beneficial Owners' List)
MT 574 (W8BENO)	IRS 1441 NRA (Beneficial Owner Withholding Statement)
MT 575	Report of Combined Activity
MT 576	Statement of Open Orders
MT 577	Statement of Numbers
MT 578	Statement of Allegement
MT 579	Certificate Numbers
MT 581	Collateral Adjustment Message
MT 582	Reimbursement Claim or Advice
MT 584	Statement of ETC Pending Trades
MT 586	Statement of Settlement Allegements
MT 587	Depositary Receipt Instruction
MT 588	Depositary Receipt Confirmation
MT 589	Depositary Receipt Status and Processing Advice
MT 590	Advice of Charges, Interest and Other Adjustments
MT 591	Request for Payment of Charges, Interest and Other Expenses
MT 592	Request for Cancellation
MT 595	Queries
MT 596	Answers
MT 598	Proprietary Message
MT 599	Free Format Message

Category 6 Messages

The table below lists the Category 6 message types, Treasury Markets, Precious Metals, with the type designation MT 6xx.

TABLE 6 Treasury Markets, Precious Metals

SWIFT Message Type	Description
MT 600	Precious Metal Trade Confirmation
MT 601	Precious Metal Option Confirmation
MT 604	Precious Metal Transfer/Delivery Order
MT 605	Precious Metal Notice to Receive
MT 606	Precious Metal Debit Advice
MT 607	Precious Metal Credit Advice
MT 608	Statement of a Metal Account
MT 609	Statement of Metal Contracts
MT 620	Metal Fixed Loan/Deposit Confirmation
MT 643	Notice of Drawdown/Renewal
MT 644	Advice of Rate and Amount Fixing
MT 645	Notice of Fee Due
MT 646	Payment of Principal and/or Interest
MT 649	General Syndicated Facility Message
MT 690	Advice of Charges, Interest and Other Adjustments
MT 691	Request for Payment of Charges, Interest and Other Expenses
MT 692	Request for Cancellation
MT 695	Queries
MT 696	Answers
MT 698	Proprietary Message
MT 699	Free Format Message

Category 7 Messages

The table below lists the Category 7 message types, Treasury Markets, Syndication, with the type designation MT 7xx.

TABLE 7 Treasury Markets, Syndication

SWIFT Message Type	Description
MT 700	Issue of a Documentary Credit
MT 701	Issue of a Documentary Credit
MT 705	Pre-Advice of a Documentary Credit
MT 707	Amendment to a Documentary Credit
MT 710	Advice of a Third Bank's Documentary Credit
MT 711	Advice of a Third Bank's Documentary Credit
MT 720	Transfer of a Documentary Credit
MT 721	Transfer of a Documentary Credit
MT 730	Acknowledgment
MT 732	Advice of Discharge
MT 734	Advice of Refusal
MT 740	Authorization to Reimburse
MT 742	Reimbursement Claim
MT 747	Amendment to an Authorization to Reimburse
MT 750	Advice of Discrepancy
MT 752	Authorization to Pay, Accept or Negotiate
MT 754	Advice of Payment/Acceptance/Negotiation
MT 756	Advice of Reimbursement or Payment
MT 760	Guarantee
MT 767	Guarantee Amendment
MT 768	Acknowledgment of a Guarantee Message
MT 769	Advice of Reduction or Release
MT 790	Advice of Charges, Interest and Other Adjustments

TABLE 7 Treasury Markets, Syndication *(Continued)*

SWIFT Message Type	Description
MT 791	Request for Payment of Charges, Interest and Other Expenses
MT 792	Request for Cancellation
MT 795	Queries
MT 796	Answers
MT 798	Proprietary Message
MT 799	Free Format Message

Category 8 Messages

The table below lists the Category 8 message types, Travellers Cheques, with the type designation MT 8xx.

TABLE 8 Travellers Cheques

SWIFT Message Type	Description
MT 800	T/C Sales and Settlement Advice [Single]
MT 801	T/C Multiple Sales Advice
MT 802	T/C Settlement Advice
MT 810	T/C Refund Request
MT 812	T/C Refund Authorization
MT 813	T/C Refund Confirmation
MT 820	Request for T/C Stock
MT 821	T/C Inventory Addition
MT 822	Trust Receipt Acknowledgment
MT 823	T/C Inventory Transfer
MT 824	T/C Inventory Destruction/Cancellation Notice
MT 890	Advice of Charges, Interest and Other Adjustments
MT 891	Request for Payment of Charges, Interest and Other Expenses

TABLE 8 Travellers Cheques *(Continued)*

SWIFT Message Type	Description
MT 892	Request for Cancellation
MT 895	Queries
MT 896	Answers
MT 898	Proprietary Message
MT 899	Free Format Message

Category 9 Messages

The table below lists the Category 9 message types, Cash Management and Customer Status, with the type designation MT 9xx.

TABLE 9 Cash Management and Customer Status

SWIFT Message Type	Description
MT 900	Confirmation of Debit
MT 910	Confirmation of Credit
MT 920	Request Message
MT 935	Rate Change Advice
MT 940	Customer Statement Message
MT 941	Balance Report
MT 942	Interim Transaction Report
MT 950	Statement Message
MT 970	Netting Statement
MT 971	Netting Balance Report
MT 972	Netting Interim Statement
MT 973	Netting Request Message
MT 985	Status Inquiry
MT 986	Status Report

TABLE 9 Cash Management and Customer Status *(Continued)*

SWIFT Message Type	Description
MT 990	Advice of Charges, Interest and Other Adjustments
MT 991	Request for Payment of Charges, Interest and Other Expenses
MT 992	Request for Cancellation
MT 995	Queries
MT 996	Answers
MT 998	Proprietary Message
MT 999	Free Format Message

Validation Collaborations

The table below lists the Validation Collaboration. Validation Collaboration Definitions are provided for many key SWIFT message types.

TABLE 10 Common Group Messages

Validation Collaborations	Validates OTD/Message Type
ValidateMt_101	MT_101 - Request for Transfer
ValidateMt_103	MT_103 – Single Customer Credit Transfer
ValidateMt_103_STP	MT_103_STP - Single Customer Credit Transfer
ValidateMt_202	MT_202 - General Financial Institution Transfer
ValidateMt_210	MT_210 – Notice to Receive
ValidateMt_300	MT_300 - Foreign Exchange Confirmation
ValidateMt_320	MT_320 – Fixed Loan/Deposit Confirmation
ValidateMt_350	MT_350 – Advice of Loan/Deposit Interest Payment
ValidateMt_500	MT_500 – Instruction to Register
ValidateMT_502	MT_502 – Order to Buy or Sell
ValidateMt_502_FUNDS	MT_502_FUNDS - Order to Buy or Sell (FUNDS)
ValidateMt_508	MT_508 – Intra-Position Advice

TABLE 10 Common Group Messages *(Continued)*

Validation Collaborations	Validates OTD/Message Type
ValidateMt_509	MT_509 – Trade Status Message
ValidateMt_513	MT_513 – Client Advice Execution
ValidateMt_515	MT_515 – Client Confirmation of Purchase or Sell
ValidateMt_515_FUNDS	MT_515_FUNDS - Client Confirmation of Purchase or Sale (FUNDS)
ValidateMt_517	MT_517 – Trade Confirmation Affirmation
ValidateMt_518	MT_518 – Market Side Security Trade
ValidateMt_527	MT_527 – Tri-party Collateral Instruction
ValidateMt_535	MT_535 - Statement of Holdings
ValidateMt_536	MT_536 - Statement of Transactions
ValidateMt_537	MT_537 - Statement of Pending Transactions
ValidateMt_538	MT_538 — Statement of Intra-Position Advices
ValidateMt_540	MT_540 - Receive Free
ValidateMt_541	MT_541 - Receive Against Payment
ValidateMt_542	MT_542 - Deliver Free
ValidateMt_543	MT_543 - Deliver Against Payment
ValidateMt_544	MT_544 - Receive Free Confirmation
ValidateMt_545	MT_545 - Receive Against Payment Confirmation
ValidateMt_546	MT_546 - Deliver Free Confirmation
ValidateMt_547	MT_547 - Deliver Against Payment Confirmation
ValidateMt_548	MT_548 - Statement Status and Processing Advice
ValidateMt_558	MT_558 – Tri-party Collateral Status and Processing Advice
ValidateMt_559	MT_559 – Paying Agent's Claim
ValidateMt_564	MT_564 – Corporate Action Notification
ValidateMt_565	MT_565 – Corporate Action Instruction
ValidateMt_566	MT_566 – Corporate Action Confirmation
ValidateMt_567	MT_567 – Corporate Action Status and Processing Advice
ValidateMt_568	MT_568 – Corporate Action Narrative

TABLE 10 Common Group Messages *(Continued)*

Validation Collaborations	Validates OTD/Message Type
ValidateMt_576	MT_576 – Tri-party Collateral and Exposure Statement
ValidateMt_578	MT_578 – Statement Allegement
ValidateMt_586	MT_586 – Statement of Settlement Allegement
ValidateMt_590	MT_590 – Advice of Charges, Interest and Other Adjustment
ValidateMt_595	MT_595 – Queries
ValidateMt_596	MT_596 – Answers
ValidateMt_598	MT_598 – Property Message
ValidateMt_900	MT_900 - Confirmation of Debit
ValidateMt_910	MT_910 - Confirmation of Credit
ValidateMt_940	MT_940 - Customer Statement Message
ValidateMt_950	MT_950 - Statement Message

For information about the Validation Collaborations, see "Using Message Validation Features" on page 24.

SWIFT Generic Library

The SWIFT OTD Libraries include a Generic OTD used to route SWIFT messages. The Generic OTD can be used to parse any valid SWIFT message, allowing you to unmarshal and read the message headers to determine the message type, while leaving the message data as a String. Messages can then be routed to the appropriate OTD for that message type.

SWIFT Message Library JAR Files

The SWIFT Message Library include four JAR files, `bic.jar`, `BICPlusIBAN.jar`, `SwiftOTDLibrary.jar`, and `SEPA.jar`. These files are visible from the NetBeans Projects Window Swift directory. These JAR files provide the classes and methods that support the Validation Collaborations.

Using Message Validation Features

This section explains how to use specialized message validation features and Projects available with the SWIFT Message Library.

- "Message Validation Rules" on page 27
- "In Collaboration Validation Methods" on page 29

Basic Validation Features

The SWIFT Message Library performs validation operations through Java-based Collaboration Definitions that are packaged with the library. These Collaboration Definitions have the following validation features:

- **Message Format Validation Rules (MFVRs)**: A set of functions that accurately tests the semantic validity of a given subset of the SWIFT messages.
- **Market Practice Rules (MPRs)**: A set of functions that accurately test the semantic and syntactical validity of a particular subset of the SWIFT messages called the 500 series.
- **BICDirService (Bank Identifier Code Directory Service) Lookup**: A set of methods that provide search and validation functions for SWIFT's BIC codes and ISO currency and country codes. The information used for look ups and validation is provided by SWIFT.
- **BICPlusIBAN Validation**: A set of methods that provide search and validation functions for SWIFT's BIC and IBAN codes. The SWIFT Message Library implements the suggested validation rules provided by SWIFT. For more information, see the BICPlusIBAN Directory Technical Specifications from SWIFT.

These validation features share the following use characteristics:

- Each available method and function is fully incorporated into and used by the appropriate SWIFT message OTD.
- You can modify the validation rules for your system if desired. Customize the Collaboration's validation rules by checking the Collaboration out (from Version Control) and modifying the validation Collaboration code. The sample implementation and instructions are provided in the validation Collaboration as Java comments.
- Validation methods and functions have no dependencies outside SWIFT data files and the individual OTD.

Installing the OTD library allows the Enterprise Service Bus and any Adapters you use with the library to provide full support for these features. The rest of this section provides a summary of how these features operate with the SWIFT Message Library.

Validation Components

In addition to components described under "Basic Validation Features" on page 24, the SWIFT Message Library also contains the following basic components:

- **SWIFT OTDs (2009 and 2010)**: OTDs in the SWIFT Message Library that represent standard SWIFT message types.
- **MT Funds OTDs**: Specialized OTDs that allow you to automate the specialized funds operations. This category contains FIN-based OTDs.
- **Validation Collaboration Definitions**: Components that define the validation logic and that are provided for specific SWIFT message types. See "Validation Collaboration Definitions" on page 25 for details.
- **Sample Projects**: Sample Projects are provided as examples of validation implementation. See "SWIFT Projects" on page 34 for details.

Validation Methods

The SWIFT Message Library provides three OTD API methods, `validate()`, `validateMPR()`, and `validateMFVR()`, that can be invoked by a Collaboration to validate SWIFT OTDs directly in the Collaboration (see "In Collaboration Validation Methods" on page 29). This is an alternative to using the Validation Collaboration Definitions.

Validation Collaboration Definitions

Validation Collaboration Definitions are provided for many key SWIFT message types. These Collaboration Definitions, when combined with Enterprise Service Bus Services, become Java-based Collaborations that verify the syntax of the SWIFT messages.

Messages are verified by parsing the data into a structure that conforms to the SWIFT standard specifications. The validation functions use the Validation Collaborations to access specific data that is then verified according the algorithms of the Message Format Validation Rules (MFVR) specifications.

For lists of these Collaboration Definitions, see "Message Validation Rules" on page 27.

Validation Operation

You can combine the library's validation features in any way to meet your business requirements. The SWIFT Message Library packages a prebuilt implementation that takes SWIFT messages from a JMS Queue or Topic and validates them individually, then writes the results to a specified JMS Queue or Topic. One set contains valid messages, and the other contains the invalid ones, along with messages indicating the errors generated.

Basic validation steps

Each validation Collaboration Definition has only the applicable tests for a specific OTD or message type, but they all operate according to the same general format, as follows:

- The Service first tests a message to make sure it is syntactically correct by parsing it into the OTD.

- If the message fails, the message and its parser error are sent to an error Queue. If the message is valid, all applicable MFVR functions are applied to the message.
- Any and all errors produced from these tests are accumulated, and the combined errors, as well as the message, are written to an error Queue for later processing. As long as no error is fatal, all applicable tests are applied.
- Again any and all errors produced from these tests are accumulated, and the combined errors and message are written to the error Queue for later processing.
- If no errors are found in a message, it is sent to a Queue for valid messages.

For an explanation of using these Collaboration Definitions and the validation Project examples, see “SWIFT Projects” on page 34.

Library Methods

The SWIFT Message Library provides a set of runtime methods that allow you to manipulate OTD data in a variety of ways. The following methods are the most frequently used with validation operations:

- `set()`: Allows you to set data on a parent node using a byte array or a string as a parameter.
- `value()`: Lets you get the string value of data in a node at any tree level.
- `getLastSuccessInfo()`: Returns a string that represents information about the last node in the tree that was successfully parsed.
- `command()`: Allows you to pass flags as parameters, which set levels that determine the quantity of debug information you receive (see “Setting the Debug Level” on page 46 for details).
- `marshalToString()` and `unmarshalFromString()`: Returns string data from or accepts string data to a desired node.

In addition, the library has methods that allow you to perform basic but necessary operations with the OTDs. See Table 11.

TABLE 11 Basic OTD Methods

Method	Description
`add()`	Adds a repetition to a given child node.
`append()`	Adds given data at the end of existing data.
`copy()`	Copies given data at a specified point.
`count()`	Gives the count of node repetitions.
`delete()`	Erases data at a specified point.

TABLE 11 Basic OTD Methods *(Continued)*

Method	Description
`get()`	Retrieves data from a node.
`has()`	Checks whether a specified child node is present.
`insert()`	Inserts given data at a specified point.
`length()`	Returns the length of data contained in an object.
`marshal()`	Serializes internal data into an output stream.
`remove()`	Removes a given child node repetition.
`reset()`	Clears out any data held by an OTD.
`size()`	Returns the current number of repetitions for the current child node.
`unmarshal()`	Parses given input data into an internal data tree.

To help in your use of the SWIFT Message Library and its features, the library includes a Javadoc. You can see this document for complete details on all of these methods. See Table 12 for more information on this document and how to use it.

Message Validation Rules

Validation Collaborations are provided for specific SWIFT Message types and their corresponding OTDs in the library. For a complete list of validation Collaborations, see "Validation Collaborations" on page 21.

Message Format Validation Rules (MFVR)

The MFVR support for the SWIFT Message Library is a set of functions collectively known as the message format validation rules methods. These functions accurately test the semantic validity of a given subset of the SWIFT messages. Validation is performed according to standards provided in SWIFT's publication, the *Message Format Validation Rules Guide* (current version).

There is one validation method for each MFVR message type and its corresponding OTD. Each method is called on a particular OTD and is used to validate the data of a given instance of that message type. Because business practices vary greatly between organizations, these functions can be modified as needed.

For examples of how the MFVR validation process works, you can import the sample validation Projects. For details, see "SWIFT Projects" on page 34.

SWIFT's MFVR validation rules are known as semantic verification rules (SVRs) or semantic rules, as opposed to the syntactic rules, which verify the syntax of the fields only. Syntactic verification is built into each OTD.

SWIFT defines a total of 299 SVRs that are validated by the FIN network engine. SWIFT Alliance Access or IBM's Merva products do not implement these rules, mainly because there is no functional model, and the implementation work is mostly manual. Each message type has to be validated against a subset of these rules.

In addition this set of 299 SVRs, SWIFT has defined a new series of rules to help enable straight-through processing (STP) in the securities industry. The OTD methods that validate for MFVR compliance also validate for compliance with STP rules.

MFVR Validation Methods

The MFVR methods adhere to SWIFT's current *Message Format Validation Rules Guide*, including those in any updates section in the back of the manual. The methods implement all of the "special functions" as defined in the guide, which are required by the validation rules.

The SVR methods also implement the semantic validation functions used in the validation functions, as defined by the current *Message Format Validation Rules Guide*.

Using this semantic validation, Java CAPS can verify the contents of each message before it is sent into the FIN network, saving time and usage fees.

MFVR Errors

MFVR errors result from the application of the Semantic Validation Rules. Multiple errors are possible, and they are given in the order in which they occurred and with the sequences, fields, or subfields used to determine them.

For example, an MFVR failure on a 535 Collaboration OTD appears as follows:

```
MFVR MT535 Error
SVR Rule 103 - Error code: D031001 = Since field :94a:: is present
in Sequence B, then   fields :93B::AGGR and :94a::SAFE are not
allowed in any occurrence of Subsequence B1a.mt_535.Mt_535.Data[1].
SubSafekeepingAccount mt_535.Mt_535.Data[1].SubSafekeepingAccount[0].
SubSeqB1[0].SubSeqB1a.Balance

SVR Rule 104 - Error code: D04-1001 = Since field :93B::
AGGR is present in Subsequence B1a,then
:field 94a::SAFE must be present in the same Subsequence B1a.
mt_535.Mt_535.Data[1].SubSafekeepingAccount[0].SubSeqB1[0].
SubSeqB1a.Balance
```

For more information on error messages, see "Error Message Information" on page 46.

In Collaboration Validation Methods

As an alternative to using the Validation Collaborations, the SWIFT Message Library offers validation methods, `validate()`, `validateMPR()`, and `validateMFVR()`, that can be invoked by a Collaboration to validate SWIFT OTDs directly in the Collaboration. For example, if you have an OTD for message MT 541, you can call the OTD's `validateMFVR()` method from the Collaboration, and the Collaboration validates the message's MFVRs.

The validation methods are available for the same SWIFT message OTDs listed under "Message Validation Rules" on page 27. You can see (or select) these validation methods by right-clicking the SWIFT message OTD from the Collaboration Editor's Business Rules Designer and clicking **Select method to call** on the shortcut menu.

validate()

Description

This method validates applicable MFVR rules against the OTD instance, and throws a `MessageValidationException` if the OTD is invalid in regard to applicable MFVR rules. Call `MessageValidationException.getErrorMessage()` to obtain the error message details.

If the OTD does not have applicable MFVR rules, the method call returns without throwing a `MessageValidationException`.

Syntax

```
public void validate()
```

Parameters

None.

Return Values

None.

Throws

`com.stc.swift.validation.MessageValidationException`: Thrown when the OTD is invalid in regard to applicable MFVR rules.

validateMFVR()

Description

This method validates the applicable MFVR rules against the OTD instance, and throws a `MFVRException` if the OTD is invalid in regard to applicable MFVR rules. Call `MFVRException.getErrorMessage()` to obtain the error message details.

If the OTD does not have applicable MFVR rules the method call always returns without throwing an `MFVRException`.

Syntax

```
public void validateMFVR()
```

Parameters

`None.`

Return Values

None.

Throws

`com.stc.swift.validation.MFVRException`: Thrown when the OTD is invalid in regard to applicable MFVR rules.

validateMPR()

Description

This method validates the applicable MPR rules against the OTD instance, and throws a `MPRException` if the OTD is invalid in regard to applicable MPR rules. Call `MPRException.getErrorMessage()` to obtain the error message details.

If the OTD does not have applicable MPR rules the method call always returns without throwing an `MPRException`.

Syntax

```
public void validateMPR()
```

Parameters

`None.`

Return Values

None.

Throws

`com.stc.swift.validation.MPRException`: Thrown when the OTD is invalid in regard to applicable MPR rules.

Calling the Validation Methods in your Collaboration

The validation methods are available at the OTD level, and can be called after the OTD is populated with its values. This usually occurs after a message is unmarshaled in the OTD. The following fragment of code demonstrates the use of the `validate` method within a Collaboration. In this example, `validate()` is called and either "message OK" or the exception error String is written to the log.

```
import com.stc.swift.validation.MFVRException;
import com.stc.swift.validation.SVRException;
import com.stc.swift.validation.ValidatingSWIFTMTOTD;
import com.stc.swift.validation.bic.BICDir;
import com.stc.swift.validation.BICPlusIBAN.*;
import com.stc.swift.validation.MessageValidationException;
import com.stc.swift.otd.v2010.std.mt_541.Mt_541;
import java.util.*;


public class ValidateMt_541_Modified
{

    public boolean receive( com.stc.connectors.jms.Message input,
xsd.ValidationReplyMessage.Result output, com.stc.connectors.jms.JMS
 invalidMessages, com.stc.connectors.jms.JMS validMessages,
 com.stc.swift.otd.v2010.std.mt_541.Mt_541 mt_541_1 )
        throws Throwable
    {
        com.stc.connectors.jms.Message result = validMessages.createMessage();
        result.setTextMessage( input.getTextMessage() );
        String errors = null;
        String msg = "";
        try {
            mt_541_1.unmarshal( (com.stc.otd.runtime.OtdInputStream)
 new com.stc.otd.runtime.provider.SimpleOtdInputStreamImpl(
 new java.io.ByteArrayInputStream( input.getTextMessage().getBytes() ) ) );
        } catch ( Exception ex ) {
            errors = ex.getMessage();
            errors += "\r\n";
            errors += "Last successful parse: " + mt_541_1.getLastSuccessInfo();
            result.storeUserProperty( "ValidationErrors", errors );
            invalidMessages.send( result );
            output.setErrorMessages( errors );
            output.setIsError( true );
            output.setSwiftMessage( input.getTextMessage() );
            return false;
```

```
                }
                logger.info( "Unmarshalled MT541 message." );
                logger.info( "MFVR validation to follow ....." );
                // Call Default Validation logic for validation against applicable MFVRs
                try {
                    mt_541_1.validate();
                } catch ( MessageValidationException mve ) {
                    errors = mve.getErrorMessage();
                    msg = mve.getMessage();
                }
                logger.info( "Completed MFVR validation" );
                logger.info( "BICPlusIBAN validation to follow ....." );
                if (errors == null) {
                    logger.info( "No MFVR Exception" );
                } else {
                    logger.info( "Found MFVR Exception" );
                    logger.info( "Errors: " + errors );
                    logger.info( "msg: " + msg );
                }
                // End of "Default Validation" invoking
                //
                // Call BICPlusIBAN validation
                String BICPlusIBANresult = "";
                String bicCode = mt_541_1.getBasicHeader().getLTAddress().substring( 0, 8 );
                String ibanCode = "DE615088005034573201";
                BICPlusIBANDir.setBIC_Code( bicCode );
                BICPlusIBANDir.setIBAN_Code( ibanCode );
                BICPlusIBANresult = "\n\n\n*** Validating BICPlusIBAN ***\n";
                BICPlusIBANresult = BICPlusIBANresult +
"    BIC - " + BICPlusIBANDir.getBIC_code() + "\n";
                BICPlusIBANresult = BICPlusIBANresult +
"    IBAN - " + BICPlusIBANDir.getIBAN_code() + "\n";
                BICPlusIBANresult = BICPlusIBANresult +
"\n    a) Deriving the BIC from the IBAN...\n";
                ArrayList bicList = BICPlusIBANDir.deriveBICfromIBAN();
                if (bicList == null) {
                    BICPlusIBANresult = BICPlusIBANresult +
"      ==> Unable to derive BIC data from given IBAN.\n";
                    if (errors != null) {
                        errors = errors + "\n\nUnable to derive BIC data from given IBAN.\n";
                    } else {
                        errors = errors + "\n\nUnable to derive BIC data from given IBAN.\n";
                    }
                } else {
                    BICPlusIBANresult = BICPlusIBANresult +
"      ==> BIC CODE and BRANCH CODE = " + (String) bicList.get( 0 ) + ".\n";
                    BICPlusIBANresult = BICPlusIBANresult +
"      ==> IBAN BIC CODE and BRANCH CODE = " + (String) bicList.get( 1 ) + ".\n";
                    BICPlusIBANresult = BICPlusIBANresult +
"      ==> ROUTING BIC CODE and BRANCH CODE = " + (String) bicList.get( 2 ) + ".\n";
                }
                BICPlusIBANresult = BICPlusIBANresult + "\n   b) Validating the Bank ID...\n";
                if (BICPlusIBANDir.validateBankID()) {
                    BICPlusIBANresult = BICPlusIBANresult +
"      ==> Valid Bank ID found in BI file.\n";
                } else {
                    BICPlusIBANresult = BICPlusIBANresult +
"      ==> No valid Bank ID found in BI file.\n";
                    if (errors != null) {
```

```
                errors = errors + "No valid Bank ID found in BI file.\n";
            } else {
                errors = errors + "No valid Bank ID found in BI file.\n";
            }
        }
        BICPlusIBANresult = BICPlusIBANresult + "\n   c) Validating the BIC...\n";
        if (BICPlusIBANDir.validateBIC()) {
            BICPlusIBANresult = BICPlusIBANresult +
"      ==> Valid BIC data found in BI file.\n";
        } else {
            BICPlusIBANresult = BICPlusIBANresult +
"      ==> No valid BIC data found in BI file.\n";
            if (errors != null) {
                errors = errors + "No valid BIC data found in BI file.\n";
            } else {
                errors = errors + "No valid BIC data found in BI file.\n";
            }
        }
        BICPlusIBANresult = BICPlusIBANresult +
"\n   d) Validating the BIC/IBAN Combination...\n";
        if (BICPlusIBANDir.validateBICIBANCombo()) {
            BICPlusIBANresult = BICPlusIBANresult +
"      ==> BIC and IBAN codes are belong to the same institution.\n\n\n";
        } else {
            BICPlusIBANresult = BICPlusIBANresult +
"      ==> BIC and IBAN codes are NOT belong to the same institution.\n\n\n";
            if (errors != null) {
                errors = errors + "BIC and IBAN codes are NOT belong
 to the same institution.\n\n\n";
            } else {
                errors = errors + "BIC and IBAN codes are NOT belong
 to the same institution.\n\n\n";
            }
        }
        logger.info( BICPlusIBANresult );
        //
        if (errors != null) {
            // errors = errors + BICPlusIBANresult;
            result.storeUserProperty( "ValidationErrors", errors );
            invalidMessages.send( result );
            output.setErrorMessages( errors );
            output.setIsError( true );
            output.setSwiftMessage( input.getTextMessage() );
            return false;
        }
        // passed validation
        String currMsg = result.getTextMessage();
        currMsg = currMsg + BICPlusIBANresult;
        result.setTextMessage( currMsg );
        validMessages.send( result );
        output.setErrorMessages( "" );
        output.setIsError( false );
        output.setSwiftMessage( input.getTextMessage() );
        return true;
    }
```

To select a validation method from the Collaboration Editor's Business Rules Designer, right-click the SWIFT message OTD and click **Select method to call** from the shortcut menu.

SWIFT Projects

Two sample projects, Swift_SAA_XMLv2_MQHA and SAG610_FTA_Sample.zip, are provided with the SWIFT Message Library on the installation media. Additional samples can be found on the Java CAPS samples site at `http://java.net/projects/javacaps-samples/pages/Home`. You can import these projects directly into Java CAPS using the project import utility.

Note – You need to register with Oracle to access this site.

Importing a Sample Project

The sample projects on the installation media can be imported directly into NetBeans. The sample projects from the Java CAPS sample site must be unzipped before you can import them.

▼ To Import a Sample Project

1. **Do one of the following:**

   - **To use the samples supplied with the SWIFT Message Library, copy the samples from the installation media to a local directory.**

     The sample file are located in `components/message_libraries/Swift/samples`.

   - **To use the samples provided on the sample site, navigate to the sample site at `http://java.net/projects/javacaps-samples/pages/Home`, click Application Adapters, and download the `swift_otd.zip` sample file. Extract the file to a local directory.**

2. **Save all unsaved work in NetBeans before importing a Project.**

3. **On the NetBeans toolbar, select Tools, point to CAPS Repository, and then select Import.**

   The Import Project dialog box appears.

4. **Click Yes to continue the process, or click No to save your changes and repeat the previous step.**

   The Import Manager appears.

5. **Browse to the directory that contains the sample Project zip file. Select the sample file and click Import.**

6. **After the sample Project is successfully imported, click Close.**

7. **Before an imported sample Project can be run, you need to do the following:**

- Create an Environment
- Create a Deployment Profile
- Create and start a domain
- Build and deploy the Project

SWIFT Projects and NetBeans

A Project contains all of the Java CAPS components that you designate to perform one or more desired processes.

- **Connectivity Map Editor**: Contains the business logic components, such as Collaborations, Topics, Queues, and Adapters, that you include in the structure of the Project.
- **OTD Editor**: Contains the source files used to create Object Type Definitions (OTDs) to use with a Project.
- **Business Process Designer and Editor**: Allows you to create and modify Business Rules to implement the business logic of your Project's business processes.
- **Java Collaboration Editor**: Allows you to create and modify business rules to implement the business logic of your Project's Java Collaboration Definitions.

About the SWIFT MX Validation Sample

The SWIFT MX Validation sample demonstrates what types of "Generic Validations" are done on MX messages and how they are applicable. The sample zip file on the Java CAPS sample site contains the following files and directories for SWIFT MX validation:

- Input Data — MXSample_input.xml.fin – Sample MX message to be read by inbound File Adapter.
- jcdSchemaValidation.java – Java collaboration to validate MX messages against relevant XSD schema.
- jcdGenericValidation.java – Java collaboration to validate MX messages against Extended Validation Rules.
- Output Data – MX_GenericValidationLog_output1.dat – This is a log file for validation results from the jcdGenericValidation java collaboration.
- Sample Project – SwiftMXSample.zip: This is the sample project.
- XSD Data — XSD Schema file () to be validated against the MX input message: RedemptionBuldOrderV02.xsd and swift.if.ia_setr.001.001.02.xsd.

Note – The Batch Adapter is required when running the SWIFT MX Validation sample.

Sample Project

The Project's flow is represented in the Connectivity Map as follows:

```
Inbound File Adapter —> Schema Validation —> JMS Queue —> Generic Validation —> Batch Adapter, Outbound File Adapter
```

These are explained further below.

Descriptions of components

- Inbound File Adapter – The File Adapter is used to read MX messages to be validated.
- Schema Validation – Each MX message has a corresponding XSD Schema file. You must use the XSD OTD Wizard to build an XSD OTD based on the schema file. In this collaboration, the logic is to unmarshal the inbound message to the XSD OTD and then to marshal the OTD to String and send the payload to JMS Queue. This process is to ensure the MX message is well-formed and is validated against the XSD schema. For a different MX message type, build the XSD OTD and create this simple collaboration.

  Note – The sample project chooses the "RedemptionBulkOrderV02" message and schema to demo the usage. The RedemptionBulkOrderV02 schema is obtained from the SWIFTNet Funds ver3.0 CD, which also contains all element types to be validated in the Generic Validation collaboration.

- JMS Queue – JMS Queue to hold schema validated messages.
- Generic Validation Collaboration – This collaboration contains a set of generic validation rules, which SWIFT recommends must be applied to an MX message. You can reuse this collaboration to validate all MX Message types. The generic validation rules validate the following identifiers and codes in a MX message:
  - Verifying BIC (datatype: BICIdentifier), against existence in the BIC directory (ISO 9362)
  - Verifying BEI (datatype: BEIIdentifier), against existence in the BEI list on SWIFTNet
  - Verifying ActiveCurrencyAndAmount (datatype: ActiveCurrencyAndAmount), against existence in Currency Code and number of valid decimal digits (ISO 4217)
  - Verifying Country Code (datatype: CountryCode), against existence in Country Code list (ISO 3166)
  - Verifying IBAN Identifier (datatype: IBANIdentifier), against IBAN structure as provided by ISO 13616
  - Verifying BICOrBEI (datatype: AnyBICIdentifier), against existence in the BIC list on SWIFTNet
  - Verifying ActiveCurrency (datatype: ActiveCurrencyCode), against existence in Currency Code list on SWIFTNet

- Verifying ActiveOrHistoricCurrency (datatype: ActiveOrHistoricCurrencyCode), against existence in Currency Code list on SWIFTNet
- Batch Adapter – The Batch (Local File) Adapter is used to read XSD files for Generic Validation. Place all XSD schema files in one directory and make sure the name of the XSD file matches the target namespace specified in the MX message. For example, in the sample input file, there is:

  xmlns:Doc=”urn:swift:xsd:swift.if.ia$setr.001.001.02

  Therefore the matching schema file name must be `swift.if.ia_setr.001.001.02`. Please rename the **$** character to _, because the **$** character is not considered a valid file name pattern in Java.

  In Java CAPS, you must open the Batch Adapter configuration in the connectivity map (and under the Target Location node) and make sure the directory name for the XSD files are set to Target Directory Name field.
- Outbound File Adapter – The File Adapter is used to log validation results and error messages in Generic Validation.

Note – You can place all XSD schema files in one directory. In Connectivity Map, set the directory name in the Target Directory Name, under Target Location section in Batch Local File configuration window. In Generic Validation, the collaboration will read the input message and locate the associated schema file name, in the directory name specified in Batch Adapter. Make sure the schema file name does not contain any illegal character **$**. This **$** character should be replaced with _ character in file name. For example, schema file name `swift.if.ia$setr.001.001.02` should be renamed to `swift.if.ia_setr.001.001.02`and placed in the target directory.

Running the MX Sample Project

To run the MX Sample Project, complete the following steps.

1. Import the SWIFT Message Library SAR file.
2. Import the sample project.
3. In the NetBeans Projects window, under CAPS Components Library > Message Library > Swift, right-click on `bic.jar` and update CT, CU, and FI bic data files.
4. In the Connectivity Map, make sure the directory name and the file name in both the File Adapter and Batch Adapter are valid.
5. On the NetBeans Services window, create a new Environment for the project.
6. Under the project, create a new Deployment Profile and map all components.
7. Build and Deploy the project.
8. Send the input file to the inbound File Adapter and watch for the outbound file.

Note – You must build your own XSD OTD and Schema Validation collaboration, based on different MX message types to be validated. You can always reuse the Generic Validation collaboration for all MX messages.

SWIFT Correlation Repository Sample

The SWIFT Correlation Repository (SCR) is a Java CAPS utility used to visualize SWIFT workflows. In addition, the SCR does the following:

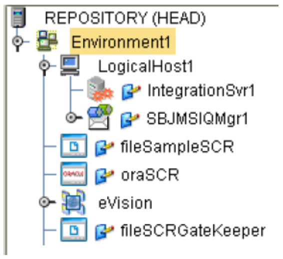
- Reconciles Messages into Transaction Processes
- Provides Message Browsing and Message Monitoring
- Offers services for Duplicate Checking and validation of MX and MT messages
- Allows for Message Repair and Resubmit

Prerequisites

You must have an Oracle database, version 9i or greater to run the project.

Installation steps

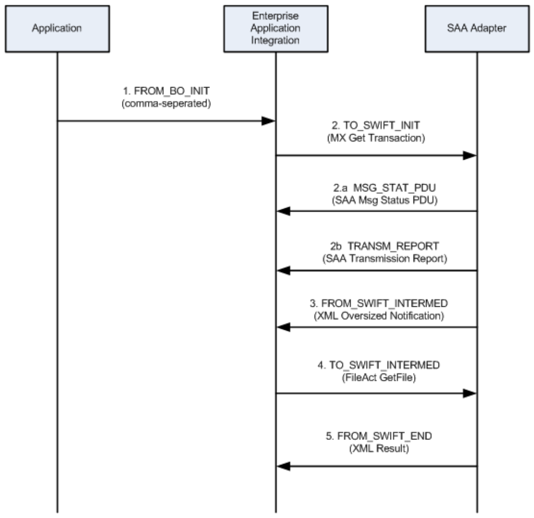
1. Install the database schema from the `SCR_CreateUser.sql` and `SCR_CreateTable.sql` files (located in the `SCR_Create_Cleanup.zip` file).
2. Extract the contents of the `SampleSCR.zip` file into your local drive.
3. Import the `SCRProject.zip` file.
4. Set the environment variables (as shown in the figure below).



5. Create a deployment profile in the SCR project.
6. Create a deployment profile in the TesterGatekeeper project.
7. Deploy both the SCR and TesterGatekeeper deployment profiles.

Preparing an SCR flow

The SCR Workflow follows the tasks, procedural steps, required input and output information, for each step in the business process. The SCR workflow is used to view, manage, and enforce the consistent handling of work. The following figure is an example of a design of an SCR flow.


Application
Enterprise Application Integration
SAA Adapter
1. FROM_BO_INIT (comma-seperated)
2. TO_SWIFT_INIT (MX Get Transaction)
2.a MSG_STAT_PDU (SAA Msg Status PDU)
2b TRANSM_REPORT (SAA Transmission Report)
3. FROM_SWIFT_INTERMED (XML Oversized Notification)
4. TO_SWIFT_INTERMED (FileAct GetFile)
5. FROM_SWIFT_END (XML Result)

Designing an SCR flow

1. Start NetBeans.
2. Open the imported SCR project.
3. Choose both a short name and a long name for the flow (example: `t2 :: Target2`).
4. Choose a string name for each event / message / direction (as shown in the SCR flow example above: `TO_SWIFT_INIT`).
5. Add the flow name as a new choice in the viewer by navigating to the Viewer on the SCR page, then to the 1TrxList, and then to the pgTrxList.
   a. On the Properties tab, select `SelDomain`.
   b. Right-click the highlighted area on the design canvas, and select Edit Options. The Edit Options window opens.
   c. Add new flow elements to the properties of the control SelDomain. This project already has default values entered (`t2 :: Target2`).

Linking the Domain Name and Direction to a Color

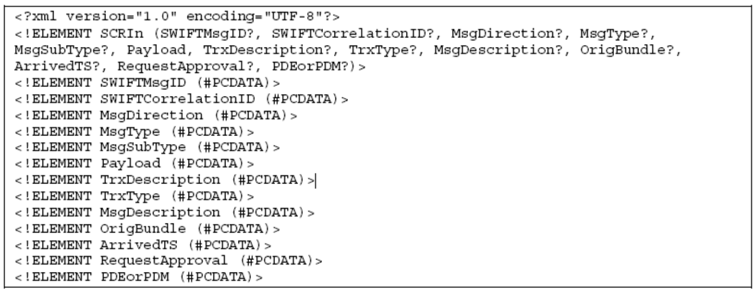
You can link the name of a Domain to specific pointer directions and colors within the monitoring application.

1. Extract the `SampleSCR.zip` file included with the sample project.
2. Link the domain name and the direction to a color by opening the `SCR.properties` file located in the directory where you extracted the ZIP file.
3. A list of available directions and colors are listed in the SCR properties file. Possible Colored Directions (CD) for message lists include:
   - DEFAULT
   - LGREY, RGREY
   - LBLUE, RBLUE
   - LGREEN, RGREEN
   - LORANGE, RORANGE
   - LRED, RRED
4. Link the Domain to a specific pointer direction and color by using the following Syntax: `CD_<Direction String> = <Colored Directions>.`

Using the SCR for Monitoring Flows

Applications that send events to the SCR must do two things:

1. Create a message following the input format shown below. Do not use the field whose usage is indicated as "Gatekeeper only".



```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT SCRIn (SWIFTMsgID?, SWIFTCorrelationID?, MsgDirection?, MsgType?,
MsgSubType?, Payload, TrxDescription?, TrxType?, MsgDescription?, OrigBundle?,
ArrivedTS?, RequestApproval?, PDEorPDM?)>
<!ELEMENT SWIFTMsgID (#PCDATA)>
<!ELEMENT SWIFTCorrelationID (#PCDATA)>
<!ELEMENT MsgDirection (#PCDATA)>
<!ELEMENT MsgType (#PCDATA)>
<!ELEMENT MsgSubType (#PCDATA)>
<!ELEMENT Payload (#PCDATA)>
<!ELEMENT TrxDescription (#PCDATA)>
<!ELEMENT TrxType (#PCDATA)>
<!ELEMENT MsgDescription (#PCDATA)>
<!ELEMENT OrigBundle (#PCDATA)>
<!ELEMENT ArrivedTS (#PCDATA)>
<!ELEMENT RequestApproval (#PCDATA)>
<!ELEMENT PDEorPDM (#PCDATA)>
```

2. Send the message to either:
   - A file in the `\SampleSCR\In` location, with a `.txt` extension and a name starting with `Loader`.
   - A JMS message to the JMS queue, qSCRInEnv, in SCR/Loader.

Using the Viewer for Monitoring Transactions

1. Use an Internet browser and navigate to the URL `http://localhost:8080/scr`. The Select Transaction window opens.
2. Use one of the following criteria for monitoring transactions:
   - Select the 10 most recently updated transactions from the drop-down list.
   - Use the domain selector to restrict the transaction list.
   - Search for a transaction with a specific ID.
   - Search for a transaction that contains a message with a specific ID.
3. Click the Search button.

Using the SCR as Gatekeeper

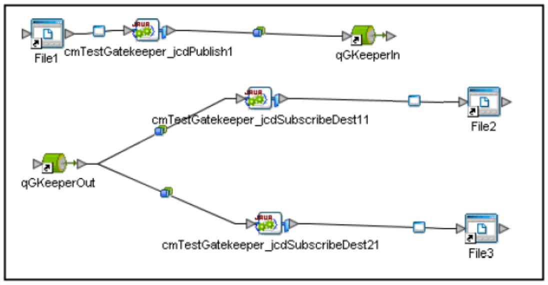
Applications sending events to the SCR as Gatekeeper must do two things:

1. Create a message following the input format (as shown in the previous section)
2. Send the message to the JMS queue “qGKeeperIn” in SCR/Gatekeeper. Make sure to add a JMS topic to the message. A code sample is shown below.

```
com.stc.connectors.jms.Message outMsg =
jmsPublish.createTextMessage();
outMsg.storeUserProperty( "SCRDestination", "DEST1" );
outMsg.setTextMessage( input.getText() );
jmsPublish.send( outMsg );
```

3. Subscribe to the JMS queue ”qGKeeperOut” in SCR/Gatekeeper.
4. Subscribe to the JMS topic that you used to publish the message.

Note – A complete test setup is located in the project `TesterGatekeeper`.


File1
cmTestGatekeeper_jcdPublish1
qGKeeperIn
cmTestGatekeeper_jcdSubscribeDest11
File2
qGKeeperOut
cmTestGatekeeper_jcdSubscribeDest21
File3

Using the Viewer to Repair Messages

1. Use an Internet browser and navigate to the URL `http://localhost:8080/scr`. The Select Transaction window opens.

2. Select the 10 most recently updated transactions from the drop-down list. Messages that have been held for review and resubmitting (e.g. messages that are duplicates, incorrect, or awaiting approval) are displayed.
3. Select the message you wish to examine and click the Repair button. The Message Repair window opens, displaying detailed information regarding the message.
4. You can resolve the message in the following ways:
   - Correct the message error and click the **Resubmit** button.
   - Examine a message that requires approval and click the **Approve** button.
   - Delete the message by clicking the **Delete** button

Updating BICDirService

The BICDirService feature is a database service. The data files used to populate BICDirService must be updated periodically from SWIFT's source CD-ROM issued once every four months.

Source of Information

The Java constructor for the `BICDir` class loads the required data from the following SWIFT-supplied files:

- `FI.dat`
- `CU.dat`
- `CT.dat`

The constructor takes an argument from the directory that contains these two files. It then opens each file and loads the appropriate fields into a searchable structure. For more details on these files, see the current SWIFT *BIC Database Plus Technical Specifications* document for actual file layout and positioning information.

The data used to look up and validate comes from SWIFT's own BIC bank files containing its BIC codes and its currency and country codes. When necessary, SWIFT updates these files with a new version of its lookup tables, to keep them current. You can upload these files to Java CAPS and control when updates to the system occur and access these files via SWIFT updates.

Update Operation

The BICDirService feature allows multiple simultaneous objects to access its methods with near-local object response times. The SWIFT standards are not always sufficiently complete to enable STP. Currently a message can pass network validation but fail at the receiving end because of incompatible definitions or codes, or because of missing data. The result is having to manually repair or follow up on these messages and possible retransmission of the message.

The SWIFT Message Library's BICDirService ensures that valid, up-to-date BIC, country, and currency codes are present in the messages processed through Java CAPS. This feature increases the likelihood that a given message can flow "straight through".

You must update the BICDirService information before running components that utilize this feature. This procedure updates the BICDirService information.

▼ To Update BIC Information

1. **In the NetBeans Projects window, expand CAPS Components Library > Message Library > Swift.**
2. **Right-click the `bic.jar` file, point to Version Control, and select Check Out. Click Check Out on the dialog box that appears.**
3. **Right-click the `bic.jar` file again, and select Update BIC Files.**

   A browser window appears so you can browse to and select the files to update.
4. **Navigate to the location of the `CU.dat` file on the SWIFT update CD-ROM. select the file, and click Open.**
5. **Repeat the above two steps to update the `FI.dat` and `CT.dat` files.**

BICDirService Method Operation

The BICDirService methods are static methods of a single Java class, the `BICDir` class. There is one method per each required lookup and validation. The `BICDir` methods are not dependent on any module other than SWIFT data files.

Lookup Method Definitions

The `BICDir` class has the following lookup methods:

- **Look up BIC by Institution Name:** Takes a string and returns a byte array of BICs (one element is possible). The signature is:

  ```
  BIC[] getBIC(institutionName*);
  ```
- **Look up BIC by Institution Name, City and Country:** Takes three strings, an institution name, city, and country, and returns a byte array of BICs (one element is possible). The signature is:

  ```
  BIC[] getBIC(institutionName*, city*, country*);
  ```
- **Look up Institution Name by BIC:** Takes a BIC string, either a BIC 8 or BIC11, and returns a byte array of institution names (one element is possible). The signature is:

  ```
  institutionName[] getInstitutionName(BIC);
  ```

- **Look up Currency Code by Country Code:** Takes a string, a country code, and returns the currency code. The signature is:

  ```
  currencyCode getCurrencyCode(countryCode);
  ```
- **Look up Country Code by Currency Code:** Takes a string, a currency code, and returns the country code. The signature is:

  ```
  countryCode getCountryCode(currencyCode);
  ```

Validation Method Definitions

The `BICDir` class has the following validation methods:

- **Validate BIC:** Takes a string, either a BIC 8 or BIC11, and returns true or false. The signature is:

  ```
  boolean validateBIC(BIC);
  ```
- **Validate Currency Code:** Takes a string, a currency code, and returns true or false. The signature is:

  ```
  boolean validateCurrencyCode(currencyCode);
  ```
- **Validate Country Code:** Takes a string, a country code, and returns true or false. The signature is:

  ```
  boolean validateCountryCode(countryCode);
  ```

BICDir Exceptions

The purpose of the exceptions is to give you some indication of what error has occurred and how to rectify it.

Error message framework

These error messages are implemented using the log4j framework. **STC.OTD.SWIFT.BICDirService** is used as the logging category.

Error Message General Form

The message of `BICDir` exception takes the following general form:

```
“BICDirService Error [”XX“]— “ error-message
```

Where:

- **“”**: Marks static text.
- **XX**: Stands for a unique number assigned to each error message.
- **error-message**: A descriptive narrative derived from the condition that caused the error, and a possible solution to rectify it.

Updating BICPlusIBAN

The data files used to populate BICPlusIBAN directory must be obtained from SWIFT directly. The sample BICPlusIBAN directory from SWIFT Message Library is only the test data files to be used with the sample project. They are not intended to be used in a production environment.

The Java constructor for the BICPlusIBAN class loads the required data from the following SWIFT-supplied files:

- `BI.TXT`
- `IS.TXT`

The constructor takes an argument from the directory that contains these two files. It then opens each file and loads the appropriate fields into a searchable structure. For more details on these files, see the current SWIFT BICPlusIBAN Directory Technical Specifications document for actual file layout and positioning information.

The BI data contains the BICPlusIBAN information. The IS data provides IBAN structure information. The SWIFT Message Library takes these data together to execute the validation rules. These base files and update delta files should be obtained directly from SWIFT.

▼ To Update BICPlusBAN Information

1. **In the NetBeans Projects window, expand CAPS Components Library > Message Library > Swift.**
2. **Right-click the `BICPlusIBAN.jar` file, point to Version Control, and select Check Out. Click Check Out on the dialog box that appears.**
3. **Right-click the `BICPlusIBAN.jar` file again, and select Update BICPlusIBAN Files.**

   A browser window appears so you can browse to and select the files to update.
4. **Navigate to the location of the `BI.txt` file on the SWIFT update CD-ROM. select the file, and click Open.**
5. **Repeat the above two steps to update the `IS.txt` file.**

BICPlusIBAN Validation Method Definitions

The SWIFT Message Library provides the following validation methods for BICPlusIBAN:

- **Deriving the BIC from the IBAN**: This validation method is used to derive the BIC from the IBAN. This can be useful in situations where the IBAN is present but the BIC is missing in a SEPA payment instruction. The method takes no arguments, and will return an array list of BIC code and BRANCH code. The signature is:

```
ArrayList deriveBICfromIBAN()
```

- **Validating the Bank ID**: This validation method is used to validate that the Bank ID contained in an IBAN is a valid Bank ID. This can be useful in situations where the ordering customer has constructed the IBAN. However, the validation does not guarantee that the IBAN itself is valid. The method takes no arguments, and will return a boolean result. The signature is:

  ```
  boolean validateBankID()
  ```

- **Validating the BIC**: This validation method is used to validate that the BIC is a valid BIC. This can for example be useful in situations when the ordering customer attempted to derive the BIC itself from financial institution's name and address. The method takes no arguments, and will return a boolean result. The signature is:

  ```
  boolean validateBIC()
  ```

- **Validating the BIC/IBAN combination**: This validation method is used to validate that the BIC and the IBAN belong to one and the same institution. The method takes no arguments, and will return a boolean result. The signature is:

  ```
  boolean validateBICIBANCombo()
  ```

Error Message Information

This section explains the SWIFT Message Library validation error files and messages.

Error Messages

There are separate error messages and reporting mechanisms for each type of validation performed by a Service. You can control the amount of debugging information in the error messages you receive by using the debug flags as parameters when you call the `command()` method. The library's error parser provides the following debug levels:

- **Regular Information**: Gives general information, and if an error occurs, the path to the node or piece of data that caused the error.
- **Debug**: Gives all of the node information generated by the parse, that is, each field and subfield.
- **Parser Debug**: Combines the debug level with information regarding just what the parser is matching, and the data being used. In general, you only need to use this level for situations where the error cannot be determined using the other levels because of the quantity of data. This level gives the exact location and nature of the failure.

Error message file output appears at the end of any message that generates an error.

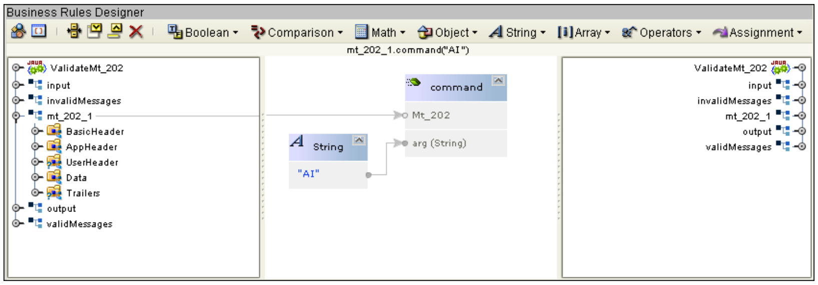
Setting the Debug Level

The available debug level flags are:

- **A** or **a**: Enables the abbreviation of path names. This reduces the path output when you are printing to a Regular Information set.
- **D** or **d**: Enables Debug (mid-level) debugging. If enabled, this generates more debug data than the Regular Information level, but less than the Parser Debug level.
- **I** or **i**: Enables Regular Information level debugging.
- **L** or **l**: Enables saving and display of the last successfully parsed node. When a parse has failed, this information is the last item printed by the current root node.
- **P** or **p**: Enables the Parser Debug-level information. If enabled, this generates the maximum information about what the internal parser is doing.

Using the Debug Level flags, you can configure the debugging information you receive by setting the appropriate debug parameter in the OTD's `command()` method. For example, to set the error message level to the Regular Information level (I flag), with abbreviations turned on (A flag), you would set `command()` with the parameters A and I. You can do this from the Collaboration Editor's Business Rules Designer as displayed below.

FIGURE 1 Setting the debug level using the Business Rules Designer



This produces the following Java code (this example uses the mt_202 Validation Collaboration:

```
mt_202_1.command( "AI" );
```

Calling `command()` enables any of the debug functions presented as a parameter. For more information, see the SWIFT Message Library Javadoc.

Message Examples

An example of a regular information-level parse error (cannot find a required field) is:

```
at 0: com.stc.swift.runtime.SwiftUnmarshalException: mt_103.Mt_103: 0:
 Failed to parse required child(Data).
```

An example of a parse error with the debug level enabled (cannot find a required field) is:

```
at 146: null: com.stc.swift.runtime.SwiftUnmarshalException:
 mt_543.Mt_543.Data.GeneralInformation.FunctionOfTheMessage: 146:
 Failed to parse required child(String2).
```

Given this path to the data, you can determine where in the message the parser failed by looking at:

- The *SWIFT User Handbook*
- The structure of the OTD in the NetBeans OTD Editor
- The Javadoc for the OTD

See "MFVR Errors" on page 28 for MFVR-specific error information. For more detailed error information, see "Error Message Information" on page 46.

Parse Debug Level Message Example

The following example shows error message output at the parse debug level:

```
[main] PARSE - Swift: matchDelimSkip("{1:") --> true.
[main] PARSE - Swift: getData("F|A|L") --> "F".
[main] DEBUG - Swift: mt_502.Mt_502.BasicHeader.AppIdentifier: 3: Mapped data("F").
[main] DEBUG - Swift: mt_502.Mt_502.BasicHeader.AppIdentifier: 3: Mapped rep[0].
[main] PARSE - Swift: getData(charSet, 2, 2) --> "01".
[main] DEBUG - Swift: mt_502.Mt_502.BasicHeader.ServiceIdentifier: 4:
 The following is the last field successfully parsed the 4th 22a:
[main] PARSE - Swift: matchDelimSkip("22H::") --> true.
[main] PARSE - Swift: getData(charSet, 4, 4) --> "PAYM".
[main] DEBUG - Swift: mt_502.Mt_502.Data.OrderDetails.Indicator.IndicatorH.String3:
 218: Mapped data("PAYM").
[main] PARSE - Swift: matchDelimSkip("//") --> true.
[main] DEBUG - Swift: mt_502.Mt_502.Data.OrderDetails.Indicator.IndicatorH.String3:
 218: Mapped rep[0].
[main] PARSE - Swift: getData(charSet, 4, 4) --> "APMT".
[main] DEBUG - Swift: mt_502.Mt_502.Data.OrderDetails.Indicator.IndicatorH.String5:
 224: Mapped data("APMT").
[main] DEBUG - Swift: mt_502.Mt_502.Data.OrderDetails.Indicator.IndicatorH.String5:
 224: Mapped rep[0].
[main] PARSE - Swift: matchDelimSkip("
:") --> true.
[main] DEBUG - Swift: mt_502.Mt_502.Data.OrderDetails.Indicator.IndicatorH: 213:
 Mapped rep[0].
```

The message goes on for several more lines, not indicating any error. Then the parser is looking for any more 22a's, F or H, and does not find one. See the following example:

```
[main] DEBUG - Swift: mt_502.Mt_502.Data.OrderDetails.Indicator[3]: 159: Mapped rep[3].
[main] PARSE - Swift: matchDelimSkip("22F::") --> false.
[main] PARSE - Swift: mt_502.Mt_502.Data.OrderDetails.Indicator.IndicatorF: 231:
 Failed to find BeginDelimiter("22F::").
[main] PARSE - Swift: matchDelimSkip("22H::") --> false.
[main] PARSE - Swift: mt_502.Mt_502.Data.OrderDetails.Indicator.IndicatorH: 231:
 Failed to find BeginDelimiter("22H::").
```

The parser then looks for a 98a either option A|B|C as follows:

```
[main] PARSE - Swift: matchDelimSkip("98A::") --> false.
[main] PARSE - Swift: mt_502.Mt_502.Data.OrderDetails.DateTime[0].DateTimeA: 231:
 Failed to find BeginDelimiter("98A::").
[main] PARSE - Swift: matchDelimSkip("98B::") --> false.
[main] PARSE - Swift: mt_502.Mt_502.Data.OrderDetails.DateTime[0].DateTimeB: 231:
 Failed to find BeginDelimiter("98B::").
[main] PARSE - Swift: matchDelimSkip("98C::") --> false.
[main] PARSE - Swift: mt_502.Mt_502.Data.OrderDetails.DateTime[0].DateTimeC: 231:
 Failed to find BeginDelimiter("98C::").
```

The parser finds no repetitions, which does not fit in the required range of 1 to 3 as described in the following example, so at this point, the parser fails, because no expected repetitions were found:

```
[main] PARSE - Swift: mt_502.Mt_502.Data.OrderDetails: 231:
 Failed to match minimum repititions[ 1 < 0 <= 3 ].

[main] PARSE - Swift: mt_502.Mt_502.Data.OrderDetails:
145: Failed to parse
 required child(DateTime).
[main] PARSE - Swift: mt_502.Mt_502.Data:
145: Failed to match minimum
 repititions[ 1 < 0 <= 1 ].
[main] PARSE - Swift: mt_502.Mt_502.Data:
73: Failed to parse required
 child(OrderDetails).
[main] PARSE - Swift: mt_502.Mt_502:
67: Failed to match minimum repititions[ 1 < 0 <= 1 ].
[main] PARSE - Swift: mt_502.Mt_502:
0: Failed to parse required child(Data).
 [main] LAST  - Swift: Last match: mt_502.Mt_502.
Exception in thread "main" at 0: null: com.stc.
swift.runtime.SwiftUnmarshalException:
 mt_502.Mt_502: 0: Failed to parse required child(Data).
    at com.stc.swift.runtime.SwiftOtdRep.
throwExcept(SwiftOtdRep.java:1977)
    at com.stc.swift.runtime.SwiftOtdRep.
parseChildren(SwiftOtdRep.java:1577)
    at com.stc.swift.runtime.SwiftOtdRep.
parse(SwiftOtdRep.java:1486)
    at com.stc.swift.runtime.SwiftOtdRep.
unmarshal(SwiftOtdRep.java:1339)
```

Using SWIFT FIN-Based Funds OTDs

This section explains how to use specialized funds features available with the SWIFT Message Library and Java CAPS.

- "SWIFT Message Library Funds Features" on page 50

SWIFT Message Library Funds Features

The SWIFT Message Library Object Type Definitions (OTDs) contain specialized OTDs that allow you to automate the following funds operations:

- Orders to buy and sell
- Client confirmations
- Checking order status
- Statement of holdings, for fund balances reconciliation

In the past, many funds industry players have asked SWIFT to help automate these operations by providing standards and connectivity between funds distributors, transfer agents, funds management companies, and other intermediaries like funds processing hubs. To meet these needs, SWIFT has developed standards and message templates based on these standards.

The SWIFT Message Library contains the following FIN-based MT Fund OTDs (see Table 12) specialized for the associated SWIFT message types and fund operations:

TABLE 12 FIN-based Funds OTDs

OTD Name	Base	Description
mt_502_FUNDS	FIN	**Order to buy and sell**: for funds subscription, redemption, switch, and cancellation.
mt_509_FUNDS	FIN	**Order status**: for status update on orders (for example, a rejection or acknowledgement of a receipt).
mt_515_FUNDS	FIN	**Client confirmation**: for confirmation of the funds subscription, redemption, switch and cancellation.
mt_535_FUNDS	FIN	**Statement of holdings**: for funds balance reconciliation.
mt_574_IRSLST	FIN	**IRS 1441 NRA**: IRS Beneficial Owners' List
mt_574_W8BENO	FIN	**IRS 1441 NRA**: Form W8-BEN

These MT Fund OTDs apply to the funds message types in the ISO 15022 FIN Standard. The Category 5 directory contains the SWIFT MT Funds message OTDs.

Using SWIFT Message Library Java Classes

This section provides an overview of the Java classes, interfaces, and methods contained in the SWIFT Message Library. These methods are used to extend the functionality of the library.

The SWIFT Message Library exposes various Java classes to add extra functionality to the library and its Object Type Definitions (OTDs). Some of these classes contain methods that allow you to set data in the library OTDs, as well as get data from them.

Relation to OTD Message Types

The nature of this data transfer depends on the available nodes and features in each of the individual SWIFT OTD message types. For more information on the SWIFT Message Library's messages and message types, see "SWIFT Message Type OTDs" on page 7.

SWIFT Message Library Javadoc

The SWIFT Message Library Javadoc is an API document in HTML format that provides information on the various classes and methods available with the SWIFT Message Library. You can access the Javadoc from the Java CAPS Documentation Library on the Oracle Technology Network (OTN) at `http://www.oracle.com/technetwork/indexes/documentation/index.html`. You can download the Javadoc or you can view in online.

OTD Library Java Classes

The Javadoc shows a Java class for each OTD in the SWIFT Message Library. For example, the class Mt_101 includes the OTD for the MT 101 SWIFT message type. See "SWIFT Message Type Reference" on page 8"SWIFT Message Type Reference" on page 8 for a complete list of the SWIFT message types and OTDs in the library.

In addition to the classes for OTDs, the following Java classes contain methods for runtime operation:

- **SwiftMarshalException**
- **SwiftOtdChild**
- **SwiftOtdInputStream**
- **SwiftOtdLocation**
- **SwiftOtdRep**
- **SwiftParseUtils**
- **SwiftUnmarshalException**