Developing Connectors with Telco Service Manager

V4.0
Document ID: TSGN-07-4.0-01
Date Published: 9.10.03

edocs, Inc., One Apple Hill Drive, Suite 301, Natick, MA 01760

Preface

In This Section

Using this Manual iv
Finding the Information You Need ix
If You Need Help xi

Using this Manual

Welcome to Developing Connectors.

This manual helps you understand the integration architecture and how your solution can work with today's market-leading OSS software.

Before You Get Started

You should be familiar with the following:

- Your application architecture
- Programming in Java
- Designing or working with databases
- eXtended Markup Language (XML)
- Your OSS

Who Should Read this Manual

This manual is for developers and project managers who are responsible for configuring and developing connectors which allow you solution to exchange data with OSS.

However, there are other topics covered in this manual that may interest other members of the project development team.

- Administrators

  You will find information about running and managing connectors in this manual. You will also find detailed information about how the solution exchanges information with backend systems. This will help you not only understand the application architecture, but help you quickly pinpoint problems that may occur when communication between your solution and the OSS breaks down.

- Developers

  You will find everything you need to know in order to build, generate and run connectors. This manual contains instructions on how to use the Integration Logic Studio you use to build connectors quickly and easily. It also shows you how to generate the runtime files then launch the connector. If your solution requires customized data structures, you also can read about using the Message Schema Reference. This part of the connector describes the structure of messages which carry the data exchanged between your solution and the back end.

- Project Architect

  You can use the information in this manual to learn about connectors and how they exchange information with the backend. You will find information about the way the connector works and how you can use them to integrate data from the backend. There is also a section covering integration scenarios.

- Project Manager

  You will find information about connectors and the different tools used to create and run them. You will find information about how your solution uses connectors and how they exchange messages. You may also find the section about the Message Schema Reference of interest as this section deals with the structure and customization of the messages used to exchange data with the backend.

How this Manual is Organized

This manual contains the following chapters:

- **Overview of Integrating Account Applications**

  This chapter covers the fundamentals of integrating an Account Management solution.

  It contains information about:

  - Essential concepts
  - Components
  - Integration strategies

- **Installing the SmartLink Framework**

  This chapter covers installing the SmartLink Framework.

  It contains information about installing:

  - SmartLink Framework components
  - Integration Logic Studio

- **Before You Start**

  This chapter covers describes what you need to know before you start building connectors.

  It contains information about:

  - Basic concepts of the SmartLink Framework
  - Connector components
  - Connector files

- Selecting the connector to build

- **Designing Your Connector**

  This chapter covers how to design your connector.

  It contains information about:

  - The basics of each connector type
  - How to create a connector from one of the templates.

- **Working With the Integration Logic Studio**

  This chapter covers using the Integration Logic Studio (ILS).

  It contains information about using the ILS to create and configure the components of a connector:

  - Integration processes
  - Processors
  - Scripts
  - Macros
  - Extensions
  - Parameters

- **Working With Connector Extensions**

  This chapter covers working with connector extensions in the ILS.

  It contains information about:

  - Creating custom components
  - Using custom components

- **Working With the Message Schema Reference**

  This chapter covers working with the Message Schema Reference.

  It contains information about:

  - Adding messages
  - Customizing messages
  - Working with the Schema Reference tool
  - Generating the Message Schema Reference files

- **Generating Your Connector**

  This chapter covers generating the runtime files of your connector.

  It contains information about:

  - Using the ILS to generate the runtime files
  - Using the command line tool to generate runtime files

- **Running Your Connector**

  This chapter covers running the connector.

  It contains information about:

  - Specifying processing priorities
  - Starting and stopping connectors
  - Administrating connectors

- **SmartLink Framework Tool Reference**

  This appendix is a tool reference guide. It covers the location, configuration and use of the administration tools.

  The administration tools covered include:

  - Synchronizer connector tools
  - OSS connector tools
  - Approval Sequencer tools

- **Core Messages**

  This appendix lists available core messages.

- **Core Business Objects**

  This appendix lists the Business Objects which support additional parameters.

What Typographical Changes and Symbols Mean

This manual uses the following conventions:

TYPEFACE	MEANING	EXAMPLE
Italics	Manuals, topics or other important items	Refer to *Developing Connectors*.
Small Capitals	Software and Component names	Your application uses a database called the CID.
Fixed Width	File names, commands, paths, and on screen commands	Go to `//home/my file`

Finding the Information You Need

The product suite comes with comprehensive documentation set that covers all aspects of building Account Management solutions. You should always read the release bulletin for late-breaking information.

Getting Started

If you are new to the edocs Telco Solutions, you should start by reading *Introducing Telco Service Manager*. This manual contains an overview of the various components along with a list of the available features. It introduces various concepts and components you must be familiar with before moving on to more specific documentation. Once you have finished, you can read the manual which covers different aspects of working with the application. At the beginning of each manual, you will find an introductory chapter which covers concepts and tasks.

Designing Your Solution

While reading *Introducing Telco Service Manager*, you should think about how the different components can address your Account Management Solution's needs.

You can refer to *Developing Telco Service Manager* for information about extending the object model, application security, and other design issues. The *CID Reference Guide* also gives you the information about how the information in your solution is managed and stored.

Installing Your Telco Service Manager

You should start by reading the Release Bulletin. For detailed installation and configuring information, refer to *Installing Telco Service Manager*. This manual covers installing TSM on one or more computers. It also contains the information you need to configure the different components you install. You might also refer to *Developing Telco Service Manager* and *Developing Connectors for Telco Service Manager* as these manuals contain information on customizing applications and working with other software.

Building Account Management Solutions

If you are designing and programming *Telco Service Manager*, you have several different sources of information. If you are programming the user interface of the solution, you should read *Developing User Interfaces for Telco Service Manager*. You also refer to the BLM Specification for detailed information about programming the user interface. For configuring the various components, you refer to *Installing Telco Service Manager*and sections in other documents which deal with the component to configure.

If you are working with the business logic of your solution, you should read *Developing Telco Service Manager*. You can also refer to the *BLM Reference Guide* for more information about the design and structure of the BLM object model. For information about how this information is stored, you should refer to the *CID Reference Guide* along with the CID Reference documentation for your database. In order to develop your application, you most likely will need to install and run the Loopback Connector. This component mimics back-end applications for development purposes. For information about installing and running this component, refer to *Using the Loopback Connector with Telco Service Manager*.

Integrating Account Management Solutions

If you are involved in configuring your solution to work with Operation Support Software (OSS), you should read *Developing Connectors with Telco Service Manager*. This manual helps you understand the integration architecture and shows you how to build connectors to connect to today's market-leading OSS software. You can also read *Using the Loopback Connector with Telco Service Manager* for information about a connector built for development purposes. Other manuals you can refer to for information about configuring your application include *Introducing Telco Service Manager* and *Developing Telco Service Manager*.

Managing Telco Service Manager (TSM)

If you are responsible for managing TSM, you should read the *Installing Telco Service Manager* for information about configuring various components and information about working with different application servers. *Administrating Telco Service Manager* covers what you need to know about managing your solution at runtime. For information about OSS systems, you should read *Developing Connectors with Telco Service Manager*.

If You Need Help

Technical support is available to customers who have valid maintenance and support contracts with edocs. Technical support engineers can help you install, configure, and maintain your edocs application.

To reach the U.S. Service Center, located in Natick, MA (Monday through Friday 8:00am to 8:00pm EST):

- Telephone: 508.652.8400
- Toll Free: 877.336.3362
- E-support: support.edocs.com (This requires a one-time online registration)
- E-mail: support@edocs.com

When you report a problem, please be prepared to provide us the following information:

- What is your name and role in your organization?
- What is your company's name?
- What is your phone number and best times to call you?
- What is your e-mail address?
- In which edocs product did a problem occur?
- What is your Operating System version?
- What were you doing when the problem occurred?
- How did the system respond to the error?
- If the system generated a screen message, please send us that screen message.
- If the system wrote information to a log file, please send us that log file.

If the system crashed or hung, please tell us.

Contents

Preface iii

Overview of Integrating Account Applications 17

About the SmartLink Framework 18
Overview of the SmartLink Framework Application Integration Services 18
Overview of the SmartLink Framework Connectors 19
Overview of the Message Schema Reference Repository 20
Overview of the SmartLink Framework Tools 21
About Integration Architectures 22
Connecting to OSS Applications Systems Through an EAI Product 23
Connecting to OSS Applications Systems Through an OSS Connector 24
Connecting to OSS Applications Directly Through a MOM 25

Installing the SmartLink Framework 27

Installing the SmartLink Framework 28
Uninstalling 30

Before You Start 31

About Developing Connectors 32
About the Principle Components of Connectors 33
Integration Processes 34
Macros 35
Scripts 36
Connector Program Files 37
Selecting the Template 39

Designing Your Connector 41

Overview of Designing Connectors 42
About the Synchronizer Template 43
About the OSS Connector Template 44
Overview of Modifying Existing Integration Processes 45
Overview of Creating Integration Processes 46

Working With the Integration Logic Studio 47

Working with Connectors 48
About Connectors 48
Working with Integration Processes 50
Working with Processors 52
About Processors 52
Working with Processors 59

Configuring Processors 61
Connecting Processors 78
Working with Scripts 81
Programming Scripts 82
Working with Macros 89
Types of Macros 89
Specifying the Macro file 91
Programming Macros 92
Samples 96
Working with Extensions 99
Working with Parameters 101

Working With Connector Extensions 105

About Extensions 106
Defining Custom Processors 107
Validating Custom Processors 108
Using Custom Processors 109
Generating Custom Processors 110
Samples 111
Sample Custom Outbound Queue 111
Sample Custom Inbound Queue 112

Working With the Message Schema Reference 115

About the Message Structure Files 116
About Working with the Message Schema Reference 117
Overview of the Business Message Structure 118
About the Control Area 119
About the Business Data Area 120
Adding New Messages 121
To add a new message (Continued) 123
Overriding Default Outbound Message Mapping 129
Customizing Message Fields 131
To add a new message field 132
Working with Synchronizer Outbound Message Fields 137
Customizing Synchronizer Inbound Messages 138
Customizing Standard Message Structures 139
Generating the Message Schema Reference Files 141
To generate message documentation files 141
To generate message mapping definition files for Synchronizer connectors 141
To generate message mapping definition files for OSS Connectors 142

Generating Your Connector 143

About Generating Your Connector 144

Running Your Connector 147

About Running the Connector 148
Specifying Processing Priority 149
Starting and Stopping the Synchronizer Connector 150
About Synchronizer Run Modes 150

Starting and Stopping the OSS Connector 151
Administrating the Connector 152
About Administrating Connectors 152

SmartLink Framework Tool Reference 155

About the SmartLink Framework Administration Tools 156
Synchronizer Connector Administration Tools 157
agentstart Syntax 157
agentadm Syntax 157
OSS Connector Administration Tools 160
ossstart Syntax 160
ossadm Syntax 160
SmartLink Framework Administration Tool 161
isfadm Syntax 161
Message Schema Reference Tool 162
schemarefTool Syntax 163
Connector Generator Tool 165
ilccGenerator Syntax 165

Core Messages 167

Core Business Objects 171

About Core Business Objects 172
About Parameters and Messages 173
Outbound Messages 173
Inbound Messages 173

Index 175

CHAPTER 1

Overview of Integrating Account Applications

In This Section

About the SmartLink Framework 18
About Integration Architectures 22

About the SmartLink Framework

The SmartLink Framework enables TSM to communicate, or integrate with the core CSP technology infrastructure, for example billing platforms, Customer Relationship Management (CRM) software, and other business and operational support systems (BSS/OSS). This in turn allows the CID to maintain a synchronized cache of semi-static customer and service data, as well as passing requests and responses between the various systems.

The SmartLink Framework can be considered to comprise:

- Connectors that move data between TSM and the transport layer. These connectors are called Synchronizers.
- Connectors that move data between BSS/OSS applications and the transport layer. These connectors are called BSS/OSS Connectors.
- A repository of message definitions to build and process messages for all common transactions. This is the Message Schema Reference Repository.
- Associated tools to design and configure Connectors along with administration tools to run and manage them.

Overview of the SmartLink Framework Application Integration Services

When integrating TSM, you need a set of fundamental integration services to exchange information between the application and the BSS/OSS. The SmartLink Framework provides the application integration services needed for the TSM to communicate with any BSS/OSS system. These services include:

- Message structure transformation

  This involves transforming message structures from one structure to another or one format to another.
- Message data mapping

  This involves filling a message structure with data extracted by core or external APIs.
- Message routing

  This involves dispatching messages to different destinations.
- Middleware interaction

  This involves sending and receiving messages using middleware.
- Error handling

  This involves catching and managing exceptions and errors.
- Call sequencing

  This involves core or external API call sequencing.

Depending on the service, the SmartLink Framework may have either a dedicated component or a core component which helps perform the service.

Overview of the SmartLink Framework Connectors

The SmartLink Framework is not only a set of application integration services, but is a framework in which the providers of the services work together.

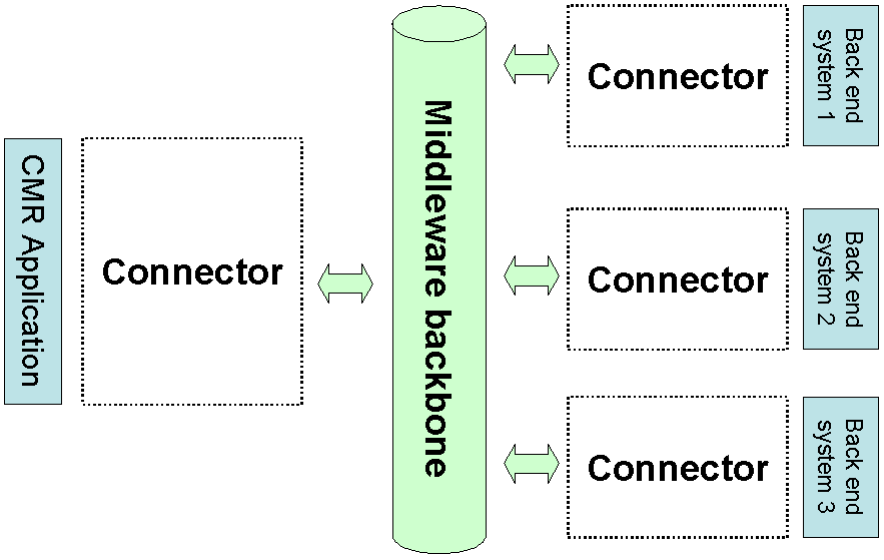
In order to integrate an application, these service providers, or SmartLink Framework components, are arranged into dataflows. This means that they are connected together and organized into a step-by-step sequence. A connector is part of the SmartLink Framework that is responsible for moving data between applications and transport layers.

There are two types of connectors:

- Synchronizers

  They are responsible for data exchanged between your TSM application and the transport layer
- OSS Connectors

  They are responsible for data exchanged between the BSS/OSS applications and the transport layer



For example, the Synchronizer connector is responsible for moving data between the TSM and the transport layer. This connector uses different connector components to carry out the following tasks:

- **Send data to the middleware**
  1. Extract the requests from the request queue located in the CID
  2. Use these requests to generate XML messages
  3. Send this XML message to the middleware backbone
- **Receive data from the middleware**
  1. Receive the XML message from the middleware
  2. Use these messages to create CID objects
  3. Update the CID by running scripts which use the CID objects

Overview of the Message Schema Reference Repository

The Message Schema Reference repository is a repository of message definition XML files. Each message handled by a connector has a specific definition file to manage the exchange of information.

These messages fall into one of the following categories:

- REQ Messages

  The connector creates these messages using the requests in the CID and sends them to the middleware.
- DO Messages

  The connector receives these messages from the middleware and uses them to update the CID.
- System

  These definition files are for other types of messages which do not fall into one of the categories above.

Each of these messages has a corresponding definition file to specify the structure of the message and its mapping definitions.

Overview of the SmartLink Framework Tools

The SmartLink Framework comes with a set of tools to help you create and manage your connectors. The tools include:

- The Integration Logic Studio

  This tool is for designing connectors. Its easy-to-use interface makes it easy to design and configure your connector.

- The Connector Administration tools

  These tools are for running and managing the connectors.

About Integration Architectures

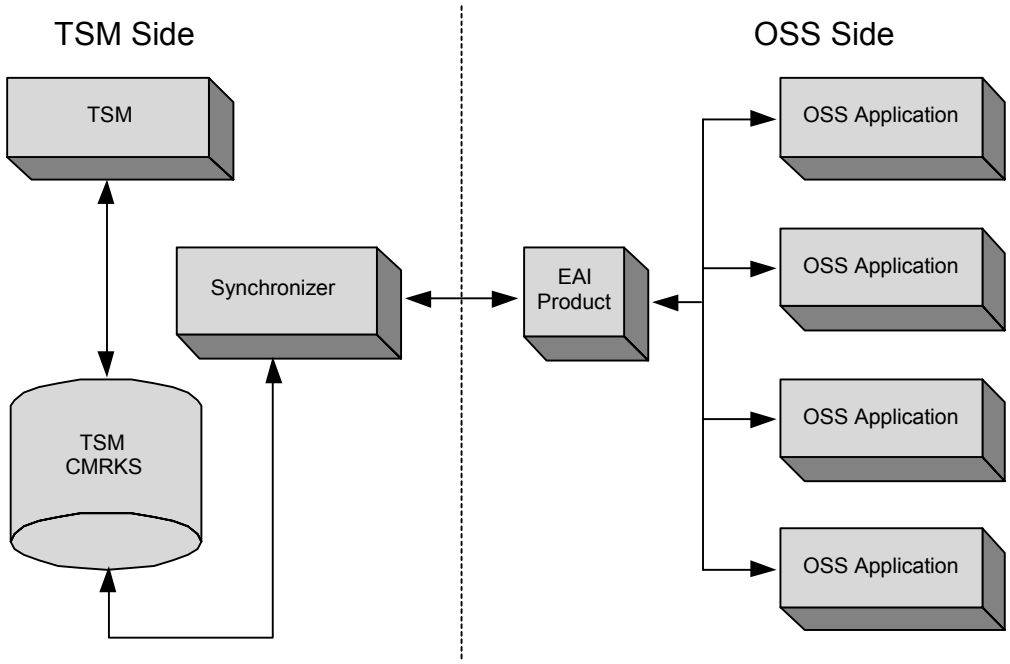
Because of the many different technological infrastructures deployed, the SmartLink Framework is based on a flexible architecture, offering support for the multiple environments. Example integration scenarios include:

- Enterprise Application Integration (EAI)
- Message Oriented Middleware (MOMs) (Java Message Service)
- Direct connection to legacy APIs

The following are possible integration architectures:

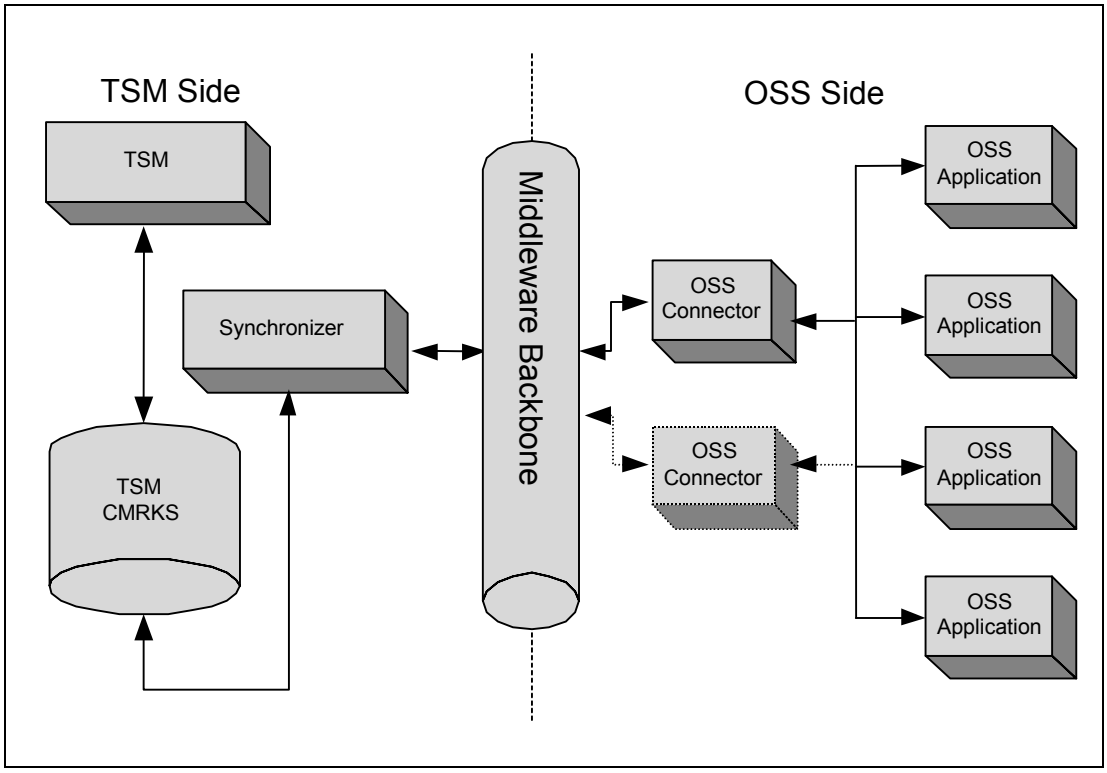
- Connecting to OSS Applications Systems Through an EAI Product
- Connecting to OSS Applications Systems Through an OSS Connector
- Connecting to OSS Applications Directly Through a MOM

Connecting to OSS Applications Systems Through an EAI Product



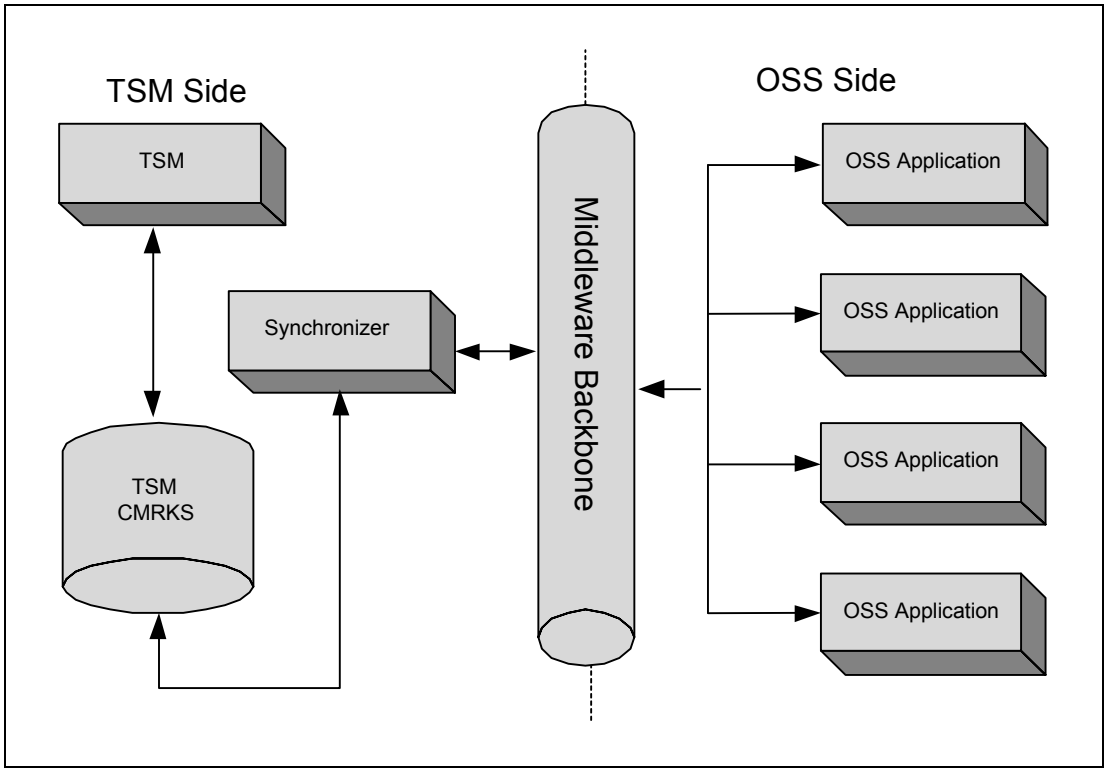
ARCHITECTURE WITH AN EAI PRODUCT	
When to use	EAI product already used by the OSS system TSM model can be imported into EAI product repository TSM and OSS applications integrated through the EAI product
Benefits	Leverage EAI power (make use of workflow, transaction and process management features...) Scalable Recommended solution

Connecting to OSS Applications Systems Through an OSS Connector



ARCHITECTURE WITH A CONNECTOR ON THE OSS SIDE	
When to use	No EAI product already being used by the OSS system OSS applications provide exposed Java APIs Middleware backbone available
Benefits	Scalable Ready to go solution not requiring any EAI product

Connecting to OSS Applications Directly Through a MOM



ARCHITECTURE WITHOUT THE CONNECTOR ON THE OSS SIDE	
When to use	The OSS systems already know how to interact with the middleware backbone
Benefits	Integration with existing middleware backbones

CHAPTER 2

Installing the SmartLink Framework

In This Section

Installing the SmartLink Framework 28
Uninstalling 30

Installing the SmartLink Framework

To install the SmartLink Framework components

1. Run the installer application on the TSM CD-ROM for your platform. The TSM Installer appears.
   - AIX: *tsm_aix.bin*
   - HPUX: *tsm_hpux.bin*
   - Solaris: *tsm_solaris.bin*
   - Windows: *tsm_windows.exe*
2. Click *Next*. The License Agreement window appears.
3. Read and accept the license agreement then click *Next*. The Location window appears.
4. Enter the TSM home directory then click *Next*. The Installation Type window appears.
5. Select *Custom Installation*. The Product Components window appears.
6. Select the following components and all of their subcomponents:
   - Synchronization Framework
   - SmartLink Framework Message Reference
   - OSS/BSS Connectors
7. Click *Next*.
8. Follow the on screen instructions. When finished, the installer displays a message.

To install the Integration Logic Studio

1. Run the installer *ils_windows.exe* on the TSM CD-ROM (Integration Logic Studio only runs on Windows). The Integration Logic Studio Installer appears.
2. Click the link to download the version of Integration Logic Studio to install.
3. Run the installer. The Integration Logic Studio Installer appears.
4. Click *Next*. The License Agreement window appears.
5. Read and accept the license agreement then click *Next*. The Location window appears.
6. Enter the Integration Logic Studio home directory then click *Next*. The Installation Type window appears.
7. Choose Integration Logic Studio then click *Next.* The Choose Shortcut window appears.
8. Choose the location of the shortcuts then click *Next*. The Pre-Installation Summary window appears.

9 Review the summary and d the following:

- If the summary is correct, click Install to begin installation
- If the summary is not correct of if you want to change a setting, click *Previous.*

When finished, the installer displays a message.

Uninstalling

TSM comes with an uninstaller to completely remove the SmartLink Framework components from your computer. This uninstaller removes only the components installed using the installer.

Due to the limitations of the installer, the uninstaller can only uninstall the components installed the last time you ran the install. For example, if you install the Channels on a computer, then run the installer again to install the OSS connectors on the same computer, the uninstaller only removes the OSS connectors.

If you install TSM on a computer running Windows2000, you must use the Windows Install/Remove Programs utility. You can find this utility in the Control Panel. Go to *Start>Settings>Control Panel*.

---

1. Go to the location where you installed the component.
2. Go to `<home_dir>/uninstalldata`.
3. Run `Uninstaller`. The Uninstaller appears.
4. Click *Uninstall* to remove all of the components. When finished, the uninstaller displays a confirmation message.

CHAPTER 3

Before You Start

In This Section

About Developing Connectors 32
About the Principle Components of Connectors 33
Connector Program Files 37
Selecting the Template 39

About Developing Connectors

Before you begin, you need to be familiar with the various components which make up a connector. You also need to understand the default directory structure and the names of different connector program files.

Once you are familiar with connectors and their components, the first step is to choose the type of connector you need to develop for your TSM application.

The SmartLink Framework comes with the following connectors you use as templates:

- Synchronizer
- OSS Connector

These connectors contain all of the components you need to start designing your own connector.

When using these connectors as templates, developing connectors involves:

- Selecting the template to use
- Copying the template in a new connector configuration directory
- Designing the connector
- Generate the connector runtime files
- Customizing message structure and mapping

About the Principle Components of Connectors

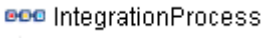
In general, connectors have the following components:

- Integration Processes

  An Integration Process is the dataflow for requests or messages between a system and the transport layer, or between the transport layer and one or more systems.

  An integration process has one or more processors, associated scripts and macros.

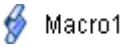
  They are represented by the following icon:

- Macros

  A Macro is a definition of external APIs to be used as commands.

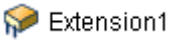
  They are represented by the following icon:

- Extensions

  An extension is a custom element you use to build Integration Processes.

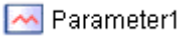
  They are represented by the following icon:

- Parameters

  A parameter is a connector-wide variable you use when setting element properties.

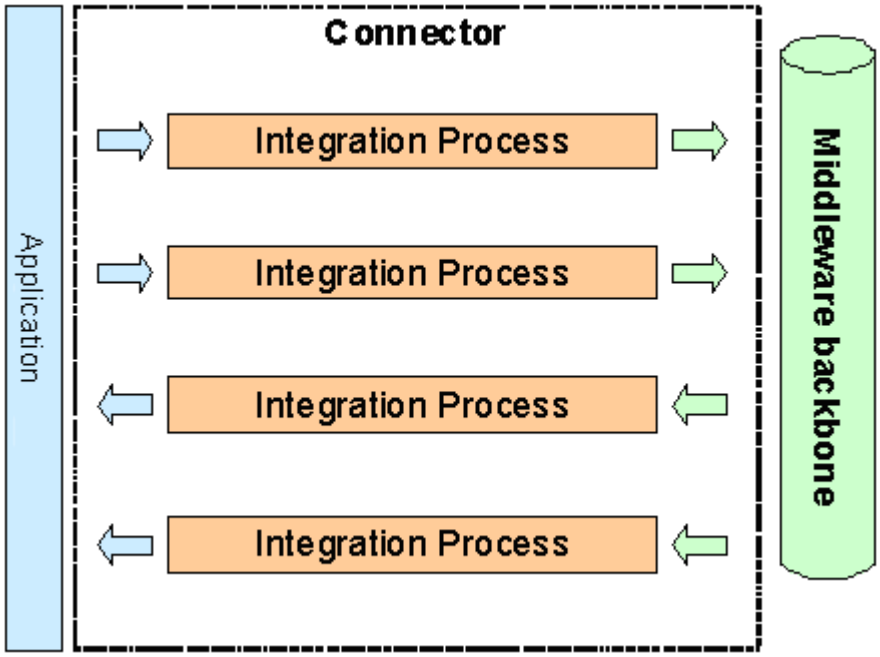
  They are represented by the following icon:

Integration Processes

An Integration Process is the dataflow for requests or messages between a system and the transport layer, or between the transport layer and one or more systems. Each connector includes one or more Integration Processes.

These processes consist of a number of processors connected together in a sequence to handle messages.



An integration process has the following:

- One or more Processors
- One or more Scripts
- One or more Macros

They are represented by the following icon:

The components of the integration process carry out the following basic tasks:

1 Extract Data

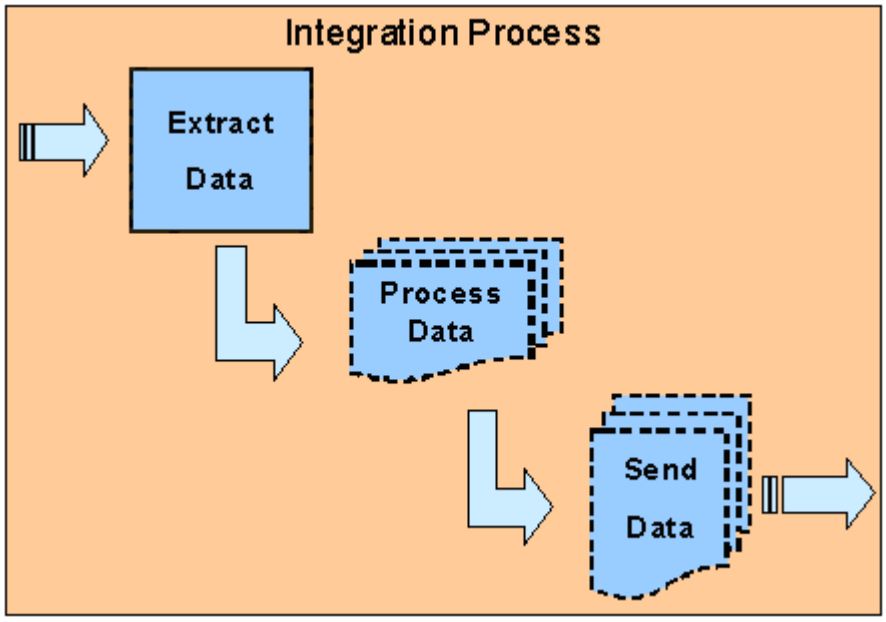
This task is extracting the data the integration process works on. This is a mandatory task and is carried out by a unique processor.

2 Process Data

This task is processing the extracted data. This is also mandatory and has several different components which handle data, forwards data to another processor, generates messages, handles errors, routes messages and so on.

3 Sending Data

This optional tasks is at the end of the processing and involves sending the data out of the connector.



Macros

A Macro element defines the custom APIs that can be called in scripts.

When using Macros to create your own API, you can:

- Wrap external Java code to custom script APIs
- Instantiate external Java classes

For instance, your connector may have to establish a connection with a back end system when the connector starts up and then reuse this connection for all Integration Processes. In this case, you can use a Macro to instantiate the connection then reuse it whenever necessary in any of your Integration Processes.

A macro can be one of the following:

- Connector Macro

  These macros can be used by all of the processors. They are located under `<connector_home>/Macros`.

- Integration Process

  These macros can only be used by the elements of the Integration Process. They are located under `<connector_home>/<integration_process>/Macros`.

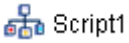
Scripts

This element points to an XML file. This XML file is a Script.

A Script is a sequence of commands or operations, which are executed by a Script Engine Processor. Each command actually calls an API, either defined within the SmartLink Framework or registered as an external Macro.

A number of core Script features include a `Switch` command which allows the script to branch conditionally on the results of any variable, providing “If-Then-Else” functionality within the script. Scripts can also execute sub-script components

They are represented by the following icon:



Because of the power and flexibility of Scripts, they are used for a number of functions:

- Updating data in the CID
- Routing messages within an Integration Process
- Calling external APIs for data processing

Connector Program Files

A connector and its components are built and configured using the following files.

- **Connector Design File**

  This file is the design file used by the ILS tool to build and configure your connector. The name of this file is the name of the connector and its extension is `ilcd`.

  This file is automatically generated file. Do not manually edit this file.

- **Connector Runtime Files**

  These files are generated by the ILS tool and have the same name as the connector design file.

  - Connector Configuration File

    This file is a runtime connector file containing the private and internal configuration settings. The extension is `ilcc`.

    This file is automatically generated file. Do not manually edit this file.

  - Connector Parameters File

    This file is a runtime connector file. It contains the runtime parameters of your connector. The extension is `ilcr`.

    You can edit this file to customize your configuration after deployment.

- **Macro Files**

  These files are the macro files.

- **Script Files**

  These files are the script files.

- **Message Schema Files**

  These XML files contain the structure and mapping of messages. Each type of message has a corresponding message schema file.

- **Configuration Files**

  These files are for configuring various parts of the connector. These files configure the database connection, system logger settings, and so on.

By default, they are installed in:

- `<connector_home>/config/connectors/connectortemplate` for the OSS connector
- `<connector_home>/config/synchronizers/synchronizer` for the Synchronizer

  where `<connector_home>` is the location where the connectors are installed.

The default connector directories:

DIRECTORY	CONTENTS
<connector_name>	Contains the following files: • Connector design file • Connector runtime files
/macro	Contains the connector macro files
/nmycfg/util	Contains the configuration files
/schema_map	Contains the mapping files for messages
/script	Contains the connector script files

Selecting the Template

Selecting the template you want to use to create your connector depends on the role it plays in integration your TSM application. There are two connector templates you can choose from. These templates correspond to the two basic types of connectors.

Once you have decided which type of connector you want to build, you do the following:

1. Create a copy of the installed template files.You copy the entire contents of the folder which contains the template.
2. Rename the directory and files

Once you have created a copy of the template, you are ready to begin working with the ILS tool to design and generate your connector.

To create a copy of a template

- For a Synchronizer connector, create a copy of the entire contents of `<connector_home>/config/synchronizers/synchronizer`
- For an OSS connector, create a copy of the entire contents of `<connector_home>/config/connectors/connectortemplate`

  where `<connector_home>` is the location where the connectors are installed.

To rename the template

To change the name of your template, you rename:

- The directory containing the template.
- The `ilcd` connector design file.
- The connector name

  You can change the connector name when you open it using the ILS.

The directory and configuration file must have the same name. If they do not, the connector will not run.

For instance, if you are creating a Synchronizer connector called `MySynch`, you copy the contents of the `<isf_home>/config/synchronizers/synchronizer` in `<isf_home>/config/synchronizers/MySynch`. You then rename the `synchronizer.ilcd` file to `MySynch.ilcd`.

CHAPTER 4

Designing Your Connector

In This Section

Overview of Designing Connectors 42
About the Synchronizer Template 43
About the OSS Connector Template 44
Overview of Modifying Existing Integration Processes 45
Overview of Creating Integration Processes 46

Overview of Designing Connectors

Designing your connector using the templates involves:

- Modifying existing integration processes
  - Adjusting the configuration of existing processors to suit your specific needs
  - Adding specific message content or type processing
  - Adding processors in order to perform specific processing on messages
  - Adding message routing features
- Creating new integration processes
  - Creating new message queues
  - Creating new message processing

You use the Integration Logic Studio (ILS) to modify and create integration processes.

About the Synchronizer Template

This connector is a pre-configured, ready-to-run connector. This connector does the following:

- Extracts the requests from the CID.

  For each extracted request:

  - Generates the corresponding XML message
  - Sends the XML message over the middleware backbone

  The `CID_Requests_Generation` Integration Process manages these actions.
- Manages incoming messages from the back end (systems or other connectors)
  - Extracts the incoming XML message from the middleware backbone
  - Parses the message
  - Creates the corresponding CID business objects
  - Updates the CID using the CID business objects

  The `CID_Update` Integration Process manages these actions.

About the OSS Connector Template

This connector template is a pre-configured OSS connector. This connector does the following:

- Manages the incoming messages from a TSM connector (Synchronizer)
- Extracts the incoming message from the middleware backbone
- Parses the XML message
- Calls the Script APIs to update your back end system with the information in the message
- Using the reply of the back end system, builds the corresponding reply message
  - Generates the corresponding XML message
  - Sends the message over the middleware backbone

The `Inbound_Message_Handling` Integration Process handles the core actions of this connector.

This connector must be generated by the Integration Logic Studio before you can run it. This template handles messages but does not process the data they contain.

Overview of Modifying Existing Integration Processes

The integration of your TSM application may require you to modify the default configuration and behavior of existing integration processes.

You can modify:

- Message queue parameters
- Request Extraction parameters
- Transmapper Message type lists
- Retry parameters

If your messages have custom fields or when you use custom message types, you may have to modify the following:

- Transmapper Message type lists
- Message mapping and structure definition files
- Scripts which process a specific message type

If you need to perform specific processing on messages, you may have to modify the following:

- Message Queues
- Retry and error handling components

If you need to add message routing features, you may need to modify the following:

- Message envelope based routing through message broadcasters and message type filters
- Message content based routing through scripts

Overview of Creating Integration Processes

If the template does not meet all of your needs or if your TSM application requires more specific handling of messages, you create integration processes in your connector .

Creating integration processes involves:

1. Creating a new integration process and entering its name
2. Create a single processor in charge of extracting your data, either through a message inbound queue or an request queue agent or any extension you may have created.
3. Add as many processors you need in order to perform the necessary processing on your data and connect them to create your data flow.

If necessary, add one or several processors in charge of sending your data to a middleware using a message outbound queue or any extension you may have created.

CHAPTER 5

Working With the Integration Logic Studio

In This Section

Working with Connectors 48
Working with Integration Processes 50
Working with Processors 52
Working with Scripts 81
Working with Macros 89
Working with Extensions 99
Working with Parameters 101

Working with Connectors

About Connectors

A connector is part of the SmartLink Framework that is responsible for moving data between applications and transport layers.

There are two types of connectors:

- Synchronizers

  They are responsible for data exchanged between your TSM application and the transport layer
- BSS/OSS Connectors
- They are responsible for data exchanged between the BSS/OSS applications and the transport layer

Each connector includes the following components:

- Integration Processes

  An Integration Process is the dataflow for requests or messages between a system and the transport layer, or between the transport layer and one or more systems.

  One or more Integration Processes may be declared as being entry points of messages.
- Macros

  A Macro is a script to define external APIs to be defined and used as commands.
- Extensions

  An extension is a custom element you use to build Integration Processes.
- Parameters

  A parameter is a connector-wide variable you use when setting element properties.

To create connectors

1. Choose *File* > *New*. The *Select a connector template* dialog box appears
2. Browse and select the directory containing the connector to use as a template. The *Select your connector configuration home directory* dialog box opens.
3. Browse and select the parent directory of your connector home directory and files then choose *Select*. The *Create new Connector* dialog box appears.
4. Enter the name of your connector. This name is used for:
   - The directory containing the connector files
   - The name of the connector design file

5 Select *OK*. The Integration Logic Studio creates the connector and its corresponding files in the specified directory. When finished, the connector appears in the Connector explorer and a message appears in the message pane.

To open connectors

1 Choose *File* > *Open*. The *Open Connector* dialog box opens.

2 Select the connector file to open.

3 Choose *Open*. The connector opens in the Connector Explorer.

To validate connectors

Choose *Connector* > *Check connector*. The ILS validates the connector.

After verification, the ILS displays the result in the message pane. If your connector is invalid or if a problem occured during verification, the ILS displays messages to help you resolve the problem.

To close connectors

1 Choose *File* > *Close*. The connector closes. If any changes have not been saved, you are prompted to save or discard the changes.

To save connectors

1 Do one of the following:

- Choose *File* > *Save* to save the connector.
- Choose *File* > *Save As* to save the connector with a different name.

To generate documentation

1 Choose *Connector* > *Generate documentation*. The *Export Documentation* dialog box opens.

2 Select the location and enter the name of the HTML file to generate.

3 Choose *Save*. The ILS generates the contents of the connector in an easy-to-read HTML file. When finished, the ILS displays a message in the message pane.

Working with Integration Processes

A connector includes one or more Integration Processes.

An Integration Process is the dataflow for requests or messages between a system and the transport layer, or between the transport layer and one or more systems.

Each Integration Process consists of a number of Processors connected together in a specific sequence. An Integration Process may also be declared as being an entry point of messages in the connector.

Working with Integration Processes involves:

- Creating Integration Processes
- Copying Integration Processes
- Opening Integration Processes
- Renaming Integration Processes
- Deleting Integration Processes

To create Integration Processes

1. Open the connector.
2. In the Connector Explorer, expand *Integration Processes.*
3. Right click *Integration Processes* then choose *New*. The *Create New Integration Process* dialog box appears.
4. Enter the name of the integration process then choose OK. The new Integration Process appears.

To open Integration Processes

1. Open the connector.
2. In the Connector Explorer, expand *Integration Processes.*
3. Double-click the Integration Process. The Integration Process appears in the Workspace.

To close Integration Processes

1. On the Workspace, select the tab of the Integration Process to close. The Integration Process appears in the Workspace.
2. Right click the tab then choose *Close*. The Integration Process disappears from the Workspace.

To refresh the contents of an Integration Process

1. On the Workspace, select the tab of the Integration Process to refresh. The Integration Process appears in the Workspace.
2. Right click the tab then choose *Reload*. The Integration Process view in the Workspace refreshes and displays all of the components which currently make up the process.

To copy an Integration Process

1. Open the connector.
2. In the Connector Explorer, expand *Integration Processes.*
3. Select the Integration Process to duplicate.
4. Right click the Integration Process then choose Copy.
5. Right click the location under *Integration Processes* where you want to place the process then choose *Paste*. The *Create New Integration Process* dialog box appears.
6. Enter the name of the integration process then choose OK. The copy of the Integration Process appears with the name you entered.

To rename Integration Processes

1. Open the connector.
2. In the Connector Explorer, expand *Integration Processes.*
3. Select the Integration Process to rename.
4. Slowly double-click the name. The name changes to a text field.
5. Enter the new name of the Integration Process.

To delete an Integration Process

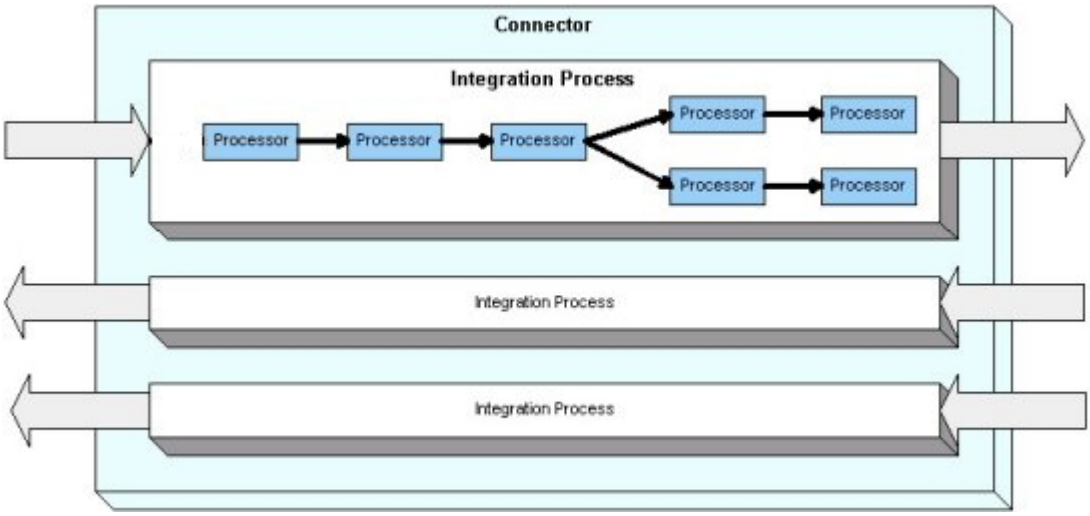
1. Open the Integration Process.
2. In the Connector Browser, find the Integration Process to remove.
3. Right-click the Integration Process then select *Delete*. A confirmation dialog box appears.
4. Choose one of the following:
   - *Yes* to permanently remove the element from your configuration
   - *No* to cancel

Working with Processors

About Processors

An Integration Process is the dataflow for requests or messages between a system and the transport layer, or between the transport layer and one or more systems.

Each Integration Process consists of a number of Processors connected together in a specific sequence as shown in this diagram:



There are several different types of processors you can use to build your Integration Process. They include:

- Request Queue Processors

  These processors handle the extraction of requests from the CID.

- Transmapper Processors

  These processors handle transforming Java objects used for interaction with systems (your TSM application and BSS/OSS) and XML strings (used for message transport)

- Utilities and Error Processors

  These processors are a collection of processors to assist with common integration workflow issues, such as error handling, retries, wait states, and so on.

- Routing Processors

  These processors handle complex dataflows where data does not simply flow from one processor to the next.

About Request Queue Processors

The processors in this category are for managing messages extracted from the request queue. The request queue is the table in the CID which holds the requests of the TSM.

The available Request Queue Processors include:

-  Request Queue Agent

  Request Queue Agents periodically collect requests from the CID. As this processor reads information in the CID, it is used in Synchronizer connectors. The frequency at which requests are collected can be configured anywhere from every second (for service subscriptions, handset theft etc) to a matter of hours (address changes etc). The Agent Processor can be configured to retrieve only specific types of request. This effectively determines which Integration Process, or even which Synchronizer, handles which requests.

  There can only be one Agent Processor per connector. However, the Agent Processor can be configured to generate multiple threads to increase throughput, and a number of other parameters can be defined to improve the overall messaging performance between your TSM application and the BSS/OSS platforms.

  These parameters include:

  - Number of requests processed per cycle
  - Weighting of request types processed per cycle
  - Reduced repeat cycle time for high priority events

About Transmapper Processors

The processors in this category are used to transform Java objects (used for interaction with systems – TSM and BSS/OSS) and XML strings (used for message transport).

For example, the connector receives a message in the form of a BLM object. This message needs to be transformed into an XML format the connector can work with. The BLM/XML Transmapper handles converting a BLM object into XML.

Transmappers may also have associated files that specify the structure of the output message. They may also have files that manage the mapping of the input and output messages. The BLM/XML Transmapper has both of these files to define the structure of the XML and the mapping of BLM objects to XML elements.

-  BLM/XML Transmapper

  The BLM/XML Transmapper Processor transforms a Request object into an XML string. This Processor will normally follow immediately after the Request Queue Agent Processor, transforming the minimal information held in the Request object into a complete XML message providing all data required by the target system(s). The specification for this information both in terms of the XML schema and the CSS Engine APIs required to extract the data from the CID, are defined in the Message Schema Reference.

-  Java/XML Transmapper

  The Java/XML Transmapper Processor is the more generic form of the BLM/XML Transmapper. The Java/XML Transmapper Processor transforms any Java object into an XML string. This processor will often be used within a BSS/OSS Connector to transform BSS/OSS objects into an XML message for delivery to the TSM.

-  CID Objects Builder

  The CID Objects Builder Processor transforms a Java hash table into Data Access Layer (DAL) objects, for use in updating the CID. Note that the Processor does not actually update the CID. Updating the CID is performed by the Script Engine.

-  XML Parser

  The XML Parser Processor transforms an XML string into a Java hash table.

About Router Processors

The processors in this category are for broadcasting messages and managing message flows. Routing Processors are used to create more complex dataflows, where data does not simply flow from one processor to the next.

In general, Messages are broadcast to the queues and the queue has an associated filter to determine whether or not the queue accepts the message. There are processors in this category to manage the broadcasting and filtering of messages for a single queue or multiple queue configurations.

-  Message Boradcaster
-  Multiqueue Message Broadcaster
-  Message Type Filter
-  Multiqueue Message Filter

  The Message Broadcaster Processor and Message Type Filter Processors work together to enable simple branching of dataflows. The Broadcaster Processor broadcasts all messages to a number of Message Type Filter Processors. Each Filter Processor is configured to accept specific message types only, so that, for example, trouble ticket messages are delivered to one outbound queue, while new service requests are delivered to another. The SmartLink Framework can therefore support simultaneous message delivery to multiple BSS/OSS components. The initial request can be converted to a single message, which is delivered to multiple components – or additionally transmapped into multiple messages for the different components.

-  Script Engine

  The Script Engine Processor can create very complex sequence of APIs, by running a pre-defined script to determine where each message and/or message type should be routed. As each script consists of a series of customizable command functions or operations, a script can for example, look up the end-user affected by a message, evaluate their rank within the organization (using an external system), and route messages for higher-ranking users towards a fast-track back-office process. Another example might involve calling an external credit-scoring API to help determine the different systems required to provision a new service.

  Scripts and script engines have another very specific use. A script is executed to update the CID when a message is received by your TSM application. Using the objects created by the CID Objects Builder Processor, the Script Engine calls core APIs (identified by the script) to write the new data into the CID.

About Message Queue Processors

The processors in this category are for message queues. A message queue is a temporary storage location where messages are held.

There are two types of queues:

- Inbound
  These queues are for messages that the connector receives for processing.
- Outbound
  These queues are for messages that the connector has processed.

The protocols include:

- Batch

   InboundEmailQueue
- Email

   Inbound Email Queue

   Outbound Email Queue
- File

   Inbound File Queue

   Outbound File Queue
- JMS

   Inbound JMS Queue

   Outbound JMS Queue
- JMS Topic Queue

   Inbound JMS Topic Queue

   Outbound JMS Topic Queue
- Memory

   Inbound Memory Queue

   Outbound Memory Queue
- Socket

   Inbound Socket Queue

   Outbound Socket Queue

About Error Processors

The processors in this category are for handling errors. When processing messages, errors may occur and your connector needs to be able to manage them.

For instance, if an error occurs when processing a message, you can configure your connector to retry the message. You can also specify the conversion of an error into another in order to customize your error handling or trigger other events.

-  Retry Handler

  This processor manages the resending of messages. If an error occurs when processing a message or if a connection is lost, this processor tries to send the message to a specified processor again a specified number of times before logging an error.

-  Message Retry Counter

  This processor manages the resending of messages. If an error occurs when processing a message or if a connection is lost, this processor tries to send the message again a specified number of times before logging an error.

-  Error Handler

  This process handles errors.

-  Exception Converter

  This processor converts Java exception errors.

-  CID Update Error Converter

  This processor converts error messages when an error occurs when updating the CID and updates the CID accordingly (in the reason field of the request queue table)

About Utility Processors

A set of processor types which are provided to assist with common integration workflow issues.

-  Pause

  This processor is simply a pause in the processing of messages. It receives a message and waits the specified amount of time before sending it to the next processor.

-  Encoding Converter

  This processor handles assigning character encoding for messages and XML files. By default, the messages and XML files are in UTF-8. This processor can reassign character encoding in order to use the required encoding for BSS/OSS systems.

-  Macro Runner

  This processor handles running macros. It launches the specified macro at this point in the integration process.

-  DOM Builder (XML/DOM)

  This processor handles assigning character encoding for messages and XML files. By default, the messages and XML files are in UTF-8. This processor can reassign character encoding in order to use the required encoding for BSS/OSS systems.

-  DOM Serializer (DOM/XML)

  This processor handles assigning character encoding for messages and XML files. By default, the messages and XML files are in UTF-8. This processor can reassign character encoding in order to use the required encoding for BSS/OSS systems.

-  XML Validator

  This processor handles assigning character encoding for messages and XML files. By default, the messages and XML files are in UTF-8. This processor can reassign character encoding in order to use the required encoding for BSS/OSS systems.

-  Message Counter

  This processor handles assigning character encoding for messages and XML files. By default, the messages and XML files are in UTF-8. This processor can reassign character encoding in order to use the required encoding for BSS/OSS systems.

Working with Processors

When working with the different processors, some of the actions you have to carry out are the same for all processors.

Common actions include:

- Creating Processors
- Editing Processor properties
- Renaming Processors
- Deleting Processors

To create Processors

1. Open the Integration Process.
2. Right-click *Processors* then select *Create New Processor*. The *Choose a Processor Type* dialog box appears.
3. Expand the node of the Processor Type to create.
4. Select the Processor to create.
5. Enter the name of the Processor.
6. Select *Confirm*. The new processor appears in the Connector Explorer and the Integration Process displayed in the Workspace.

To edit Processor properties

1. Open the Integration Process.
2. Expand the *Processors* node.
3. Select the Processor to modify.

   If the processor has any properties you can set, the properties appear in the Properties pane.

   Depending on the processor, you may also be able to add or remove properties. If you can add and remove properties, the *Add* and *Remove* buttons on the Properties pane are active.

To add and remove properties

1. Open the Integration Process.

2 Expand the *Processors* node.

3 Select the Processor to modify. The properties appear in the Properties pane. If the processor allows you to add or remove properties, the *Add* and *Remove* buttons on the Properties pane are active.

4 Do one of the following:

- Choose *Add* to add a new property. Depending on the Processor, a dialog box may appear or a new row created in the *Properties* pane.
- Select the property then choose Remove to delete the custom property.

To rename Processors

1 Open the Integration Process in the Workspace.

2 Select the Processor to rename.

3 Select the name of the Processor. The name changes to a text field.

4 Enter the new name of the processor.

To delete processors

1 Open the Integration Process in the Workspace.

2 Find the Processor you want to delete.

3 Press *Del*. A conformation dialog box appears.

4 Do one of the following:

- Choose OK to confirm your changes. The Processor disappears from the Workspace and a message appears in the message pane.
- Choose Cancel

Configuring Processors

Configuring Request Queue Processors

Configuring Request Queue Processors involves:

- Specifying the properties of the processor
- Specifying the request types to manage
- Specifying the sequences
- Specifying the filters

Specifying the Request Queue Agent Properties

To specify the Request Queue Agent properties

1. Open the *Integration Process*.
2. Expand the *Processors* node.
3. Select the *Request Queue Agent* processor to modify. The properties appear in the Properties pane.
4. Enter the following properties in the Properties pane:

PROPERTY	DESCRIPTION
Number of Threads	Number of threads to allocate
Napping Time	The period of time between polling (in milliseconds)
Looping Time	The period of time between each internal queue filling (in milliseconds)
Extract Synchronous requests only	Boolean to specify if the Request Queue agent extracts only Synchronous requests
Execute once	Boolean to specify if the Request Queue agent runs once. If true, the Request Queue agent extracts and processes the specified number of requests then shuts down.
No. of Requests to Extract	The number of requests to extract from the request queue
Internal Queue Size	The number of requests in the internal queue of extracted requests
Reconnection Delay	The time in milliseconds to wait between attempts to reconnect to the database
Reconnection Retries	The number of times the agent must try to reconnect to the database
BLM User Name	The user name used to authenticate the connector with the BLM
BLM User Password	The associated password

Specifying the request types to manage

To specify request types

1. Open the *Integration Process*.
2. Expand the *Processors* node.
3. Expand the *Request Queue Agent* processor to modify.
4. Right-click *Request Types* then select *Create New Request Type*. The *Create New Request Type* dialog box appears.
5. Enter the name of the Request Type then select *OK*. The Request Type appears in the Connector Explorer.
6. Select the new Request Type then enter the following properties in the Properties pane:

PROPERTY	DESCRIPTION
Enabled	Boolean to specify if the message type is enabled or disabled.
Execution Mode	The execution mode for the request type. Can be one of the following: - Asynchronous - Synchronous - Asynchronous MultiQueue
BLM Request Type	The Request Type code as declared in the CID.
Message Type	The associated REQ message.

To specify the status code of the Request Type

1. Open the *Integration Process*.
2. Expand the *Processors* node.
3. Expand the *Request Queue Agent* processor to modify.
4. Expand the Request Type to modify. The *Status Code* node appears.
5. Select the *Status Code* node. The properties of this element appear in the Properties pane.
6. Select the status values of the following properties:

PROPERTY	DESCRIPTION
Success Status	The message has been sent successfully
Failure Status	The message could not be sent
Transport Retry Status	the send operation is being retried
Transport Failure Status	The send operation for this message has failed
Waiting for ACK Status	Waiting for an Acknowledge message for the request
Acknowledged Status	An Acknowledge message has been received for the request

Specifying the Sequences

To activate sequences

1. Open the *Integration Process*.
2. Expand the *Processors* node.

3 Expand the *Request Queue Agent* processor to modify.

4 Select *Sequence*. The properties appear in the Properties pane.

5 In the Properties pane, select the value of the property:

PROPERTY	DESCRIPTION
Enabled	Specifies if sequences are enabled. Possible values: - true - false

To specify sequences

1 Open the *Integration Process*.

2 Expand the *Processors* node.

3 Expand the *Request Queue Agent* processor to modify.

4 Select *Sequence*. The properties of this element appear in the Properties pane.

5 In the Properties pane, select *Add*. The *Request* row appears.

6 In the new row, enter the name of the Request.

7 Repeat as required.

Specifying Filters

To specify filters

1 Open the *Integration Process*.

2 Expand the *Processors* node.

3 Expand the *Request Queue Agent* processor to modify.

4 Right-click the *Filters* then select *Create New Filter*. A list of possible filters appears.

5 Select one of the following filters:

FILTER	DESCRIPTION
BLM Request Type	The type of the request
Request Status Code	The status code of the request
Request Generator	The Login ID of the user who generated the request
Impacted Object	The object impacted by the request
Impacted Object Type	The object type of the object impacted by the request
Request Creation Date	The creation date of the request
Organization ID	The Organization ID of the level impacted by the request

FILTER	DESCRIPTION
Starting Date	The start date of the request
Ending Date	The end date of the request
Order By Date	The date to use for sorting

The new Filter appears in the Connector Explorer.

6 Select the new Filter then enter the following properties in the Properties pane:

PROPERTY	DESCRIPTION
Enabled	Specifies if the filter is active or inactive. Possible values: - true - false

7 If required, add a custom property.

In the Properties pane, do the following:

1. Click Add. A new property called Request appears.
2. Enter the value of this new property.

To activate filters

1 Open the *Integration Process*.

2 Expand the *Processors* node.

3 Expand the *Request Queue Agent* processor to modify.

4 Select *Filter*. The properties appear in the Properties pane.

5 In the Properties pane, select the value of the property:

PROPERTY	DESCRIPTION
Enabled	Specifies if filters are enabled. Possible values: - true - false

To set the values of the filter

1 Open the *Integration Process*.

2 Expand the *Processors* node.

3 Expand the *Request Queue Agent* processor to modify.

4 Select *Filter*. The properties appear in the Properties pane.

5 In the Properties pane, select *Add*. The *Value* row appears.

6 In the new row, enter the value of the filter.

7 Repeat as required.

Configuring Transmapper Processors

Configuring Transmapper Processors involves:

- Specifying the request types to manage
- Specifying the message types to manage
- Specifying the location of the configuration files

Configuring BLM/XML Transmappers

Configuring BLM/XML Transmappers involves:

- Specifying the request types
- Specifying the configuration files for each request type

To specify request types

1 Open the *Integration Process*.

2 Expand the *Processors* node.

3 Expand the *BLM/XML Transmapper* to modify.

4 Right-click *Request Types* then select *Create New Request Type*. The *Create New Request Type* dialog box appears.

5 Enter the name of the Request Type then select OK. The Request Type appears in the Connector Explorer.

6 Select the new Request Type then enter the following properties in the Properties pane:

PROPERTY	DESCRIPTION
Structure definition file path	Full path of the XML file containing the definition of the message structure
Mapping definition file path	Full path of the XML file containing the definition of the mapping of BLM content to XML.

Configuring CID Object Builder Transmappers

Configuring CID Object Builder Transmappers involves:

- Specifying the path of the schema definition files
- Specifying the message types

To specify message types

1. Open the *Integration Process*.
2. Expand the *Processors* node.
3. Select the *CID Object Builder* transmapper to modify. The properties appear in the Properties pane.
4. In the Properties pane, select *Add*. A *Message type* row appears.
5. In the new row, select the message type.
6. Repeat steps 4 and 5 as necessary.

To specify the location of schema definition files

1. Open the *Integration Process*.
2. Expand the *Processors* node.
3. Select the *CID Object Builder* transmapper to modify. The properties appear in the Properties pane.
4. In the Properties pane, select *Schema def path*.
5. Enter the location of the Schema definition files to use.

Configuring XML Parser Transmappers

Configuring XML Parser Transmappers involves:

- Specifying the Message Types
- Specifying the configuration files for each Message Type

To specify request types

1. Open the *Integration Process*.
2. Expand the *Processors* node.
3. Expand the XML Parser to modify.
4. Right-click *Request Types* then select *Create New Request Type*. The *Create New Request Type* dialog box appears.
5. Enter the name of the Request Type then select OK. The Request Type appears in the Connector Explorer.
6. Select the new Request Type then enter the following properties in the Properties pane:

PROPERTY	DESCRIPTION
Structure definition file path	Full path of the XML file containing the definition of the message structure

To add Request Type Properties

1. Open the *Integration Process*.
2. Expand the *Processors* node.
3. Expand the XML Parser processor to modify.
4. Select the Request Type to modify. The properties of this element appear in the Properties pane.
5. In the Properties pane, select *Add*. The *Mapping definition file path* row appears.
6. In the new row, enter the full path of the XML file containing the definition of the mapping of Java to XML.

Configuring Java/XML Transmappers

Configuring Java/XML Transmappers involves:

- Specifying the request types
- Specifying the configuration files for each request type

To specify request types

1. Open the *Integration Process*.
2. Expand the *Processors* node.
3. Expand the Java*/XML Transmapper* to modify.
4. Right-click *Request Types* then select *Create New Request Type*. The *Create New Request Type* dialog box appears.
5. Enter the name of the Request Type then select *OK*. The Request Type appears in the Connector Explorer.
6. Select the new Request Type then enter the following properties in the Properties pane:

request properties - java/xml table

PROPERTY	DESCRIPTION
Structure definition file path	Full path of the XML file containing the definition of the message structure
Mapping definition file path	Full path of the XML file containing the definition of the mapping of BLM content to XML.

Configuring Router Processors

Configuring Transmapper Processors involves:

- Specifying the scripts of the Script Processor
- Specifying the message filters
- Specifying the location of the configuration files

Configuring Script Engines

Configuring Script Engine Processors involves:

- Specifying the scripts

To specify the scripts

1. Open the *Integration Process*.
2. Expand the *Processors* node.
3. Select the Script Engine to modify.
4. In the Properties pane, select *Add*. A new Script row appears.
5. Select the script from the drop-down list.
6. Repeat as necessary.

Configuring Message Broadcasters

You do not need to configure this Processor.

Configuring Message Type Filters

Configuring Message Type Filters involves:

- Specifying the filters

To specify the filters

1. Open the *Integration Process*.
2. Expand the *Processors* node.
3. Select the *Message Type Filter* to modify.
4. In the Properties pane, select *Add*. A new Filter name row appears.
5. Enter the Message type name.
6. Repeat as necessary.

Configuring MultiQueue Message Filters

Configuring MultiQueue Message Type Filters involves:

- Specifying the filters
- Specifying the filter properties

To specify the filters

1. Open the *Integration Process*.
2. Expand the *Processors* node.
3. Select the *MultiQueue Message Type Filter* to modify.
4. In the Properties pane, select *Add*. A new Filter name row appears.
5. Enter the Message type name.
6. Repeat as necessary.

To specify the filter properties

1. Open the *Integration Process*.
2. Expand the *Processors* node.
3. Expand the *MultiQueue Message Type Filter* to modify.
4. Select the filter to modify. The properties of this element appear in the Properties pane.
5. In the Properties pane, enter the following properties:

PROPERTY	VALUE
acknowledge	Specifies if the filter needs to send an ACK message

Configuring MultiQueue Message Broadcasters

Configuring MultiQueue Message Broadcasters involves:

- Specifying the Database Driver
- Specifying the Database URL
- Specifying the Database User Name
- Specifying the Database Password

To specify the database properties

1. Open the *Integration Process*.

2 Expand the *Processors* node.

3 Select the MultiQueue Message Broadcasters to modify.

4 In the Properties pane, enter the following:

PROPERTY	DESCRIPTION
Database Driver	The name of the database driver to use
Database URL	The location of the CID database
Database User Name	The user name to connect to the database
Database Password	The associated password

Configuring Inbound Message Queue Processors

To configure Inbound File System Queue Processors

1 Open the *Integration Process*.

2 Expand the *Processors* node.

3 Select the Inbound File System Queue Processor to modify.

4 In the Properties pane, enter the following:

PROPERTY	DESCRIPTION
Directory name	Path of the file system queue
Queue Size	Max number of messages
Napping time	Time between polling (in milliseconds)
Number of Threads	Number of threads to allocate
Java Priority	JVM priority of the threads

To configure Inbound Email Queue Processors

1 Open the *Integration Process*.

2 Expand the *Processors* node.

3 Select the Inbound Email Queue Processor to modify.

4 In the Properties pane, enter the following:

PROPERTY	DESCRIPTION
Host name	Hostname of the POP server and the domain (example: myhost. mydomain.com)
User Name	POP account name
User Password	POP account password
Napping time	Time between polling (in milliseconds)
Number of threads	Number of threads to allocate

PROPERTY	DESCRIPTION
Java priority	JVM priority of the threads

To configure Inbound Socket Queue Processors

1. Open the *Integration Process*.
2. Expand the *Processors* node.
3. Select the Inbound Socket Queue Processor to modify.
4. In the Properties pane, enter the following:

PROPERTY	DESCRIPTION
Host Name	Host name
Port Number	The TCP/IP socket to connect to
Number of threads	Number of threads to allocate
Java priority	JVM priority of the threads

To configure Inbound JMS Queue Processors

1. Open the *Integration Process*.
2. Expand the *Processors* node.
3. Select the Inbound JMS Queue Processor to modify.
4. In the Properties pane, enter the following:

PROPERTY	DESCRIPTION
JMS Queue Name	Queue name obtained by JNDI
Connection Factory	JMS connection factory obtained through JNDI
Transactional Flag	Boolean. If set to true, creates a transaction for each message
Initial context factory	JMS initial connection factory
Provider URL	URL of the initial context factory provider
Napping Time	Time between polling (in milliseconds)
Number of Threads	Number of threads to allocate
Java Priority	JVM priority of the threads

To configure Inbound JMS Topic Queue Processors

1. Open the *Integration Process*.
2. Expand the *Processors* node.

3 Select the Inbound JMS Topic Queue Processor to modify.

4 In the Properties pane, enter the following:

PROPERTY	DESCRIPTION
Connection Factory	JMS connection factory obtained through JNDI
Transactional Flag	Boolean. If set to true, creates a transaction for each message
Initial context factory	Boolean. If set to true, creates a transaction for each message
Provider URL	JMS initial connection factory
Napping time	Time between polling (in milliseconds)
Number of threads	Number of threads to allocate
Java priority	JVM priority of the threads

5 Expand the JMS Topic Queue Processor.

6 Select the *Topics* node.

7 In the Properties pane, select *Add*. A new *Topic Name* row appears.

8 Enter the name of the topic.

9 Repeat as necessary.

Configuring Outbound Message Queue Processors

To configure Outbound File System Queue Processors

1 Open the *Integration Process*.

2 Expand the *Processors* node.

3 Select the Outbound File System Queue Processor to modify.

4 In the Properties pane, enter the following:

PROPERTY	DESCRIPTION
Directory Name	Path of the file system queue

To configure Outbound Email Queue Processors

1 Open the *Integration Process*.

2 Expand the *Processors* node.

3 Select the Outbound Email Queue Processor to modify.

4 In the Properties pane, enter the following:

PROPERTY	DESCRIPTION
Host name	Hostname of the POP server and the domain (example: myhost. mydomain.com)

PROPERTY	DESCRIPTION
User Name	POP account name
User Password	POP account password
Recipient	Email address of the recipient (example: user@mydomain.com)
Xmailer	Xmailer

To configure Outbound Socket Queue Processors

1. Open the *Integration Process*.
2. Expand the *Processors* node.
3. Select the Outbound Socket Queue Processor to modify.
4. In the Properties pane, enter the following:

PROPERTY	DESCRIPTION
Host Name	Host name
Port Number	The TCP/IP socket to connect to

To configure Outbound JMS Queue Processors

1. Open the *Integration Process*.
2. Expand the *Processors* node.
3. Select the Outbound JMS Queue Processor to modify.
4. In the Properties pane, enter the following:

PROPERTY	DESCRIPTION
JMS Queue Name	Queue name obtained by JNDI
Connection Factory	JMS connection factory obtained through JNDI
Transactional Flag	Boolean. If set to true, creates a transaction for each message
Initial context factory	Boolean. If set to true, creates a transaction for each message
Provider URL	JMS initial connection factory

To configure Outbound JMS Topic Queue Processors

1. Open the *Integration Process*.
2. Expand the *Processors* node.
3. Select the Outbound JMS Topic Queue Processor to modify.

4 In the Properties pane, enter the following:

PROPERTY	DESCRIPTION
Connection Factory	JMS connection factory obtained through JNDI
Transactional Flag	Boolean. If set to true, creates a transaction for each message
Initial context factory	Boolean. If set to true, creates a transaction for each message
Provider URL	JMS initial connection factory

5 Expand the JMS Topic Queue Processor.

6 Select the *Topics* node.

7 In the Properties pane, select *Add*. A new *Topic Name* row appears.

8 Enter the name of the topic.

9 Repeat as necessary.

Configuring Error Processors

To configure Retry Handler Processors

1 Open the *Integration Process*.

2 Expand the *Processors* node.

3 Select the Retry Handler Processor to modify.

4 In the Properties pane, enter the following:

PROPERTY	DESCRIPTION
Number of Repeats	The number of attempts to send the message
Delay between Repeats	The time between attempts (in milliseconds)

To configure Message Retry Counter Processors

1 Open the *Integration Process*.

2 Expand the *Processors* node.

3 Select the Message Retry Counter Processors to modify.

4 In the Properties pane, enter the following:

PROPERTY	DESCRIPTION
Number of Retries	The number of attempts to send the message
Delay	The time between attempts (in milliseconds)

To configure Error Handler Processors

You do not need to configure this Processor.

To configure Exception Converter Processors

1. Open the *Integration Process*.
2. Expand the *Processors* node.
3. Select the Exception Converter Processor to modify.
4. In the Properties pane, select *Add*. The *Add property* dialog box opens.
5. Enter the name of the source exception and the target exception.
6. Select OK. The new property appears in the Properties pane.

To configure CID Update Error Handler Processors

1. Open the *Integration Process*.
2. Expand the *Processors* node.
3. Select the CID Update Error Processor to modify.
4. In the Properties pane, enter the following:

PROPERTY	DESCRIPTION
Error string	The error message used tu update the reason column of the REQUEST table in the CID

Configuring Utility Processors

To configure the Pause Processor

1. Open the *Integration Process*.
2. Expand the *Processors* node.
3. Select the Pause Processor to modify.
4. In the Properties pane, enter the following:

PROPERTY	DESCRIPTION
Delay	The time to wait between two actions (in milliseconds)

To configure the Encoding Converter Processor

1. Open the *Integration Process*.

2 Expand the *Processors* node.

3 Select the Encoding Converter Processor to modify.

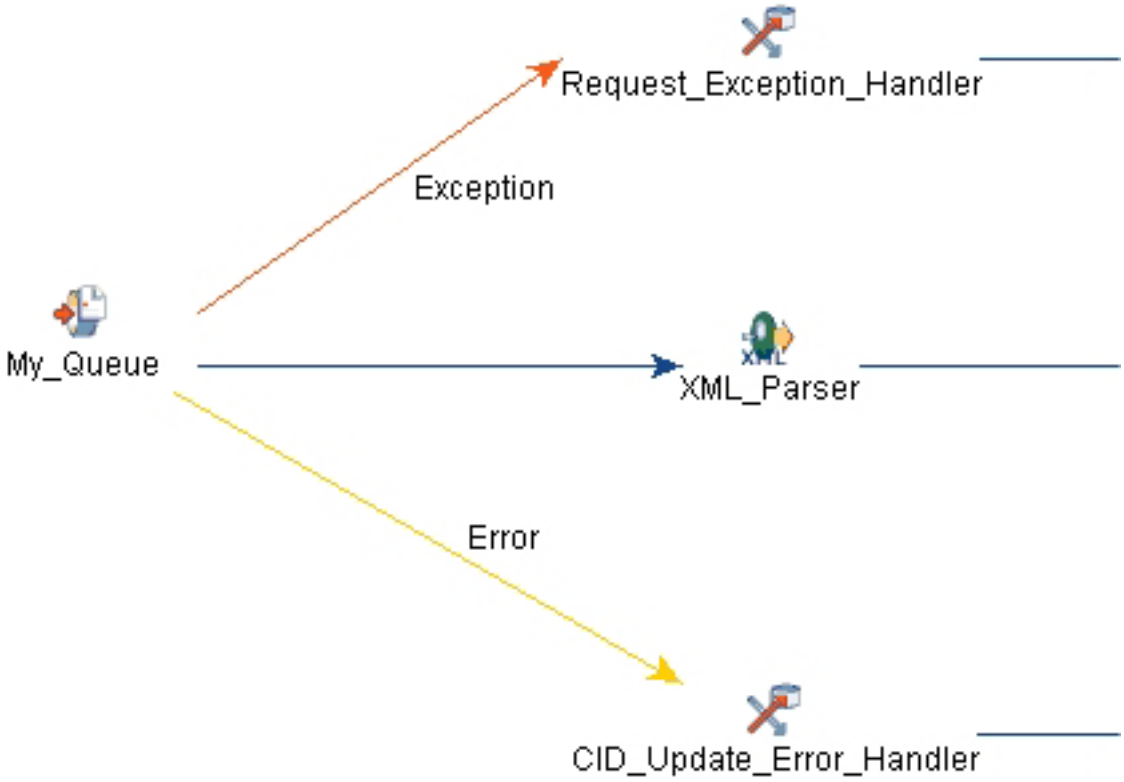
4 In the Properties pane, enter the following:

PROPERTY	DESCRIPTION
Middleware Encoding	The name of the character set to use for the messages
XML Encoding	The name of the character set to use for the XML files

Connecting Processors

The Integration Process is the dataflow for messages and requests. The dataflow consists of a set of processors and the connections between them. Each processor has an input connection and an output connection which trace the path of data through the Integration Process.

In the Workspace, the connections are represented by arrows connecting the processors.



Depending on the connector, the following types of connections can be made:

- Send Connection

  This connection is the standard connection between processors.

  In the Workspace, this connection is represented by a blue arrow.

- Error Connection

  This connection provides the processor with the initial message content that generated the error. This is the flow of data when an error occurs.

  In the Workspace, this connection is represented by a yellow arrow.

- Exception Connection

  This connection is a connection between a processor and a processor which handles exceptions. This is the flow of data when an exception occurs. A processor with an exception connection can catch exceptions for other processors further on in the data flow.

  In the Workspace, this connection is represented by a red arrow.

Working with connections involves:

- Creating connections
- Deleting connections

To connect Processors

1. Open the Integration Process in the Workspace.
2. Find the Processor you want to connect.
3. Click the Processor then drag the pointer to the destination Processor. Depending on the type of destination Processor, the following may occur:
   - An arrow appears connecting the two Processors.
   - If more than one type of connection can be made, a menu appears asking you to select the type of connection.
   - A dialog box appears informing you that you cannot connect the Processors. This occurs when:

     - The Processor is already connected

     - The Processors are not compatible

To disconnect Processors

1. Open the Integration Process in the Workspace.
2. Find the Processor you want to disconnect.
3. Select the arrow connecting the Processor to other another Processor then press *Del*. A conformation dialog box appears.
4. Do one of the following:
   - Choose OK to confirm your changes. The arrow disappears and a message appears in the message pane.

- Choose Cancel

Working with Scripts

A Script is a sequence of commands executed by a Script Engine Processor. Because of the power and flexibility of Scripts, they are used for a number of functions:

- Scripts update data in CID
- Scripts control message-routing decisions within an Integration Process
- Scripts call external APIs for data processing
-

Working with scripts involves:

- Creating the Script in the Integration Process
- Building the sequence
- Declaring the Script in a Script Engine Process

Working with Scripts

1. Open the Integration Process.
2. Right-click *Scripts* then select *Create New Script*. The *Create new Script* dialog box appears.
3. Enter the name of the script.
4. Choose OK. The new script appears.

Before you start working with this script, you must create or specify the script file.

To specify a script XML file

After creating your Script, you need specify the associated script file. A script file is an XML file containing the code of your script.

By default, the script files are located in `<connector_home>/scripts`.

1. Open the *Integration Process*.
2. Expand the *Scripts* node.
3. Right-click the script then choose *Open*. The file selection dialog box opens.
4. Browse to the directory for scripts.
5. Do one of the following:
   - Enter the full name of the file (`<file_name>.xml`) to create a new script file
   - Select the script file to associate with the script
6. Choose *Save.* The script opens in the Workspace.

To open a script

1. Open the *Integration Process*.
2. Expand the *Scripts* node.
3. Double-click the Script to open. The script opens in the Workspace.

To close a script

1. On the Workspace, select the tab of the Script to close. The Script appears in the Workspace.
2. Right click the tab then choose *Close*. The Script disappears from the Workspace.

To delete a script

1. Open the Integration Process.
2. Expand the *Script* node.
3. In the Connector Browser, find the Script to remove.
4. Right-click the Script then select *Delete*. A confirmation dialog box appears.
5. Choose one of the following:
   - *Yes* to permanently remove the element from your configuration
   - *No* to cancel

Programming Scripts

You use the Script Editor to create the sequence. This editor displays the sequence of steps in your script as a simple and easy-to-understand flow chart. This not only helps you understand how the script works, it also makes editing and modifying scripts easy.

Programming a script involves:

- Working with Steps
  - Creating Steps
  - Editing Steps
  - Deleting Steps
- Connecting the steps to create the sequence

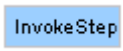
Working with Steps

The commands in a script call APIs either defined within the SmartLink Framework or registered as an external macro.

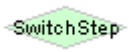
The sequence of commands are created using the following:

- 

  This step starts the sequence. Your Script must begin with an Init Step

- 

  The Invoke Step calls the API or external macro.

- 

  The Switch Step determines the next step depending on the result of a test.

- 

  The Call Step forwards the processing to another Integration Process or Processor

Working with Steps involves:

- Creating steps
- Setting step properties
- Editing steps
- Deleting steps

When you are finished, you connect the steps.

To create steps

1. Open the Script in the Workspace.
2. Right-click the Workspace and choose *Create New*. A list of steps to create appears.
3. Select the type of step to create. The *Create New Step* dialog box opens.
4. Enter the name of the step.
5. Select *OK*. The new step appears on the Workspace.

To configure Invoke steps

1. Open the Script.
2. Select the Invoke step to modify.

3 In the Properties pane, enter the following:

PROPERTY	DESCRIPTION
Method	The macro expression to call
Variable	The name of the variable to store the result in
Log	Optional text message for the logger

To configure Init steps

1 Open the Script.

2 Select the Init step to modify.

3 In the Properties pane, enter the following:

PROPERTY	DESCRIPTION
Message Type	The type of the message to process with this script. To process all message types, leave blank.
Variable	The name of the variable to store the received message in
Dispatcher	Mandatory property. Must be set to DISPATCHER.
Log	Optional text message for the logger

To configure Call steps

1 Open the Script.

2 Select the Call step to modify.

3 In the Properties pane, enter the following:

PROPERTY	DESCRIPTION
Processor	The name of the processor to call
Message Type	The type of message to send to the specified processor
Variable	The name of the variable containing the message to send
Reply	The name of the variable containing the message to correlate
Result	The name of the variable that contains the message returned by the specified processor
Log	Optional text message for the logger

To configure Switch steps

1. Open the Script.
2. Select the Switch step to modify.
3. In the Properties pane, enter the following:

PROPERTY	DESCRIPTION
Condition	Macro expression to evaluate. This value is transmitted by the transition connection.
Log	Optional text message for the logger

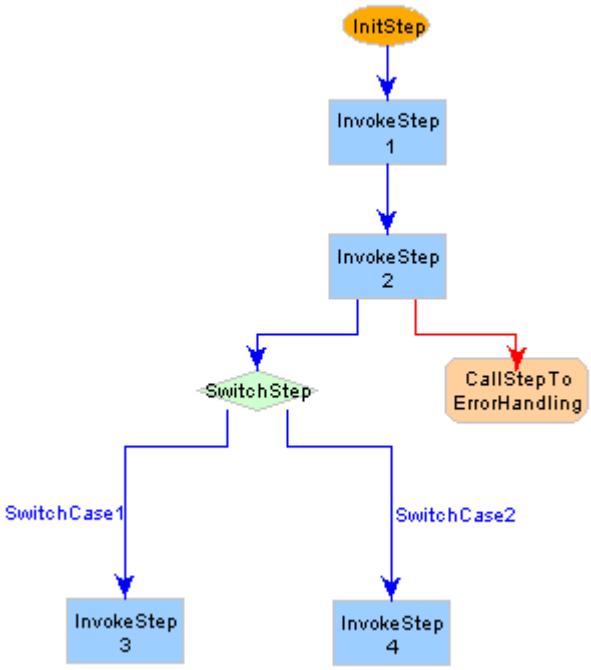
To delete steps

1. Open the Script in the Workspace.
2. Find the Step you want to delete.
3. Press *Del*. A conformation dialog box appears.
4. Do one of the following:
   - Choose OK to confirm your changes. The step disappears from the Workspace and a message appears in the message pane.
   - Choose Cancel

Connecting Steps

The script is a sequence of calls.

In the Workspace, the connections are represented by arrows connecting the processors.



The following types of connections can be made:

- Transition Connection

  This connection is the standard connection between steps.

  In the Workspace, this connection is represented by a blue arrow.

- Switch Connection

  This connection is a standard connection with the value associated with Switch step.

  Using a Switch step in your script is to create a point in the processing where the sequence may branch. The Switch step has a Condition which corresponds to the test carried out by the step. The result of this test determines the subsequent step. This result is entered as a property of the connection between the Switch step and the subsequent step.

  In the Workspace, this connection is represented by a blue arrow with the value of the decision property displayed next to the arrow.

- Exception Connection

  This connection is a connection between a step and a step which handles exceptions.

  In the Workspace, this connection is represented by a red arrow.

Working with connections involves:

- Creating connections
- Deleting connections

To connect steps

1. Open the script in the Workspace.
2. Find the step you want to connect.
3. Click the step then drag the pointer to the destination step. A menu appears asking you to select the type of connection.
4. Select the type of connection. The arrow representing the connection appears.

To create a switch connection

1. Open the script in the Workspace.
2. Find the Switch step you want to connect.
3. Click the Switch step then drag the pointer to the destination step. A menu appears asking you to select the type of connection.
4. Select Transition as the type of connection. The arrow representing the connection appears.
5. Select the arrow. The properties appear in the Properties pane.
6. In the Properties pane, select *Add*. A *Decision* row appears.
7. In the new row, enter the value of the Decision property. This is the value returned by the Condition property of the Switch step.

To disconnect steps

1. Open the script in the Workspace.
2. Find the steps you want to disconnect.
3. Select the arrow connecting the steps then press *Del*. A conformation dialog box appears.
4. Do one of the following:

- Choose OK to confirm your changes. The arrow disappears and a message appears in the message pane.
- Choose Cancel

Working with Macros

Macro statements are logically grouped and stored in a text file. They are loaded and interpreted during connector configuration loading. The Method declaration statements of a macro define custom APIs that can be invoked in scripts.

When using Macros to create your own APIs, you can, for instance:

- Wrap external Java code to custom script APIs
- Instantiate external Java classes
- Perform operations (add, multiply…) on macro variables
- Invoke external java code on macro variables

Macro statements are case sensitive.

Types of Macros

You can create macros which are limited to a specific Integration Process. You can also create create macros which can be used by the all of the processors in your connector.

A macro can be one of the following:

- Connector macro

  These macros can be used by all of the processors in a connector. They are located under `<connector_home>/Macros.<connector >/Macros`.

- Integration Process macro

  These macros can only be used by elements of the Integration Process. They are located under `<connector_home>/<integration_process>/Macros.<connector >/<integration_process>/Macros`.

Creating and programming both types of macros are the same.

When running the connector, the Macros are loaded in the following order:

1. Connector Macros
2. Integration Process Macros

To create Macros

1. Do one of the following depending on the type of Macro:

- Open the connector.
- Open the Integration Process.

2 Right-click *Macros* then select *Create New Macro*. The *Create new Macro* dialog box appears.

3 Enter the name of the Macro.

4 Choose OK. The new Macro appears.

To open Macros

1 Do one of the following depending on the type of Macro:

- Open the connector.
- Open the Integration Process.

2 Expand the *Macros* node.

3 Double-click the Macro to open. The Macro opens in the Workspace.

To close a Macro

1 On the Workspace, select the tab of the Macro to close. The Macro appears in the Workspace.

2 Right click the tab then choose *Close*. The Macro disappears from the Workspace.

To delete a Macro

1 Do one of the following depending on the type of Macro:

- Open the connector.
- Open the Integration Process.

2 Expand the *Macros* node.

3 In the Connector Browser, find the Macro to remove.

4 Right-click the Macro then select *Delete*. A confirmation dialog box appears.

5 Choose one of the following:

- *Yes* to permanently remove the element from your configuration
- *No* to cancel

This action only removes the macro from your Connector or Integration Process. It does not delete the macro source file.

Specifying the Macro file

After creating your Macro, you need specify the associated macro source file. If the file does not exist, you can create a new macro file using the ILS.

Specifying the macro involves:

- If the file does not exist, creating the macro source file
- If the file exists, declaring the macro source file

To create and specify a macro file

1. Do one of the following depending on the type of Macro:
   - Open the connector.
   - Open the Integration Process.
2. Expand the *Macros* node.
3. Double-click the Macro. The file selection dialog box opens.
4. Browse to the directory to create the macro file in.
5. Enter the full name of the file (`<file_name>.properties`) then choose Save.
6. The macro file is created and opens in the Workspace.

To specify the Macro

1. Do one of the following depending on the type of Macro:
   - Open the connector.
   - Open the Integration Process
2. Expand the *Macros* node.
3. Select the Macro to modify.
4. In the Properties pane, enter the path and file name of the Macro file.

Programming Macros

When you program a Macro, you create the Macro source file.

Programming a Macro involves:

- Creating a source file corresponding to the name entered in the path of your Macro element.
- Editing the Macro

If the right part of an expression contains brackets “xxx()”, then it will be evaluated ONLY when referenced by another expression. Otherwise as soon as the expression is parsed it will be evaluated.

Variables

In order to declare a variable and initialize it, you use the following statement:

```
Variable_name=initialization_value
```

The initialization value can be provided by virtually any expression that returns a value. Therefore you can use simple standard java object types, such as:

- String

  ```
  Var1=“MyString”
  ```
- Character

  ```
  Var1=‘a’
  ```
- Integer

  ```
  Var1=10
  ```
- Boolean

  ```
  Var1=true
  ```

You can also use more complex java objects. For instance, you can use:

- Java Vector

  ```
  Var1={1,2,MyString}
  ```
- Java Class Array

  ```
  Var1=java.lang.String[](“aa”,”bb”,”cc”)
  ```
- Java Hash table

  ```
  MyHash=[“key1”=>”value 1”,”key2”=>”value 2”]
  ```

Operators

The following operators are defined:

- Arithmetic (‘+’,’-‘,’*’,’/’)
- String (‘+’)
- Boolean (‘&’, ‘||’)
- Order relation (‘<’,’>’,’<=’,’>=’,’==’,’!=’)

Static Method Aliases

You can create an alias pointing to a java static method.

To declare a user function alias, use the `call` statement.

The function syntax is:

```
FunctionAliasName(Parameters list)=call(String classname,String
MethodSignature)(Parameters list)
```

The method signature can be defined using one of the following ways:

- You specify the method name and the list of your function parameters

  ```
  Alias(P1,P2,…)=call(classname,MethodName)(P1,P2,…)
  ```

- You specify the method name, the list of class method parameters and the list of your function parameters. For each method parameter, you must specify only the java type.

  ```
  Alias(P1,P2,…)=call(classname,MethodNameWithParametersType)(P,
  1P2,…)
  ```

- You specify the method name

  ```
  Alias=call(classname,MethodName)
  ```

- You specify the method name, the list of method parameters. For each method parameter, you must specify only the java type.

  ```
  Alias=call(classname,MethodNameWithParametersType)
  ```

Specifying the list of the class method parameters is required only when a method has several signatures.

If you specify the list of your function parameters, the referenced custom class is instantiated only when you call the function.

If you do not specify the list of your function parameters, the referenced custom class is instantiated when the macro is loaded. This also implies that it inherits the scope of the macro (connector or limited to the Integration Process.

Class constructors with parameters are not supported with the “call” statement

Your method must be declared as static.

Non-static Method Aliases

You can create an alias pointing to non-static methods of a class.

To declare a non static user function alias, you do the following:

1 Instantiate the class. If required, you can you can specify the parameters that will be passed to your class constructor.

   To instantiate a class, use the `instantiate` statement as follows:

   ```
   ClassAliasName=instantiate(String classname)(Constructor
   Parameters)
   ```

   This statement looks for the class constructor and, if found, calls it.

2 Invoke the method of the class. You specify the parameters that are passed to your method.

In order to define an alias to the method of your class, use the following statement:

```
FunctionAlias(P1,P2,…)=ClassAlias->MethodName(P1,P2,…)
```

Method names are case sensitive.

You may need to instantiate your call each time the method is called. In order to do this, use the following statements:

```
ClassAlias()=instantiate(String classname)(Constructor
Parameters)
FunctionAlias(P1,P2,…)=ClassAlias()->MethodName(P1,P2,…)
```

Class Fields

You can extract class field value.

To extract a class field value, use the `getField` statement:

```
thevalue=getField(String classname,String fieldname)
```

Functions

In order to perform a specific task on one or several objects, you can define a function.

To define a function, you specify the following:

- function name
- arguments
- the function

The syntax is as follows:

```
functionname(list of parameters)=<any macro expression>
```

The function definition itself can consist of any macro statement or method alias.

Namespaces

Use the following statement to declare a namespace:

```
NameSpace_Name {
Other macro statements…
}
```

Use the following syntax to access a previously defined element inside an existing namespace:

```
Namespace_Name.OtherElement
```

Samples

Defining Simple Functions

This sample describes the macro statement involved when you need to define a simple function that returns the sum of two numbers.

The Macro statement:

```
sum(N1,N2)=N1+N2
```

In an Invoke script step, you use the `sum(x,y)` macro statement to invoke your function.

Wrapping Static Methods

This sample describes the macro statements involved when you need to call a static method from external java code in a script or in a macro.

This example assumes:

- Your java class (`MyClass.class`) is referenced in your classpath
- Your java class has only one constructor without any parameters
- Your java class contains only one method with the following signature

  ```
  MyMethod(String aString, Object anObject)
  ```

The macro statement:

```
mymethod(P1,P2)=call("MyClass","MyMethod")(P1,P2)
```

In an Invoke script step, you use the `mymethod(x,y)` macro statement to invoke your custom class.

Wrapping Polymorphic Static Methods

This sample describes the macro statements involved when you need to call a polymorphic method from external java code in a script or in a macro.

This example assumes:

- Your java class (`MyClass.class`) is referenced in your classpath
- Your java class has only one constructor without any parameter
- Your java class contains one method with the following signature

  `MyMethod(String aString, Object anObject)`

The Macro statement:

```
mymethod(P1,P2)=call("MyClass","MyMethod(java.lang.String,java.lang.Object)")(P1,P2)
```

In an Invoke script step, you use the `mymethod(x,y)` macro statement to invoke your custom class.

Instantiating Custom Classes

With Constructor Parameters

This sample describes the macro statements involved when you need to instantiate external java classes with constructor parameters.

This example assumes:

- Your java class (`MyClass.class`) is referenced in your classpath
- Your java class has only one static constructor with one parameter

The Macro statement:

```
myclass=instantiate("MyClass")(parametervalue)
```

In an Invoke script step, you use the `myclass` macro statement to instantiate your custom class.

With Dynamic Constructor Parameters

This sample describes the macro statements involved when you need to instantiate external java classes with dynamic constructor parameters.

This example assumes:

- Your java class (`MyClass.class`) is referenced in your classpath
- Your java class has only one static constructor with two parameters

The Macro statement:

```
myclass(P2)=instantiate("MyClass")(P1,P2)
```

When called, the `myclass` macro statement will call your class `MyClass` constructor with two parameters:

- The first parameter P1 is set by the `myclass(xxx)` statement
- The second parameter P2 is set by the macro definition statement itself

Working with Extensions

When building your connector, your environment may require different types of processors that are not available by default. With the ILS, you can create your own processors or inbound and outbound queues that can be reused in all of the Integration Processes of the connector.

These custom processors are referred to as extensions.

Working with extensions involves:

- Creating the extension
- Configuring the extension
- Adding the extension to your Integration Process

To create extensions

1. Open the connector.
2. Expand the *Extensions* node. The *Inbound Queue* and *Outbound Queue* nodes appear.
3. Right-click the node corresponding to the type of extension you want to create then select *Create New Extension.* The *New extension* dialog box appears.
4. Enter the name of the extension.
5. Choose OK. The new extension appears.

To configure extensions

1. Open the connector.
2. Expand the *Extensions* node. The *Inbound Queue* and *Outbound Queue* nodes appear.
3. Expand the node corresponding to the type of extensions to modify.
4. Select the extension to modify. The properties appear in the Properties pane.
5. Choose *Add* to add a new property. A new Parameter name row created in the *Properties* pane.
6. Enter the value of the Parameter pane.
7. Repeat as required.

To add extensions to Integration Processes

1. Open the Integration Process.
2. Right-click *Processors* then select *Create New Processor*. The *Choose a Processor Type* dialog box appears.

3 Expand the *Extensions* node.

4 Select the extension to add.

5 Enter the name of the extension.

6 Select *Confirm*. The new extension appears in the Connector Explorer and the Integration Process displayed in the Workspace.

To delete extensions

1 Open the connector.

2 Expand the *Extensions* node.

3 Find the extension to remove.

4 Right-click the extension then select *Delete*. A confirmation dialog box appears.

5 Choose one of the following:

- *Yes* to permanently remove the element from your configuration
- *No* to cancel

Working with Parameters

When working with any connector, even if it is a simple connector with a straightforward workflow, you may find yourself dealing with many different elements and properties. The ILS uses parameters to help organize and simplify setting and changing the value of properties. A parameter is simply a property variable that you can use when setting variables.

For instance, you may have to enter the same property over and over again when configuring your connector. For example, for each message type you have to specify the location of the message structure definition file. If all of your message structure definition files are located in the same directory, you have to enter the same path for each message type. You can replace `c:/myconnector/shared/messagedef` with the `SCHEMA_PATH` parameter.

If you change the directory, you do not have to open and change the path of each message type. You change the value of the `SCHEMA_PATH` parameter.

This also helps makes changing directories or other common properties easier.

Parameters can be used to replace values such as:

- Paths:

  `path/subpath1/`

  `./path`

  `../../path`
- Strings:

  `dest@company.com`

  `BlueConnectorBeta`

- Boolean values
- Numeric values

For Request Queue Agent Filters, your parameter can have several values. You configure your parameter to have several values and only the declare the parameter in the Request Queue Agent Filter.

Parameter names have the following restrictions:

- No spaces (underscores are allowed)
- Only letters (no numbers or symbols can be used)
- Upper case only

Example of a valid parameter names:

- `HOME_PATH`
- `EMAIL`
- `CONNECTOR_NAME`

Working with parameters involves:

- Creating a parameter
- Setting the values
- Using the parameter in property values

To create a parameter

1. Open the connector.
2. Right-click the Parameters node then select *Create New Parameter.* The *New parameter* dialog box appears.
3. Enter the name of the parameter.
4. Choose OK. The new parameter appears.

To configure parameters

1. Open the connector.
2. Expand the *Parameters* node.
3. Select the parameter to modify. The properties appear in the Properties pane.
4. Enter the following parameters:

PARAMETER	DESCRIPTION
Description	Description of the parameter
Parameter description	Description of the value of the parameter
Type	The type of parameter. Possible values: - String - Boolean - Numeric
Multiplicity	Specifies if the parameter has one or more value. Possible values: • Simple • Multiple
Value	The value of the parameter.

5 If the parameter is a multiple value parameter:

- Select *Add* and enter a new value.
- Repeat as required.

Multiple value parameters are for Request Queue Agent Processor Filters only.

To use parameters in property values

1 Open the Integration Process.

2 Expand the *Processors* node.

3 Select the Processor to modify. If the processor has any properties you can set, the properties appear in the Properties pane.

4 Enter the name of the parameter as the value of a property. Use the syntax:

`$parameter_name$`

Example:

- Parameter name: `SCHEMA_PATH`
- Parameter value: `c:/myconnector/shared/messagedef`

The value of the `DOADDCONTACT` message type structure definition file path is:

`$SCHEMA_PATH$/doaddcontact.xml`

To use multiple value parameters

1 Open the Integration Process.

2 Expand the *Processors* node.

3 Expand the Request Queue Agent Processor node.

4 Expand the Filters and Filter nodes.

5 Select the Filter to modify. The properties appear in the Properties pane.

6 Enter the parameter name as the value of the property. Use the syntax:

```
$parameter_name$
```

CHAPTER 6

Working With Connector Extensions

In This Section

About Extensions 106
Defining Custom Processors 107
Validating Custom Processors 108
Using Custom Processors 109
Generating Custom Processors 110
Samples 111

About Extensions

When you are using a custom middleware backbone, or when you need to interact with a custom system, you can use ILS extensions. ILS extensions are custom processors which can be used in different connectors.

For instance, you might need to send messages to an unsupported custom middleware application. Or you might also need to extract data from a back end system in order to send a notification to your TSM connector.

The different custom processors you can create are:

- Inbound Queues
- Outbound Queues

An extension is the logical representation of the custom java code that will be called by the SmartLink Framework. The properties of your custom processor type are directly used to configure the java instance constructors at run time. Each parameter is directly bound to a parameter of the constructor of your java class.

Defining Custom Processors

Defining your custom processor type allows you to create in integration processes new processors based on your type.

To create a new custom processor:

1. Open the connector.
2. Expand the *Extensions* node. The *Inbound Queue* and *Outbound Queue* nodes appear.
3. Right-click the node corresponding to the type of extension you want to create then select *New* > *Extension.* The *New extension* dialog box appears.
4. Enter the name of the extension.
5. Choose OK. The new extension appears.
6. If needed, create as many as necessary properties for your processor type. Each property corresponds to a parameter of the java class constructor of your custom processor type java code

Validating Custom Processors

In order to validate a connector using your custom processor type, you have to configure the ILS validation engine to take your custom processor into account.

To configure the validation of a custom processor, do the following:

1. Open the `<connector_home>/config/custom/custom_processors.xsd` file.
2. Add the required schema declarations for your processor type
3. Save your changes.
4. Open the `<connector_home>/config/custom/custom_processors_categories.xml` file.
5. Add the required declarations for your processor type
6. Save your changes.

The ILS validation engine is now able to validate a connector using your custom processor.

Using Custom Processors

1. Open the Integration Process.
2. Right-click *Processors* then select *New* > *Processor*. The *Choose a Processor Type* dialog box appears.
3. Expand the Custom Processors node.
4. Select the the type of Processor to create.
5. Enter the name of the Processor.
6. Select *Confirm*. The new processor appears in the Connector Explorer and the Integration Process displayed in the Workspace.
7. Specify its properties in the Properties pane.

Generating Custom Processors

To generate a connector using your custom processor type, you have to configure the connector generation tool for your processor type.

To configure the connector generation tool:

1. Open the `<connector_home>/config/custom/custom_processors.cgf` file.
2. Add the required declarations for your processor type.
3. Use the syntax:

```
<processor_type>
   <plugin
name="Your_custom_processor_type">your.class.name</plugin>
</processor_type>
```

4. Save your changes.

   For example, for an OutboundQueue, you add the following:

```
<root>
 <OutboundQueues>
   <plugin name="Your_custom_processor_type">your.class.name</plugin>
 </OutboundQueues>
</root>
```

5. If required, you may need to add your class to the `CLASSPATH` in the environment settings found in `<connector_home>/bin/env`.

Samples

Sample Custom Outbound Queue

This sample is the contents of the `custom_processors.cgf` file and a sample java class that handles the generation of a custom OutboundQueue processor type with two parameters (Parameter1 and Parameter2.)

Sample custom_processors.cgf File

```
<root>
  <OutboundQueues>
<plugin name="MyQueueOutDriver">myqueuedriver_out</plugin>
  </OutboundQueues>
</root>
```

Following is a sample custom_processors.cgf file and a sample java class that handles the generation of a custom processor type (with Category OutboundQueue) with two parameters, Parameter 1 and Parameter2.

Sample java class

```
import com.netonomy.cgf.api.plugin.PluginIF;
import com.netonomy.cgf.api.plugin.PluginQueue;
import com.netonomy.cgf.isf.Processor;
import org.dom4j.Element;
import com.netonomy.cgf.isf.Verbose;
import com.netonomy.cgf.api.plugin.ElementUtil;
import java.util.HashMap;


public class myqueuedriver_out extends PluginQueue {
public String[] onDriver(Element processor, String connName, String integrationProcessName, String
proccessorName, String processorType) throws Exception {
String[] props = new String[] {
"PARAMETER1=" + ElementUtil.getPropertyValue(connName, processor, "PARAMETER1",ElementUtil.STRING_TYPE, null,
".",true),
"PARAMETER2=" + ElementUtil.getPropertyValue(connName, processor, "PARAMETER2",ElementUtil.STRING_TYPE, null,
".",true)};



String[] drivers = new String[] {"myqueuedriver_out=instantiate(\"com.myclasses.myqueuedriverout\")", "DRIVER=
myqueuedriver_out (properties. PARAMETER1, properties. PARAMETER2,0)"};


return ElementUtil.generateConfiguration(props,drivers);
  }
}
```

Sample Custom Inbound Queue

This sample is the contents of the `custom_processors.cgf` file and a sample java class that handles the generation of a custom InboundQueue processor with two parameters (Parameter 1 and Parameter2.)

Sample custom_processors.cgf File

```
<root>
  <InboundQueues>
    <plugin name="MyQueueInDriver">myqueuedriver_in</plugin>
  </InboundQueues>
</root>
```

Sample java class

```
import org.dom4j.Element;

import java.util.Iterator;

import org.dom4j.Attribute;

import java.util.HashMap;

import com.netonomy.cgf.api.plugin.ElementUtil;

import com.netonomy.cgf.api.plugin.PluginAgent;

import java.util.Vector;

import com.netonomy.cgf.isf.Constants;


public class myqueuedriver_in extends PluginAgent {


public String[] onDriver(Element processor, String connName, String integrationProcessName, String
proccessorName, String processorType) throws Exception {


String[] props = new String[] {

          "PARAMETER1=" + ElementUtil.getPropertyValue(connName, processor,
"PARAMETER1",ElementUtil.STRING_TYPE, null, "0",true),

" PARAMETER2=" + ElementUtil.getPropertyValue(connName, processor, "PARAMETER2",ElementUtil.INT_TYPE, null,
"0",true)};


String[] drivers = new String[] {

          ElementUtil.getTransitionValue(processor, connName, integrationProcessName, "send",true),

                    ElementUtil.getTransitionValue(processor, connName, integrationProcessName,
"exception",false),

          ElementUtil.getTransitionValue(processor, connName, integrationProcessName, "error",false),


          "myqueuedriver_in=instantiate(\"com.myclasses.myqueuedriverin\")",

          "DRIVER=fileQ(properties.PARAMETER1,properties. PARAMETER2)",

          "AGENT=Create_QueueAgent(properties.NBTHREADS,DRIVER,EXCEPTION,ERROR,properties.PRIORITY)"

         };

        return ElementUtil.generateConfiguration(props,drivers);

}

public String[] onStart(Element elementProcessor, String connName, String integrationProcessName, String
proccessorName, String processorType) throws Exception  {

// Nothing to generate

Vector v = new Vector();


// daemon

v.add("STARTWITH=null");


String[] props = new String[v.size()];

for(int i=0;i <v.size();i++){

          props[i] = Constants.TAB1 + (String)v.get(i);

}

return props;

}

}
```

CHAPTER 7

Working With the Message Schema Reference

In This Section

About the Message Structure Files 116
About Working with the Message Schema Reference 117
Overview of the Business Message Structure 118
Adding New Messages 121
Overriding Default Outbound Message Mapping 129
Customizing Message Fields 131
Working with Synchronizer Outbound Message Fields 137
Customizing Synchronizer Inbound Messages 138
Customizing Standard Message Structures 139
Generating the Message Schema Reference Files 141

About the Message Structure Files

The Message Schema Reference is a set of XML schema files.

Each message has a XML schema file which specifies:

- The structure of the message
- The mapping of information in the message

By default, these files are located in
`<connector_home>/data/schemaref/biz/events`

where `<connector_home>` is the location where the connectors are installed.

These schema files are used to generate the structure XML file and the mapping XML file. These files are used by the connector to read and build messages.

These files adhere to the W3C XML Schema specification. For more information about this specification, refer to the W3C web site at
`http://www.w3.org/TR/xmlschema-0/`

For the list of messages, refer to *Core Messages* in this manual.

About Working with the Message Schema Reference

Your application may not require you to modify the structure of format of messages. If you need to, you can customize the structure of messages.

When customizing the Message Schema reference, you do the following:

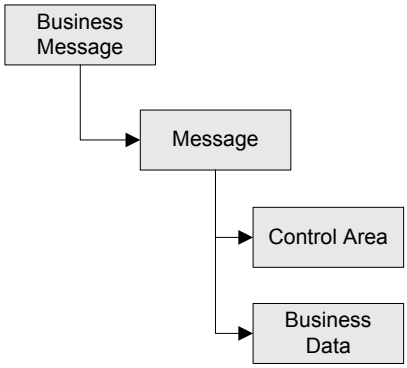
1. Create `.xsd` schema files
2. Generate the corresponding XML files using the `schemaRefTool`

Customizing messages involves:

- Adding New Messages
- Customizing Messages
- Customizing Message Fields
- Customizing Message Structure
- Generating the Message Schema Reference Files

Overview of the Business Message Structure

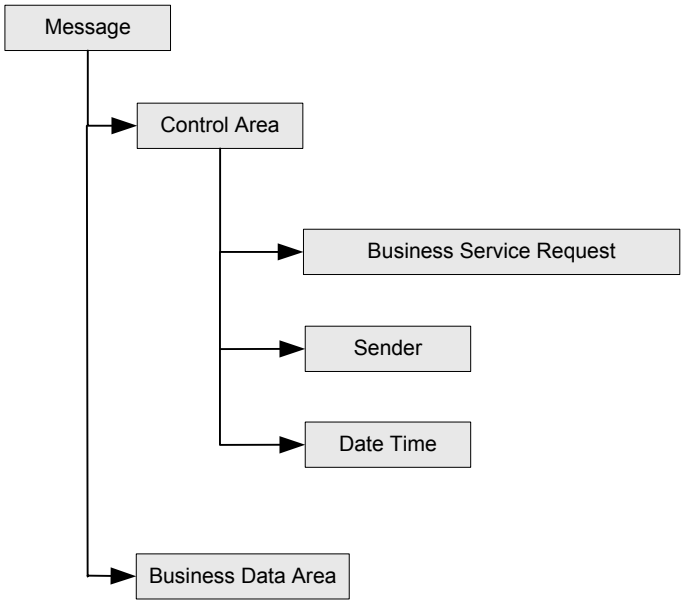
The Message Schema Reference contains the Message Schema structure and mapping definitions. These messages contain technical data and business data organized into different sections:



About the Control Area

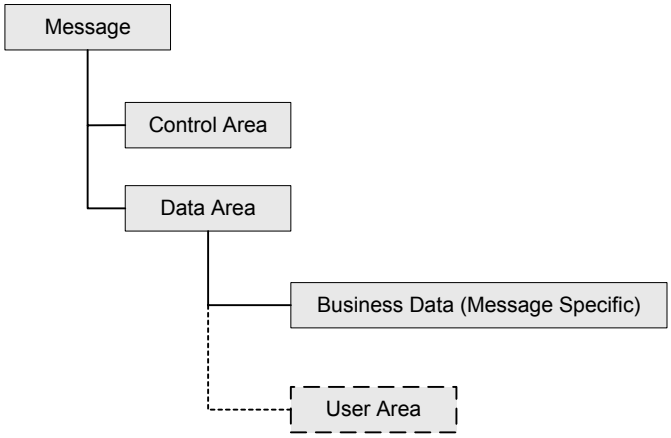
The control area is used to define general message information. This section of the message contains the following:

- The Business Service Request (BSR) qualifies the action that the Sender application wants the Receiver application to perform
- The Sender identifies characteristics and control identifiers that relate to the application which creates the Business message
- The Date Time (creation) segment is used to store the date time information of the message.



About the Business Data Area

The Business Data area contains the data dedicated to the Business message. It also contains the optional customized data the System Integrator adds to adapt it to his specific requirements:



You can extend every core Business Object to add the fields required for any specified implementation, or other business logic object.

Adding New Messages

You can add new messages to the standard messages in order to conform to the integration needs of a project.

This can be useful if you need to send fully customized messages to specific back-end systems that cannot process standard messages.

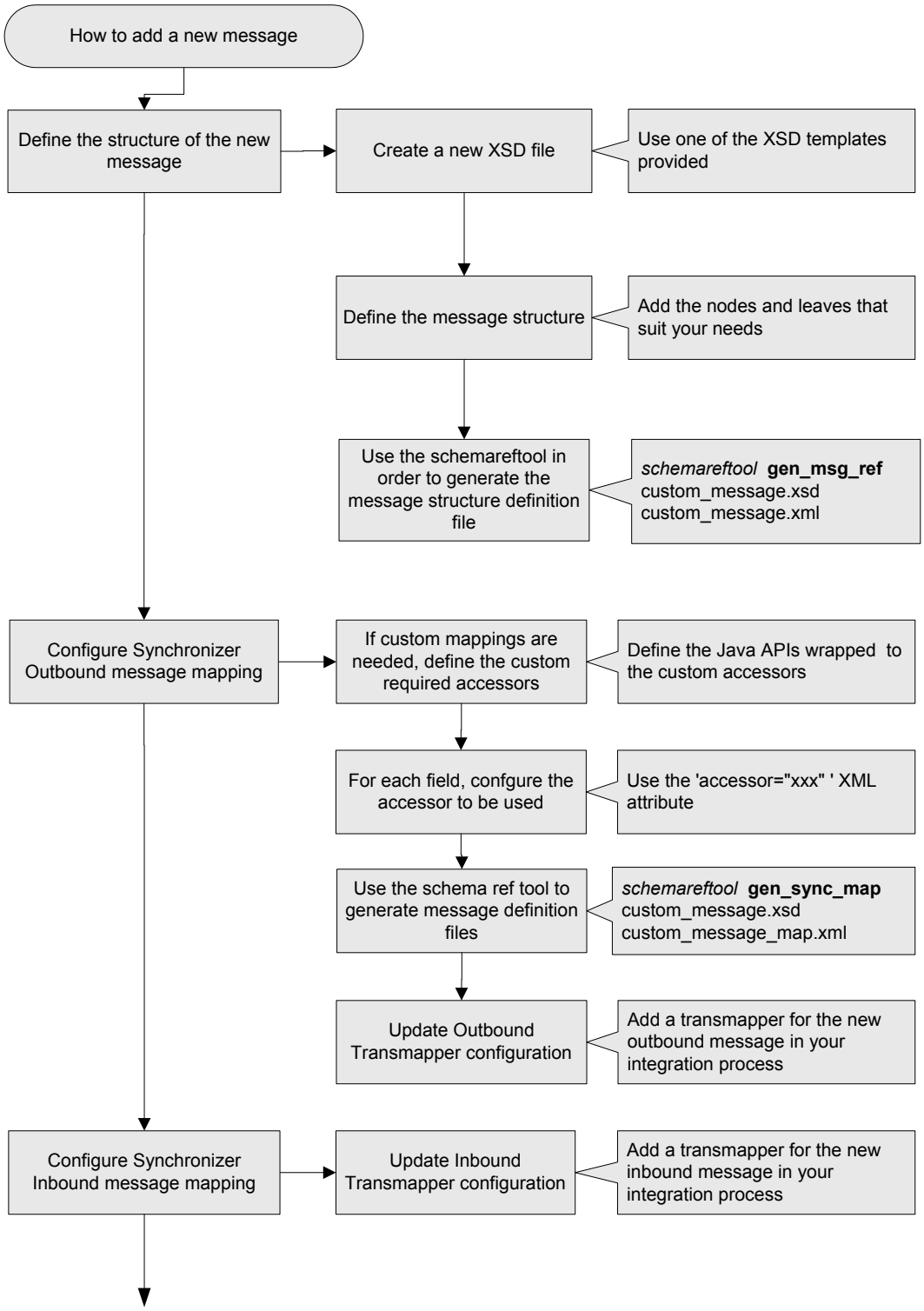
When you exchange messages with OSS Connectors or back end systems, you exchange data with them. These data must be exchanged in a structured manner. The XML format is used to format these data. This means you must define the structure of the XML message in which your are going to put (or get) data.

To create new messages, you can:

- Create a fully customized message in which you define any field you may need with the structure corresponding to you needs.
- Create a message using the standard message structure.

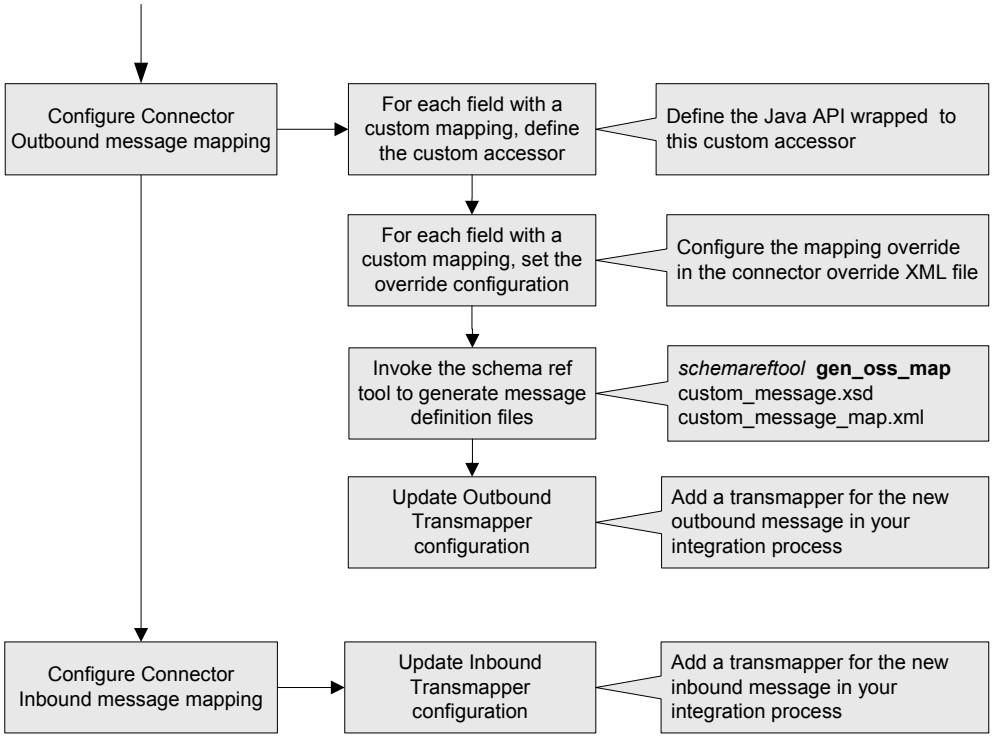
In this case, you use the standard sections (cntrol area, datarea and userarea). Then you customize the content of the datarea and userarea sections to suit your needs.

To add a new message

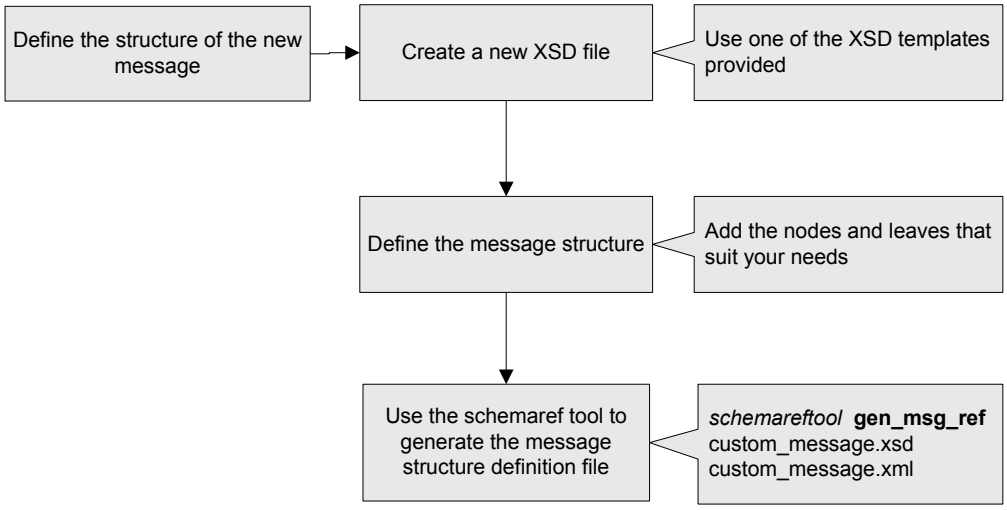


Continued on next page.

To add a new message (Continued)



Define the Structure of the New Message



To define the structure of the new message:

1 Create a new XSD file located in the custom events directory (`<home_dir>/data/schemaref/biz/custom`).

- If you want to create a new, fully customized message, use the `NewMessage_Template.xsd` template.
- If you **do not** want to create a new message, use the `NewMessageStd_Template.xsd` template.

  Both templates are provided in `Documentation/samples/Schemaref` templates.

2 In the XSD file, define your message structure: add the XML nodes and leaves that suit your needs.

You must name the root tag of the message "request"

You must name the first child tag of this root tag as the message type (for example REQADDSERVICE).

---

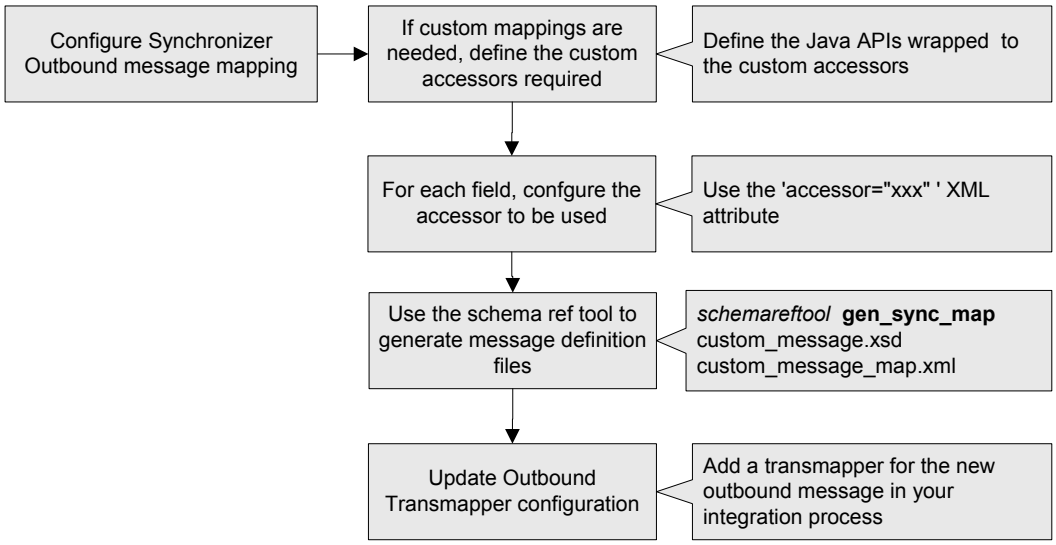
3 Once you have defined your message, generate the message structure definition files that will be used by the Synchronizer or Connector Template. Use the "`gen_msg_ref`" command of the schemareftool. You must provide the schemaref tool with the following parameters:

- custom message definition (full path to your XSD file)
- destination message structure definition file (XML file). This file should be located in `<home_dir>/share/schemaref`

Example:

```
CUST_EVENTS=/<home_dir>/data/schemaref/biz/custom

SHARE_REF=/<home_dir>/share/schemaref

/<home_dir>/bin/schemareftool gen_msg_ref
$CUST_EVENTS/reqaddservice_custom.xsd
$SHARE_REF/reqaddservice.xml
```

Configure Synchronizer Outbound Message Mapping



1 If you need to use custom java APIs to fill the content of your fields, you must define custom accessors. For each custom accessor used, you must declare the custom java API that will be wrapped to it in the Macro file `accessors_custom.properties` configuration file.

2 Once you have defined the accessors (either as custom or core), you must use the 'accessor' XML attribute to configure which accessor must be called for each field. You must create this accessor configuration in a dedicated sub-section called "appinfo" within an "annotation" section of the field you are defining:

- You must use the following template:

```
<xs:annotation>
<xs:appinfo>
      <isf:mapping accessor="your_accessor_name"/>
</xs:appinfo>
</xs:annotation>
```

For each outbound message field, you can choose one of the following options:

- generate the field
- do not generate the field
- generate the field but with no content (empty field). This option is only applicable for leaves.

  The default action is to generate the field. If you do not define any specific configuration, your new fields will be automatically present in your outbound message. For more information, refer to To activate How to activate/deactivate a field."

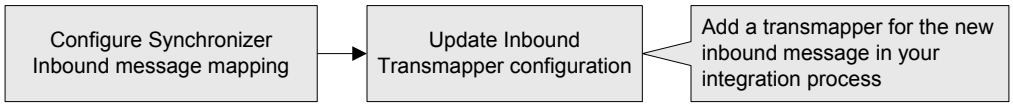
3 Once you have defined the field mapping configuration, you must generate the message mapping definition files for the Synchronizer. Use the "`gen_sync_map`" command of the schemareftool. You must provide the schemaref tool with the following parameters:

- (mandatory) custom message definition (full path to your XSD file)
- (mandatory) destination message mapping definition file (XML file). This file should be located (or created) in `<home_dir>/config/synchronizer/transmapper/accessor`
- (optional) message field activation configuration file (XML file). This file should be located (or created) in `<home_dir>/data/schemaref/biz/events/activation`

Example:

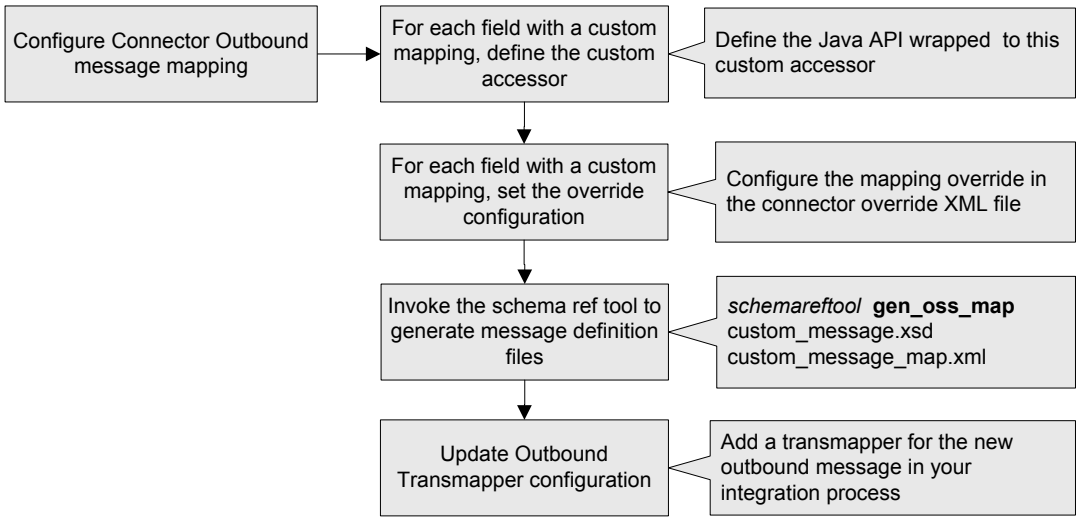
```
CUST_EVENTS=/<home_dir>/data/schemaref/biz/custom

SYNC_MAP=/<home_dir>/config/synchronizer/transmapper/access
or

/<home_dir>/bin/schemareftool gen_sync_map
$CUST_EVENTS/reqaddservice_custom.xsd
$SYNC_MAP/reqaddservice_map.xml
```

Configure Synchronizer Inbound Message Mapping



Every field of an inbound message is automatically mapped by the Synchronizer inbound transmapper default configuration. You do not have to configure any specific message field mapping. The default behavior is to map leaves to java strings and nodes to hashtables.

Configure Connector Outbound Message Mapping



1 If you need to use custom java APIs to fill the content of your fields, you must define custom accessors. For each custom accessor used, you must declare the custom java API that will be wrapped to it in the Macro file `accessors_custom.properties` configuration file.

2 The Connector outbound mapping configuration for a specific message is automatically generated based on that message's structure definition file. If you want to customize a message field mapping, you must override the default mapping configuration of this field.

You can override every message field mapping configuration by specifying the field name (and path) and the new accessor to be used in the XML override configuration file of your connector. For more information, refer to *To override a connector default mapping configuration*.

3 Once you have configured the field mapping override, you must generate the message mapping definition files for your connector. Use the `gen_oss_map` command of the schemareftool. You must provide the schemaref tool with the following parameters:

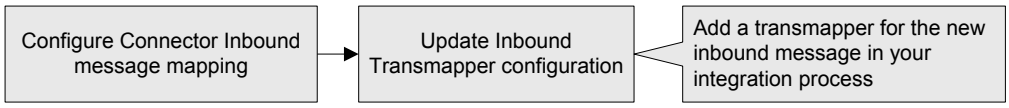
- (mandatory) custom message definition (full path to your XSD file)
- (mandatory) destination message mapping definition file (XML file). This file should be located in `<home_dir>/config/yourconnector/transmapper/accessor`
- (optional) core mapping override configuration file (XML file).
- (mandatory if core override files specified and xsd file not in custom directory) core mapping directory (`<home_dir>/data/connectors/`)

Example:

```
CUST_EVENTS=/<home_dir>/data/schemaref/BIZ/custom
OVERRIDE_DIR=/<home_dir>/data/connectors
```

```
LPBK_MAP=/<home_dir>/config/loopback/transmapper/accessor

/<home_dir>/bin/schemareftool gen_oss_map
$CUST_EVENTS/reqaddservice_custom.xsd
$LPBK_MAP$/reqaddservice_map.xml –core_override
connectortemplate.xml  -core_override_dir $OVERRIDE_DIR
```

Configure Connector Inbound Message Mapping



Every field of an inbound message is automatically mapped by your Connector inbound transmapper default configuration. You do not have to configure any specific message field mapping. The default behavior is to map leaves to java strings and nodes to hashtables.

Overriding Default Outbound Message Mapping

The Connector outbound mapping configuration for a specific message is automatically generated based on that message's structure definition file. As a consequence, if you want to customize a message field mapping, you must override the default mapping configuration of this field.

You can override a message field mapping configuration by specifying the field name (and path) and the new accessor to be used in the XML override configuration file of your connector.

To override default outbound message mapping

1. Create an XML file using the template provided in `Documentation/Samples/Schemaref templates/OverrideConfig_Template.xml`
2. For each field for which you want to change the mapping configuration, you must create one new element which specifies which accessor to use. The elements must be grouped by message type and then by element type.The following XML code gives you a sample configuration:

```
<configuration>
  <outbound_mapping_override use_custom_accessors="true">
    <event_type name="doaddservice">
      <element_type name="CNTROLAREA">
        <element_type name="SENDER">
          <element name="authid" accessor="CONSTANT:YOUR_AUTHID_ACCESSOR"/>
        </element_type>
      </element_type>
    </event_type>
  </outbound_mapping_override>
</configuration>
```

3. In order to apply your changes, you must use the schemarefTool with the `gen_oss_map` command to generate your Connector outbound message mapping configuration. Specify the XML file you created in step one as the last parameter (`-custom_override`) of the `gen_oss_map` command.

Note: If you do not use the `core_override`, your custom override has to define the default mapping for node and leaf elements:

```
<outbound_mapping_override
custom_node_accessor="getopt($ELEMENTNAME)"
custom_leaf_accessor="getopt($ELEMENTNAME)"
use_custom_accessors="true"> )
```

Customizing Message Fields

Customizing core messages involves:

- Adding or extending message fields
- Activating or deactivating message fields

Restrictions on customizing core messages:

- Synchronizer restrictions
  - New fields must be appended to existing fields
  - Field order cannot be changed
- Connector restrictions
  - Message structure definitions can only be extended. They cannot be updated.
  - Message mapping definitions can be customized (overload mechanism).

The available core messages are listed in *Core Messages*.

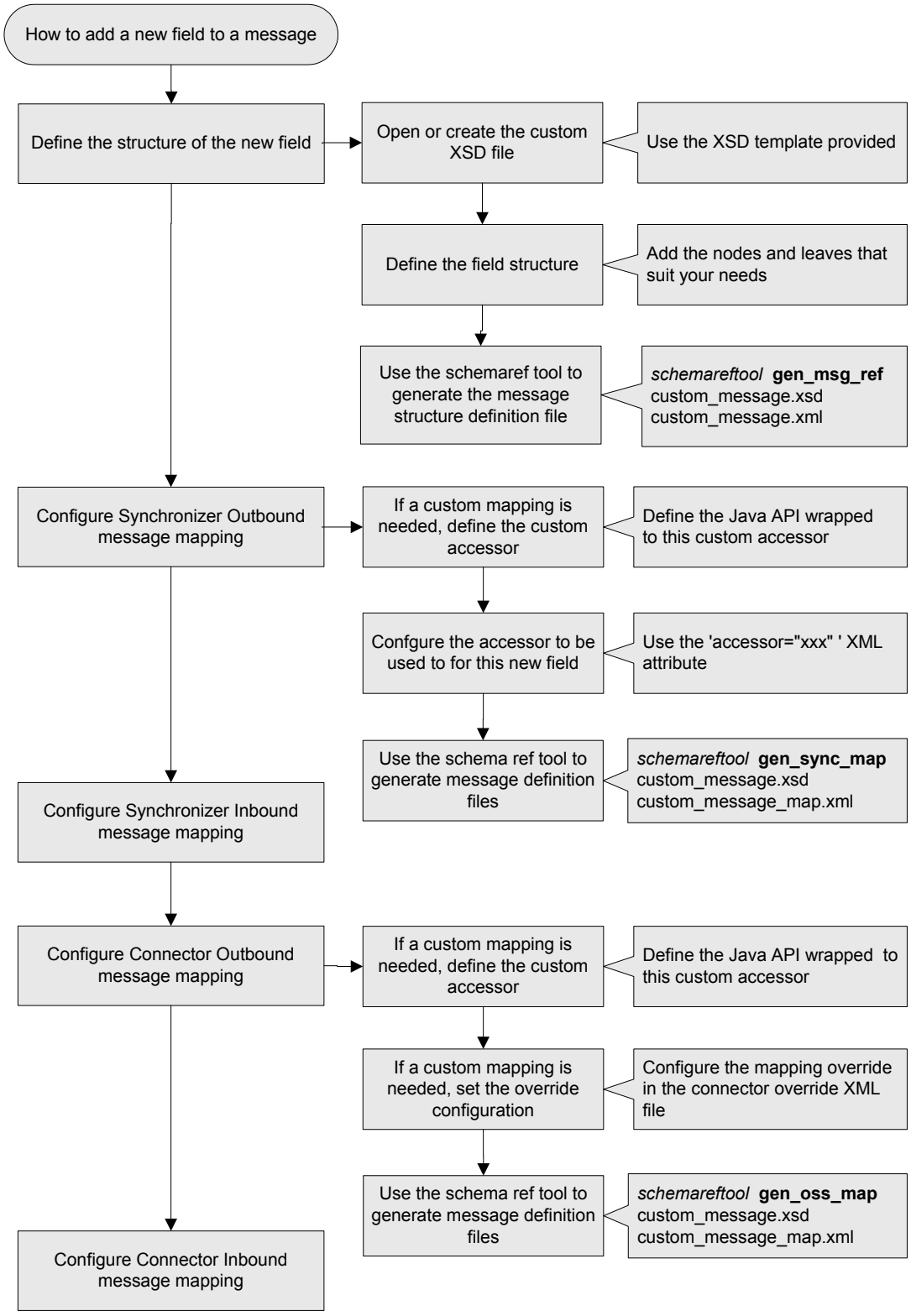
You might add a message field if you need to send additional data to the back-end systems contained in specific tags. You can also carry out specific processing tasks (such as message routing etc.) by using user-defined fields in the Synchronizer's inbound messages.

You can customize message content in two different ways.

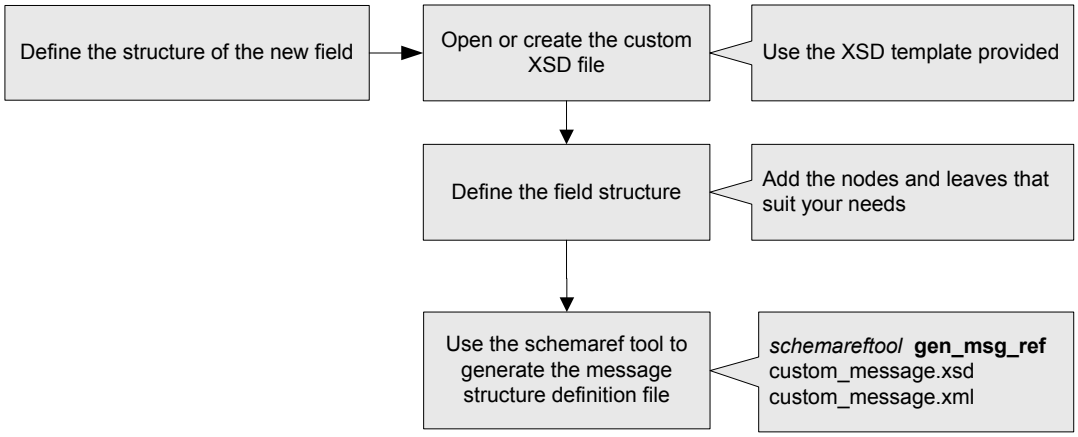
- You can add any desired field to the custom section of standard fields.

You can also add any desired field in the Userarea of each message

To add a new message field



Define the Structure of the New Field



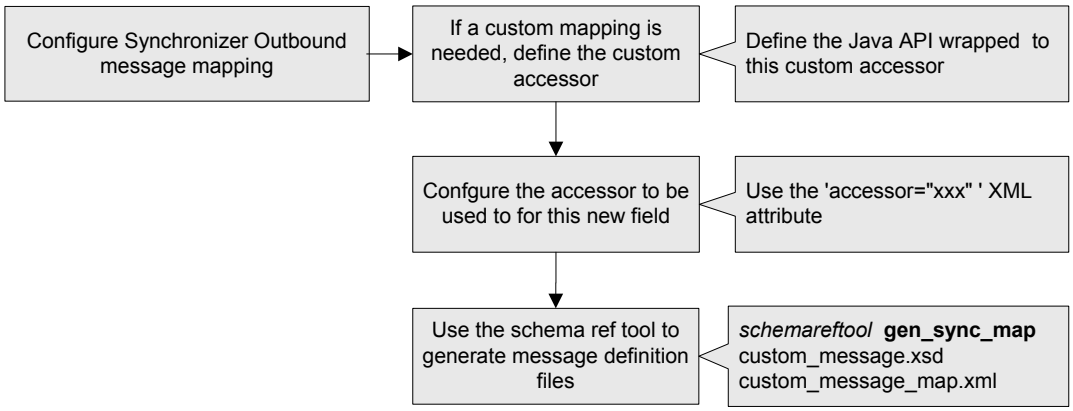
To define the structure of the new field:

1. If you have not yet customized any fields in your message, you must create a new XSD file create it in the custom events directory (`<home_dir>/data/schemaref/biz/custom`). Use the `CustomMessage_Template.xsd` template provided in `Documentation/samples/Schemaref` templates. To customize a core message you must update the reference (`<xs:redefine schemaLocation="../events/reqaddservice.xsd"`) to the core message you want to extend.
2. Once you have created your custom XSD file, open it and define your field structure: add the XML nodes and leaves that suit your needs.
3. Once you have defined your field, you must generate the message structure definition files that will be used by the Synchronizer or Connector Template. Use the `gen_msg_ref` command of the schemareftool. You must provide the schemaref tool with the following parameters:
   - (mandatory) custom message definition (full path to your custom XSD file)
   - (mandatory) destination message structure definition file (XML file). Create it in `<home_dir>/share/schemaref`

   Example:

```
CUST_EVENTS=/<home_dir>/data/schemaref/biz/custom

SHARE_REF=/<home_dir>/share/schemadef

/<home_dir>/bin/schemareftool gen_msg_ref
$CUST_EVENTS/reqaddservice_custom.xsd
$SHARE_REF/reqaddservice.xml
```

Configure Synchronizer Outbound Message Mapping



1 If you need to use custom java APIs to fill the content of your fields, you must define custom accessors. For each custom accessor used, you must declare the custom java API that will be wrapped to it in the Macro file `accessors_custom.properties` configuration file.

2 Once you have defined the accessors (either as custom or core), you must configure which accessor must be called for your new fields using the 'accessor' XML attribute. You must create this accessor configuration in a dedicated sub-section called "appinfo" within an "annotation" section of the field you are defining.

You must use the following template:

```
<xs:annotation>
 <xs:appinfo>
      <isf:mapping accessor="your_accessor_name"/>
 </xs:appinfo>
 </xs:annotation>
```

For each outbound message field, you can choose one of the following options:

- generate the field
- do not generate the field
- generate the field but with no content (empty field). This option is only applicable for leaves.

  The default action is to generate the field. If you do not define any specific configuration, your new fields will be automatically present in your outbound message. For more information, refer to "How to activate/deactivate a field."

3 Once you have defined the filed mapping configuration, you must generate the message mapping definition files for the Synchronizer. Use the `"gen_sync_map"` command of the schemareftool. You must provide the schemaref tool with the following parameters:

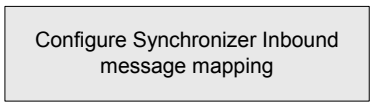
- (mandatory) custom message definition (full path to your XSD file)

- (mandatory) destination message mapping definition file (XML file). This file should be located (or created) in `<home_dir>/config/synchronizer/schema_map/accessor`
- (mandatory if custom message is not located in the custom folder) location of the core messages (`<home_dir>/data/schemaref/biz/events`)
- (optional) message fields activation configuration file (XML file). This file should be located (or created) in `<home_dir>/data/schemaref/biz/events/activation`

Example:

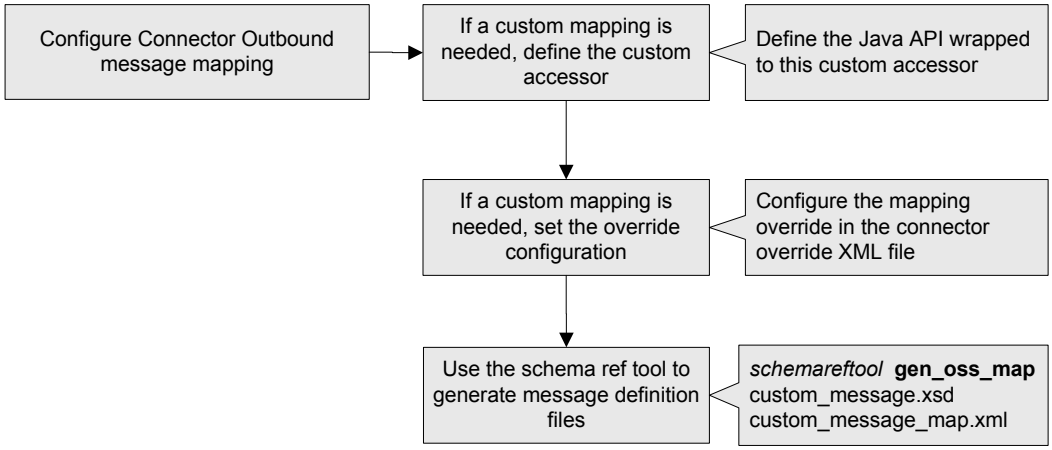
```
CUST_EVENTS=/<home_dir>/data/schemaref/biz/custom
SYNC_MAP=/<home_dir>/config/synchronizer/schema_map/acces sor
/<home_dir>/bin/schemareftool gen_sync_map
$CUST_EVENTS/reqaddservice_custom.xsd
$SYNC_MAP/reqaddservice_map.xml –schemaref_dir
<home_dir>/data/schemaref/biz/events
```

Configure Synchronizer Inbound Message Mapping



Every field of an inbound message is automatically mapped by the Synchronizer inbound transmapper default configuration. You do not have to perform any specific message field mapping configuration. The default behavior is to map leaves to java strings and nodes to hashtables.

Configure Connector Outbound Message Mapping



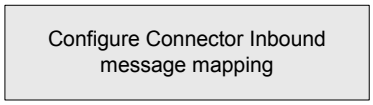
1. If you need to use custom java APIs to fill the content of your fields, you must define custom accessors. For each custom accessor used, you must declare the custom java API that will be wrapped to it in the Macro file `accessors_custom.properties` configuration file.
2. The Connector outbound mapping configuration for a specific message is automatically generated based on that message's structure definition file. If you want to customize a message field mapping, you must override the default mapping configuration of this field.

   You can override every message field mapping configuration by specifying the field name (and path) and the new accessor to be used in the XML override configuration file of your connector. For more information, refer to *To override a connector default mapping configuration*
3. Once you have configured the field mapping override, you must generate the message mapping definition files for your connector. Use the `gen_oss_map` command of the schemareftool. You must provide the schemaref tool with the following parameters:
   - (mandatory) custom message definition (full path to your XSD file)
   - (mandatory) destination message mapping definition file (XML file). This file should be located in `<home_dir>/config/yourconnector/schema_map/accessor`
   - (optional and multiple) mapping override configuration files (XML file). These files should be located in `<home_dir>/data/connectors/`

   Example:

```
CUST_EVENTS=/<home_dir>/data/schemaref/biz/custom
LPBK_MAP=/<home_dir>/config/loopback/schema_map/accessor
/<home_dir>/bin/schemareftool gen_oss_map
$CUST_EVENTS/reqaddservice_custom.xsd
$LPBK_MAP/reqaddservice_map.xml
```

Configure Connector Inbound Message Mapping



Every field of an inbound message will be automatically mapped by your Connector inbound transmapper default configuration. You do not have to configure any specific message field mapping. The default behavior is to map leaves to java strings and nodes to hashtables.

Working with Synchronizer Outbound Message Fields

You can activate or deactivate every field of the standard messages generated by the Synchronizer in order to suit your back end systems requirements.

When the standard messages do not contain enough information to suit your needs, or, on the contrary you want to reduce message contents, you might need to activate or deactivate certain message fields.

For each field of a message sent by the Synchronizer you can choose one of the following options:

- Activate the message field: The field will be mapped with its configured accessor and may be present in the Synchronizer outbound message depending on the accessor context.
- Deactivate the message field: In this case you have two options:
  - do **not** to generate the message field
  - generate an empty XML field (`<yourfieldname/>`).

To deactivate or activate a field

1. Go to `data/schemaref/biz/events/activation`.
2. Open the XML file corresponding to your message.
3. Locate the field you want to deactivate in the XML file.
4. If you want to deactivate a message field, that is to say to generate an empty XML field, add the following attribute:

   ```
   <yourfieldname tag="empty"/>
   ```

5. If you want to to deactivate a message field but **not** to generate the XML field, add the following XML attribute:

   ```
   <yourfieldname tag="null"/>
   ```

   To Activate a field:

   If you want to activate a field, you have two options. Either:

   - Do not specify any “tag” attribute, or
   - Do not add your field in the activation file

6. In order to apply your changes, you must use the schemarefTool with the `gen_sync_map` command to generate your Synchronizer outbound message mapping configuration. Specify the XML file you created in step one as the last parameter (`-activation_file`) of the `gen_sync_map` command.

Customizing Synchronizer Inbound Messages

You can customize a Synchronizer inbound message by adding fields (leaves or nodes). These new fields are automatically mapped as strings in the message hash table.

If you have defined additional sections (for example in the userarea) in the Synchronizer inbound messages, you might need to check some fields' values within these sections in order to perform specific message routing.

You can retrieve and use your custom fields' values through the workflow APIs, for example: "`Message.getElement`".

To use additional fields in Synchronizer inbound messages

1. Create a new script engine.
2. Insert the component in the inbound XML parser after the first Transmapper.
3. Create a new Script.
4. In the Script, use the API `Message.getElement()` to retrieve your value
5. Use the value retrieved to (for example):
   - route the message to another SmartLink Framework component
   - call an external API

Customizing Standard Message Structures

The Synchronizer uses messages to exchange data. These messages use the XML format, from the XML W3C specification, and consist of nodes and leaves. If necessary, you can add leaves or nodes to a message and add their respective mappings in order to fill them.

The nodes and leaves from the standard TSM messages are organized as follows:

- Root element (<request>)
  - Control area (<cntrolarea>)
    - Business Service Request area (<BSR>)
    - Sender area (<sender>)
    - Date and time area (<datetime>)
  - Data area
  - User Area

Only the Data area and the User area can be customized as follows:

For the Data area, only XML nodes that contain a dedicated “custom” tag can be modified. Within these “custom” tags, you can add any node or leaf you may need.

The additional parameter description is also put in this dedicated section.

For the User area, you can add any node or leaf you may need, this area is dedicated to your requirements.

You may need to add specific nodes or leaves in the User area. For example when you have to provide back-end systems with specific routing information, such as a specific system identifier.

If you have added additional parameters to standard TSM objects, the structure of these additional parameters is already defined. You do not have to define the structure, either for outbound messages or for inbound messages.

To add a custom field

1. Open the message structure description in the message schema reference
2. Add your nodes or leaves in the custom sections of objects as necessary:
   - in the data area
   - in the user area

   You can use the standard include mechanisms of the W3C XML Schema.
3. Save your description in a new file

   We recommend you **do not** to replace the standard message definition
4. Run the convert tool by specifying your previously created file as the input file for the tool

Generating the Message Schema Reference Files

You use the `schemarefTool` to generate the XML message definition files from the Message Schema reference.

You use this tool to generate the following from the Message Schema Reference:

- Message structure definition files
- Message documentation files
- Message mapping definition files for Synchronizer connectors
- Message mapping definition files for OSS Connectors

To generate message structure definition files

1. Go to `<ils_dir>/bin`.
2. Run the `schemarefTool` tool. Use the syntax:

   ```
   schemarefTool gen_msg_ref <xsd_source_file>
   <xsd_destination_file> [-quiet]
   ```

   where:

   `<xsd_source_file>` is the XML schema file

   `<xsd_destination_file>` is the destination XML structure definition file

To generate message documentation files

1. Go to `<ils_dir>/bin`.
2. Run the `schemarefTool` tool. Use the syntax:

   ```
   schemarefTool gen_msg_doc <xsd_src_file> <xsd_dest_struct> [-
   activation_file <file>] [-quiet]
   ```

   where:

   `<src_file>` is the XML schema file

   `<dest_struct>` is the destination XML structure definition file (without includes)

   `-activation_file<file>` is the file containing activation configuration

To generate message mapping definition files for Synchronizer connectors

1. Go to `<ils_dir>/bin`.

2 Run the `schemarefTool` tool. Use the syntax:

```
schemarefTool gen_sync_map <xsd_src_file> <mapping_dest_file>
[-schemaref_dir<dir>] [-activation_file <file>] [-quiet]
```

where:

`<xsd_src_file>` is the XML schema file

`<mapping_dest_file>` is the destination XML mapping definition file

`-schemaref_dir<dir>` is the directory containing core message definitions

`-activation_file<file>` is the file containing activation configuration

To generate message mapping definition files for OSS Connectors

1 Go to `<ils_dir>/bin`.

2 Run the `schemarefTool` tool. Use the syntax:

```
schemarefTool gen_oss_map <xsd_source_file>
<mapping_destination_file> [-core_override <file> ] [-
core_override_dir <dir>] [ -custom_override <file<,file>> ] [-
quiet]
```

where:

`<xsd_source_file>` is the XML schema file

`-core_override<file>` is the xml file defining default mapping

`-core_override_dir<dir>` is the directory where the core_override files are located

`–custom_override <file<,file>>` is a comma separated list of custom mappings

CHAPTER 8

Generating Your Connector

In This Section

About Generating Your Connector 144
To generate connectors..... 145
To generate connectors using the Connector Generator 146

About Generating Your Connector

The ILS uses the Integration Logic Connector Design file (`<connector_name>.ilcd`) to store the design and configuration of your connector. This file is for configuring and designing connectors. In order to run your connector, you need to generate the runtime files.

The generated connector runtime files are:

- Integration Logic Connector Configuration file (`<connector_name>.ilcc`)

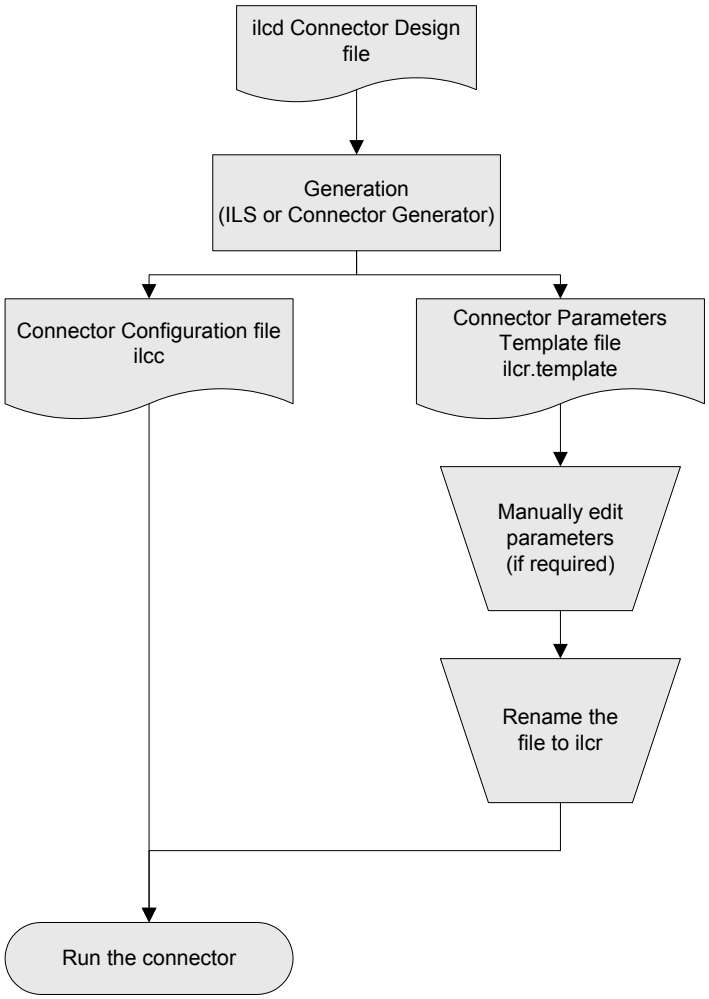
  This file is an internal configuration file used by the connector. It contains the configuration of your connector.

  This file is automatically generated and should not be edited. Use the ILS to change your configuration.

- Integration Logic Connector run time configuration file template (`<connector_name>.ilcr.template`)

  This run time configuration file contains the parameters defined for your connector. You edit this file manually and change its name.

The Generation process is as follows:



To generate the runtime files, you can:

- Use the ILS
- Use the Connector Generator command line tool

To generate connectors

1. Choose *Connector* > *Generate*. The Generate Connector Configuration dialog box appears.
2. If you want the ILS to validate the connector before generation, select *Perform Validation before Generation.*
3. In Output directory, enter the location of the generated connector files.
4. Choose *Generate*. The ILS generates the following connector runtime files:
   - `<connector_name>.ilcc`

- <connector_name>.ilcr.template

5 When finished, the ILS displays a message in the message pane.

6 If you need to change the runtime parameters before running your connector, open <connector_name>.ilcr.template and edit the parameters.

7 Change the file name to <connector_name>.ilcr.

To generate connectors using the Connector Generator

1 Go to <ils_dir>/bin.

2 Run the ilccGenerator tool. Use the syntax:

```
ilccGenerator  <connector_name>
```

where <connector_name> is the full path of the Integration Logic Connector Design .ilcd file

The tool generates the the following runtime files in the same directory as the Integration Logic Connector Design .ilcd file.

- <connector_name>.ilcc
- <connector_name>.ilcr.template

3 If you need to change the runtime parameters before running your connector, open <connector_name>.ilcr.template and edit the parameters.

4 Change the file name to <connector_name>.ilcr.

CHAPTER 9

Running Your Connector

In This Section

About Running the Connector 148
Specifying Processing Priority 149
Starting and Stopping the Synchronizer Connector..... 150
Starting and Stopping the OSS Connector 151
Administrating the Connector 152

About Running the Connector

Running your Connector involves:

- Specifying the processing priority of messages
- Starting and stopping the Synchronizer connector
- Starting and stopping the OSS connector
- Administrating connectors

Specifying Processing Priority

You can assign the priority of messages the Connector handles.

The role of the user generating the request in the CID can be configured to determine the priority of processing by the Connector. For instance, your connector can be configured to process requests created by Dealers before handling requests made by guest users.

For more information about users and roles, refer to *Managing Access to BLM Objects* in the *BLM Reference Guide*.

To assign a priority to a role

In the `ACTION_PRIORITY` table, add a new record then enter the following mandatory information:

- `ACTION_CODE` This number is the action. This code comes from the `ACTION` table.
- `ROLE_ID` The number of the role to assign the priority.
- `PRIORITY_ID` The number of the priority to assign. This ID comes from the `PRIORITIES` table.

Starting and Stopping the Synchronizer Connector

You use a set of administration tools to start and stop the Synchronizer connector. The administration tools are:

- `agentstart`
- `agentadm`

These administration tools are located in `<home_dir>/bin`.

About Synchronizer Run Modes

The Synchronizer connector works in one of the following modes:

MODE	DESCRIPTION
Normal	The default mode in which the Synchronizer scans the CID at a regular, fixed time intervals and extracts the requests
Paused	The Synchronizer connector stops scanning the CID for requests, and waits for a restart command before it resumes
Recover	The Synchronizer connector makes a final attempt to send requests (with the status 'TransportFailed') that have not been sent due to a transport problem. When finished, it goes back to the normal mode.

To start the Synchronizer connector

1. Go to `<home_dir>/bin`.
2. Run `agentstart`. Use the syntax:

   ```
   agentstart <connector_name>
   ```

   The connector is loaded and it starts the processes it needs.

To stop the Synchronizer connector

1. Go to `<home_dir>/bin`.
2. Run `agentadm`. Use the syntax:

   ```
   agentadm <host> <port> shutdown
   ```

   When finished, it displays a message.

Starting and Stopping the OSS Connector

You use a set of administration tools to start and stop the OSS connector. The administration tools are:

- `ossstart`
- `ossadm`

These administration tools are located in `<home_dir>/bin`.

To start the OSS connector

1. Go to `<home_dir>/bin`.
2. Run `ossstart`. Use the syntax:

   ```
   ossstart <connector_name>
   ```

   The connector is loaded and it starts the processes it needs.

To stop the OSS connector

1. Go to <home_dir>/bin.
2. Run `ossadm`. Use the syntax:

   ```
   ossadm <host> <port> <connector> shutdown
   ```

   When finished, it displays a message.

Administrating the Connector

You can stop, start and manage the connector interactively.

You use a set of administration tools to start, stop and manage the Synchronizer and OSS Connector.

About Administrating Connectors

To administer the connectors, you have a complete set of commands you can use to start, stop, or change the settings of the connector while it runs.

Whenever you send a command, the connector responds confirming the execution of your command. For example, you may see:

```
AGENTADM LOCALHOST 3000 STOP
-200AGENT PAUSED
AGENTADM LOCALHOST 3000 START
-200AGENT RESUMING TO NORMAL MODE
AGENTADM LOCALHOST 3000 POSE
-220UNKNOWN COMMAND POSE
```

To manage the Synchronizer connector

1. Log into the computer
2. Do one of the following:
   - If the connector is running, run `agentadm`
   - If the connector is not running, run `agentstart <synchronizer_name>`, then `agentadm`
3. Use one of the commands below to manage the Synchronizer connector:

COMMAND	DESCRIPTION	PARAMETERS	RETURN VALUES
start	Resumes the execution of the connector. Use this command to restart the connector after a stop command.	none	-200 AGENT START
stop	Stops the connector. The connector no longer processes inbound and outbound messages	none	-120 AGENT STOP
shutdown	Terminates the connector	none	-100 AGENT SHUTDOWN

COMMAND	DESCRIPTION	PARAMETERS	RETURN VALUES
setparameter	Changes the settings of the remotely. Use the following syntax: setparameter <parameter>=<value>	AGENTNAPPING AGENTLOOPING ONE_EXECUTION NBRROW QFILL MAXNUMBERRETRY RETRYDELAY	-510 AGENT SET<parameter name>=<parameter value>
getparameter	Restarts the current value of the parameter	AGENTNAPPING AGENTLOOPING ONE_EXECUTION NBRROW QFILL MAXNUMBERRETRY RETRYDELAY Any parameter defined in the properties files	-610 AGENT GET<parameter name>=<parameter value>
getmode	Displays the current running mode	none	-210 AGENT NORMAL -220 AGENT REDUCE -230 AGENT PAUSE
getstatus	Displays the current state	none	-130 AGENT NORMAL -140 AGENT REDUCE -150 AGENT PAUSE
force	Forces the connector to run in the specified mode	NORMAL REDUCE RECOVER	-410 AGENT SET MODE=NORMAL -410 AGENT SET MODE=REDUCE -410 AGENT SET MODE=RECOVER" For MODE=RECOVER, an additional parameter must be provided: FILTER <FilterName>
info	Displays information about the connector	none	[AGENTNAME:xxx,NBTHREADS:xxx ,FILL:xxx,NBROW:xxx,NAPPING: xxx,LOOPING:xxx,VERSION: xxx, BUILD:xxx,OS:xxx]"
kill	Forces shutdown	none	none
list	Returns all the profiling information	none	none
purge	Purges the profiling information	none	none
stat	Returns the profiling information of the last profiling element	none	none
version	Returns the version of the connector	none	none

To manage the OSS connector

1. Log into the computer
2. Do one of the following:
   - If the connector is running, run `ossadm`
   - If the connector is not running, run `ossstart <synchronizer_name>`, then `ossadm`
3. Use one of the commands below to manage the OSS connector:

COMMAND	DESCRIPTION	PARAMETERS	RETURN VALUES
start	Resumes the execution of the connector. Use this command to restart the connector after a stop command.	none	-200 AGENT START
stop	Stops the connector. The connector no longer processes inbound and outbound messages	none	-120 AGENT STOP
shutdown	Terminates the connector.	none	-100 AGENT SHUTDOWN
kill	Forces shutdown.	none	none

APPENDIX A

SmartLink Framework Tool Reference

In This Section

About the SmartLink Framework Administration Tools 156
Synchronizer Connector Administration Tools 157
OSS Connector Administration Tools 160
SmartLink Framework Administration Tool 161
Message Schema Reference Tool 162
Connector Generator Tool 165

About the SmartLink Framework Administration Tools

A complete set of administration tools helps you manage the different components of your TSM application.

The name of the files depends on your operating system. For Windows, the tools have a `.cmd` extension and for UNIX a `.sh` extension. The behavior of these tools is identical and they have the same command line options.

These administration tools are located in `<home_dir>/bin`.

Make sure that any required files are in the `CLASSPATH` and `LIB_PATH` specified in the `pre.env` environment setting file. This file is located in `<home_dir>/bin/env`.

Synchronizer Connector Administration Tools

You use the following administration tools to administrate the Synchronizer Connector:

- `agentstart`
- `agentadm`

These tools are located in `<home_dir>/bin`.

agentstart Syntax

```
agentstart <connector_name>
```

PARAMETERS	DESCRIPTION
<connector_name>	Name of the synchronizer connector to start

agentadm Syntax

```
agentadm help | <host> <port> <command> [<parameters>]
```

PARAMETERS	DESCRIPTION
help	Displays help for the tool
<host>	Specifies the agent host
<port>	Specifies the agent administration port
<command>	Administration command

COMMAND	DESCRIPTION	PARAMETERS	RETURN VALUES
start	Resumes the execution of the connector. Use this command to restart the connector after a stop command.	none	-200 AGENT START
stop	Stops the connector. The connector no longer processes inbound and outbound messages	none	-120 AGENT STOP

COMMAND	DESCRIPTION	PARAMETERS	RETURN VALUES
shutdown	Terminates the connector	none	-100 AGENT SHUTDOWN
setparameter	Changes the settings of the connector remotely. Use the following syntax: setparameter <parameter>=<value>	AGENTNAPPING AGENTLOOPING ONE_EXECUTION NBRROW QFILL MAXNUMBERRETRY RETRYDELAY	-510 AGENT SET<parameter name>=<parameter value>
getparameter	Restarts the current value of the parameter	AGENTNAPPING AGENTLOOPING ONE_EXECUTION NBRROW QFILL MAXNUMBERRETRY RETRYDELAY Any parameter defined in the properties files	-610 AGENT GET<parameter name>=<parameter value>
getmode	Displays the current running mode	none	-210 AGENT NORMAL -220 AGENT REDUCE -230 AGENT PAUSE
getstatus	Displays the current status	none	130 AGENT NORMAL -140 AGENT REDUCE -150 AGENT PAUSE
force	Forces the connector to run in the specified mode	NORMAL REDUCE RECOVER	-410 AGENT SET MODE=NORMAL -410 AGENT SET MODE=REDUCE -410 AGENT SET MODE=RECOVER For MODE=RECOVER, an additional parameter must be provided: FILTER <FilterName>
info	Displays information about the connector	none	[AGENTNAME:xxx,NBTHREADS:xxx,FILL:xxx,NBROW:xxx,NAPPING:xxx,LOOPING:xxx,VERSION:xxx,BUILD:xxx,OS:xxx]
kill	Forces shutdown	none	none
list	Returns all the profiling information	none	none
purge	Purges the profiling information	none	none
stat	Returns the profiling information of the last profiling element	none	none
version	Returns the version of the connector	none	none

OSS Connector Administration Tools

The tools you use to administrate the OSS connector are:

- `ossstart`
- `ossadm`

ossstart Syntax

```
ossstart help | <connector_name>
```

PARAMETERS	DESCRIPTION
`<connector_name>`	Name of the OSS connector to start

ossadm Syntax

```
ossadm help | <host> <port> <command>
```

PARAMETERS	DESCRIPTION
help	Displays help for the tool
`<host>`	Specifies the connector host
`<port>`	Specifies the connector port
`<command>`	Administration command

COMMAND	DESCRIPTION	PARAMETERS	RETURN VALUES
`shutdown`	Terminates the connector	none	none
`kill`	Forces shutdown	none	none

SmartLink Framework Administration Tool

You use the `isfadm` tool to manage the SmartLink Framework.

isfadm Syntax

```
isfadm help | <host> <port> <objecttype> <object> <command>
```

PARAMETERS	DESCRIPTION
`help`	Displays help for the tool
`<host>`	Specifies the host to use.
`<port>`	Specifies the administration port to use
`<objectype>`	The SmartLink Framework component to manage: connector queue var
`<object>`	The object of the command.
`<command>`	Administration command

COMMAND	DESCRIPTION	PARAMETERS	RETURN VALUES
`start`	Resumes the execution of the component.	none	
`stop`	Stops the component.	none	
`getstatus`	Displays the current status	none	

Message Schema Reference Tool

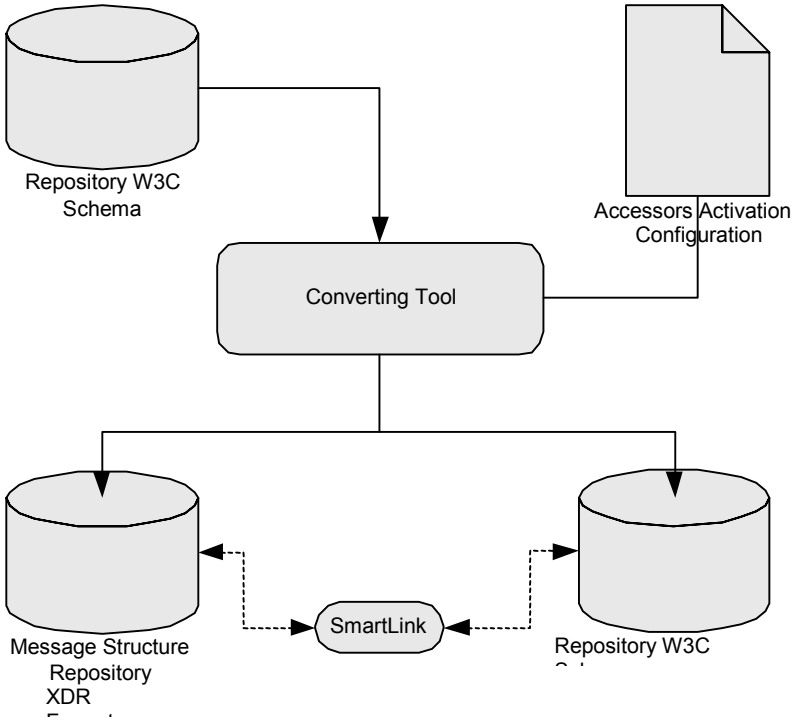
This tool handles the Message Schema Reference. This tool handles the following commands:

- `Gen_msg_ref`: Generates the message structure definition files
- `Gen_msg_doc`: Generates the message documentation files
- `Gen_synch_map`: Generates the message mapping definition files for the Synchronizer
- `Gen_oss_map`: Generates the message mapping definition files for the Connectors

You use the converting tool to:

- Transform message structure and/or mapping definitions from the W3C format into the XDR format.
- Apply the accessors' activation configuration (specified by the user).

If no configuration is available, all the accessors are generated.



schemarefTool Syntax

```
schemarefTool help | <command> [<parameters>] <help>
```

PARAMETERS	DESCRIPTION
help	Displays help for the tool
<command>	Message Schema Reference Tool command
<parameters>	Specifies the parameters for the command
<command> help	Displays help for the command

COMMAND	DESCRIPTION	PARAMETERS
gen_msg_ref	generates the message structure definition files	<xsd_source_file>: Source file containing the XML Schema definition of the message <xsd_destination_file>: Destination file in which the structure definition in old XML format will be saved -acitivation_file<file> (Optional) File containing the activation configuration for the file being processed <-quiet>: This optional parameter can be added to activate/deactivate detailed logs during the execution of the tool.
gen_msg_doc	Generates the message documentation files	<xsd_source_file>: Source file containing the XML Schema definition of the message <xsd_destination_file>:Destination file (full path) in which the flat (without any include) structure definition in XSD format will be saved <activation>: (Optional) File containing the activation configuration for the file being processed <-quiet>: This optional parameter can be added to activate/deactivate detailed logs during the execution of the tool.

COMMAND	DESCRIPTION	PARAMETERS
gen_sync_map	Generates the message mapping definition files for the Synchronizer	<xsd_source_file>: Source file containing the XML Schema definition of the message <mapping_destination_file>: Destination file in which the mapping generation in the old XML format will be saved -schemaref_dir<dir> (Optional) Directory containing the core message definitions -activation<file>: (Optional) File containing the activation configuration for the file being processed <-quiet>: (Optional) Parameter that can be added to activate/deactivate detailed logs during the execution of the tool.
gen_oss_map	Generates the message mapping definition files for the Connectors	<xsd_source_file>: Source file containing the XML Schema definition of the message <mapping_destination_file>: Destination file in which the mapping generation in the old XML format will be saved -core_override<file> (Optional) File containing the xml file defining the default mapping -core_override_dir<dir> (Optional) Directory containing the core_override files are located -custom_override_dir<dir> (Optional) Directory containing the custom_override files are located -custom_override<file<,file>: (Optional and multiple) File containing the mapping override configuration for this message. Several files can be specified and must be separated with a comma. Each file will be applied in the same order as it is specified. When you use multiple files on Windows platforms, the entire list of files must be enclosed in inverted commas ("file1,file2...") <-quiet>: This optional parameter can be added to activate/deactivate detailed logs during the execution of the tool.

Connector Generator Tool

This tool generates the following connector runtime files from the Integration Logic Connector Design (`<connector_name>.ilcd`) file:

- Integration Logic Connector Configuration file (`<connector_name>.ilcc`)
- Integration Logic Connector Run time configuration file (`<connector_name>.ilcr`)

ilccGenerator Syntax

```
ilccGenerator -help | [-check] [-verbose] [-dir <output
directory>] <ILCD file>
```

PARAMETERS	DESCRIPTION
help	Displays help for the tool
check	Checks the validity of the ilcd file before generating
verbose	Displays all generation messages
dir <output directory>	Specifies the destination directory if other than the location of the ilcd file
<connector_name>	The ilcd file to use to generate the connector runtime files

APPENDIX B

Core Messages

The following table lists available core messages:

MESSAGE NAME	VERB	NOUN
ack.xml	Acknowledge	request
doaddbillingaccount.xml	doadd	billingaccount
doaddcontact.xml	doadd	contact
doaddcontract.xml	doadd	contract
doaddlevel.xml	doadd	level
doaddlogin.xml	doadd	login
doaddmember.xml	doadd	member
doaddorg.xml	doadd	org
doaddservice.xml	doadd	service
doaddtroubleticket.xml	doadd	troubleticket
doassociatededicatedoffer.xml	doassociate	offer
dodeclarepaymentresp.xml	dodeclare	paymentresp
dodissociatededicatedoffer.xml	dodissociate	offer
domigratecontract.xml	domigrate	contract
domodifyaddress.xml	domodify	address
domodifybillingaccount.xml	domodify	billingaccount
domodifycontact.xml	domodify	contact
domodifycontract.xml	domodify	contract
domodifycontractline.xml	domodify	contractline
domodifycontractowner.xml	domodify	contractowner
domodifycontractstatus.xml	domodify	contractstatus
domodifylanguage.xml	domodify	language
domodifylevel.xml	domodify	level
domodifymember.xml	domodify	member
domodifyorg.xml	domodify	org
domodifypaymentinfo.xml	domodify	paymentinfo
domodifyrateplan.xml	domodify	rateplan
domodifyservice.xml	domodify	service
domodifytroubleticket.xml	domodify	troubleticket
doossapproval.xml	doossapproval	approval
doremoveservice.xml	doremove	service

MESSAGE NAME	VERB	NOUN
doreplaceservice.xml	doreplace	service
dosetaddinfo.xml	doset	addinfo
dosetpersonalinfo.xml	doset	persoinfo
notifychange.xml	notify	change
reqaddbillingaccount.xml	reqadd	bilaccount
reqaddbillingcontact.xml	reqadd	bilcontact
reqaddcontract.xml	reqadd	contract
reqaddlegalcontact.xml	reqadd	legalcontac
reqaddlevel.xml	reqadd	level
reqaddlogin.xml	reqadd	login
reqaddmember.xml	reqadd	member
reqaddorg.xml	reqadd	org
reqaddorgcontact.xml	reqadd	orgcontact
reqaddservice.xml	reqadd	service
reqaddtroubleticket.xml	reqadd	trblticket
reqassociatededicatedoffer.xml	reqassociate	dedicatedoffer
reqdeclarepaymentresp.xml	reqdeclare	paymentres
reqdissociatededicatedoffer.xml	reqdissociate	dedicatedoffer
reqlostdeclaration.xml	reqdeclare	lostthieft
reqmigratecontract.xml	reqmigrate	contract
reqmodifybillingaccount.xml	reqmodify	billingaccount
reqmodifybillingaddress.xml	reqmodify	biladdress
reqmodifybillingcontact.xml	reqmodify	bilcontact
reqmodifycontract.xml	reqmodify	contract
reqmodifycontractline.xml	reqmodify	contractline
reqmodifycontractowner.xml	reqmodify	contractowner
reqmodifycontractstatus.xml	reqmodify	contractstatus
reqmodifylanguage.xml	reqmodify	language
reqmodifylegalcontact.xml	reqmodify	legalcontact
reqmodifylevel.xml	reqmodify	level
reqmodifymember.xml	reqmodify	member
reqmodifyorg.xml	reqmodify	org
reqmodifyorgaddress.xml	reqmodify	orgaddress
reqmodifyorgcontact.xml	reqmodify	orgcontact
reqmodifypaymentinfo.xml	reqmodify	paymentinfo
reqmodifyrateplan.xml	reqmodify	rateplan
reqmodifyservice.xml	reqmodify	service
reqmodifytroubleticket.xml	reqmodify	trblticket

MESSAGE NAME	VERB	NOUN
reqorder.xml	reqorder	composite
reqorderdoc.xml	reqorder	doc
reqrechargeprepaid.xml	reqrecharge	prepaid
reqremoveservice.xml	reqremove	service
reqreplaceservice.xml	reqreplace	service
reqsetpersonalinfo.xml	reqset	persoinfo
system.xml	system	system

APPENDIX C

Core Business Objects

In This Section

About Core Business Objects 172
About Parameters and Messages 173

About Core Business Objects

This list presents the core Business objects. It is a list of the Business Objects which support additional parameters:

- BillingAccount
- Organization
- Organization category type
- Language
- Role
- Level
- Member
- Service
- Contract
- Rate plan
- Line
- Line Type
- Country
- Title
- Sex
- Contact
- Trouble ticket
- Trouble Ticket category
- Document
- Invoice
- Prepaid package
- Commercial Offer

About Parameters and Messages

The SmartLink Framework is configured to manage the additional parameters of every object that can have additional information.

Outbound Messages

If the object is on the list of supported objects, the SmartLink Framework is ready to handle additional parameters. If the object can support additional parameters, but has this feature inactivated, refer to *Working With the Message Schema Reference*.

Inbound Messages

The inbound messages come with a special section that deals with the additional information of all supported objects. In a message, this dedicated section manages updating all additional information.

The additional information in this section does not necessarily need to be related to the other objects being modified. This feature allows you to update additional information of unrelated objects when your application requires an update in a single transaction.

The doSetAddInfo message contains modifications to the additional parameters of each object that supports additional information. This message has a field for each object that supports additional information and is used to update additional information only.

Index

A

Additional Parameters
  list of objects • 171
Administration tools
  agentadm • 150, 157
  agentstart • 150, 157
  ossadmin • 151
  ossstart • 151
agentadm Administration Tool
  syntax • 157
  using • 150
agentstart Administration Tool
  syntax • 157
  using • 150

B

BLM/XML Transmapper
  about • 54
  configuring • 66
  properties • 66
  specifying request types • 66

C

CID Objects Builder
  about • 54
  configuring • 66
  properties • 67
  specifying message types • 67
  specifying the schema definition files • 67
CID Update Error Handler
  about • 57
  configuring • 76
  properties • 76
Connecting
  about • 78
  connecting • 79
  disconnecting • 79
  types • 78
Connector Configuration File (ilcc)
  about • 37
  generating • 144
  location • 37
Connector Design File (ilcd)
  about • 37
  generating • 144
  location • 37
Connector Parameters File (ilcr)
  about • 37
  generating • 144
  location • 37
Connector Template
  about • 39
  copying • 39
  designing • 42
  developing • 32
  renaming • 39
  selecting • 39
Connectors
  about • 19, 33
  administrating • 152
  and Integration Processes • 34
  and Macros • 35
  and Scripts • 36
  components of • 33
  creating Integration Processes • 46
  customizing • 45
  designing • 42
  developing • 32
  generating • 144
  opening • 49
  program files • 37
  running • 150
  running OSS Connector • 151
  validating • 49

E

Encoding Converter
  about • 57
  configuring • 76
Error Handler

about • 57
configuring • 76
properties • 76
Exception Converter
about • 57
configuring • 76
properties • 76
Extensions
about • 99
configuring • 99
creating • 99
deleting • 100
sample inbound queue • 112
sample outbound queue • 111

F

Filters
activating • 65
setting values of • 65
specifying • 64

H

Help
technical support • xi

I

Inbound Email Queue
about • 56
configuring • 71
properties • 71
Inbound File System Queue
about • 56
configuring • 71
properties • 71
Inbound JMS Queue
about • 56
configuring • 72
properties • 72
Inbound JMS Topic Queue
about • 56
configuring • 72
properties • 72
Inbound Socket Queue
about • 56
configuring • 72
properties • 72
Installing
Integration Logic Studio • 28
SmartLink Framework components • 28
uninstalling • 30
Integration Logic Studio
installing • 28
Integration Processes
about • 34
and Processors • 52
connecting • 79
copying • 51
creating • 50
deleting • 51
disconnecting • 79
opening • 50
overview • 46
refreshing • 51
renaming • 51
SmartLink Framework
about • 18
architectures • 22
components • 18
connectors • 19
integration services • 18
message schema reference • 20
OSS Connectors • 19
Synchronizer Connector • 19
SmartLink Framework Administration Tools
about • 156
agentadm administration tool • 157
agentstart administration tool • 157
isfadmin administration tool • 161
location • 156
ossadm administration tool • 160
ossstart administration tool • 160
SmartLink Framework Message Schema Reference
about • 118
schemarefTool • 162, 163
SmartLink Framework Messages

about • 118, 119, 120
activating fields • 137
adding fields • 131, 132, 133, 134, 135, 136, 138, 139, 140
business data area • 120
control area • 119
creating new messages • 121, 122, 123, 125, 126, 127, 128
list of • 167
overriding default mapping • 129
schemarefTool • 141

J

Java/XML Transmapper
about • 54
configuring • 68
specifying request types • 68

M

Macro Source File
about • 91
calling static methods • 93
creating • 91
operators • 93
programming • 92
specifying • 91
Macros
about • 89
connector macro • 89
creating • 89
deleting • 90
Integration Process macro • 89
location • 37, 89
opening • 90
programming • 92
source file • 91
specifying macros • 91
Message Broadcaster
about • 57
configuring • 69
Message Retry Counter
about • 57
configuring • 75
properties • 75
Message Schema Reference
about • 116, 117
location • 37
Message Type Filter
about • 55
configuring • 69
specifying filters • 69
Multi Queue Message Tracker
about • 56
MultiQueue Message Broadcaster
about • 55
configuring • 70
specifying connection properties • 70
MultiQueue Message Filter
about • 55
configuring • 70
specifying filter properties • 70
specifying filters • 70

O

OSS Connectors
about • 19, 44
administrating • 154
developing • 32
generating documentation • 49
opening • 49
overview • 19, 44
running • 151
saving • 49
stopping • 151
validating • 49
ossadmin Administration Tool
using • 151
ossstart Administration Tool
using • 151
Outbound Email Queue
about • 56
configuring • 73
properties • 73
Outbound File System Queue
about • 56
configuring • 73
properties • 73
Outbound JMS Queue
about • 56
configuring • 74
properties • 74
Outbound JMS Topic Queue
about • 56
configuring • 74
properties • 74
Outbound Socket Queue

about • 56
configuring • 74
properties • 74

P

Pause
about • 57
configuring • 76
properties • 76
Processors
about • 52
creating • 59
customizing properties • 59
deleting • 60
editing • 59
Error Processors • 57
Message Queue Processors • 56
renaming • 60
Request Queue Processors • 53
Router Processors • 55
Transmapper Processors • 54
Utility Processors • 57

R

Request Queue Agent
about • 53
properties • 62
Sequences • 63
specifying request types • 62
specifying Sequences • 64
specifying Status Code • 63
Request Queue Processors
about • 53
Retry Handler
about • 57
configuring • 75
properties • 75
Roles
configuring • 149

S

schemarefTool Administration Tool
about • 141, 162
syntax • 163
using • 141, 142
Script Engine
configuring • 69
specifying scripts • 69
Script Source Files
creating • 81
programming • 82
Script Steps
about • 83
Call steps • 84
connecting • 86
creating • 83
deleting • 85
Init steps • 84
Invoke steps • 83
Switch steps • 85
Scripts
about • 81
and Connectors • 81
associating with source file • 81
closing • 82
creating • 81
creating source file • 81
deleting • 82
location • 37, 81
opening • 82
overview • 81
programming • 82
Sequences
activating • 63
specifying • 64
Starting
OSS Connector • 151
Synchronizer Connector • 150
Stopping
OSS Connector • 151
Synchronizer Connector • 150
Synchronizer Connectors
about • 19
administrating • 152
components • 43
configuring • 152
creating • 39
managing • 152
modes • 150
overview • 19, 43
renaming • 39
running • 150
starting • 150
stopping • 150, 157

T

Transmapper Processors

about • 54
BLM/XML Transmapper • 54
CID Objects Builder • 55
configuring • 66
Java/XML Transmapper • 54
XML Parser • 54

U

Utility Properties
about • 57
Encoding Converter • 57
Pause • 57

X

XML Parser
about • 54
configuring • 67
specifying request type properties • 68
specifying request types • 67