# Contents

**What’s New in SAS Business Orchestration Services 10.1** ........................................ v

**Chapter 1 / Getting Started with SAS Business Orchestration Services** ........................................ 1
  - Overview ........................................................................................................... 1
  - Business Orchestration Services Paths ......................................................... 2
  - Environment Setup ......................................................................................... 2
  - Start SAS Business Orchestration Services ............................................... 3
  - Verify That SAS Business Orchestration Services Is Running ...................... 3
  - Stop SAS Business Orchestration Services .................................................. 4
  - Configure SAS Business Orchestration Services ......................................... 4
  - Renewing Your Software License ................................................................... 5

**Chapter 2 / Transforming Messages** ............................................................................. 7
  - Inbound Data Transformation ....................................................................... 7
  - Data Transformation ....................................................................................... 12
  - Data Enrichment ............................................................................................. 17
  - Configuring Outbound Data Mapping ........................................................... 22

**Chapter 3 / Managing SAS Business Orchestration Services** ........................................... 25
  - Logging ........................................................................................................... 25

**Chapter 4 / Managing Data Flow** .................................................................................. 29
  - Store and Forward ......................................................................................... 29
  - Configuring Stand-In Response ..................................................................... 38
  - Configuring Communication between SAS Business Orchestration Services and SAS Fraud Management ......................................................................................................................... 43
  - Consuming SAS Fraud Management Messages on Downstream Queues ...... 46
  - Consuming Messages from Offline Files ..................................................... 47
  - Data Stores ..................................................................................................... 50

**Chapter 5 / SAS Business Orchestration Services Scalability and High Availability** .............. 53
  - Scalability and High Availability Overview .................................................. 53
  - Tuning Performance Vertically ...................................................................... 53
  - Horizontal Scale and High Availability ......................................................... 54

**Chapter 6 / SAS Business Orchestration Services Security** .................................................. 57
  - Security Overview .......................................................................................... 57
  - Configuring HTTPs .......................................................................................... 57
  - Configuring Basic Authentication ................................................................... 59
  - Configuring Secured Communication with IBM MQ .................................... 60
  - Configuring SSL with SAS OnDemand Decision Engine ............................. 61
  - Encrypting Credentials .................................................................................. 61

**Appendix 1 / Migrating to SAS Business Orchestration Services 10.1** ............................... 65
  - Migration Overview ....................................................................................... 65
  - Configuration Files ......................................................................................... 66
  - Remove Unwanted Configuration Information ............................................ 66
Upgrade Log4j to Log4j2 .......................................................... 66
Install ActiveMQ and Store and Forward Using ActiveMQ ........ 67
IBM MQ and Store and Forward Using IBM MQ ......................... 68
Dynamic Load Balancer ......................................................... 70
PostgreSQL Database .......................................................... 71
Oracle Database ..................................................................... 72
Redis .................................................................................... 72
Offline Files Processing .......................................................... 72
Kafka Message Queue ............................................................ 73
Custom Outbound Mapping ...................................................... 73
What’s New in SAS Business Orchestration Services 10.1

Overview

SAS Business Orchestration Services 10.1 now enables you to harness the power of third-party identity management data, giving you deeper insights into current and potential new customers. This added intelligence aids in reducing risk and customer friction.

You now have the option to add on the following adapters when you purchase SAS Business Orchestration Services. This list of vendors provides intelligence on personal data that is supplied by customers and consumers when they apply for credit or when they make transactions online. Financial Institutions are typically associated with at least one of the vendors listed. SAS Business Orchestration Services adds efficiency by only calling vendors when needed. This efficiency reduces vendor cost for customers.

- BioCatch
- Boku
- DataVisor
- Giact
- Intellicheck
- Iovation
- Payfone
- Socure
- ThreatMetrix

Each module is licensed and installed separately. Once the module has been installed, more information about how to use the APIs from these data providers and a sample configuration file can be found in the `$BOSS_SHARE/samples/<vendorname>` directory. For instructions on how to use the sample configurations, see the `ReadMe.md` file in each of the directories.

For more information about each vendor’s service, see each vendor’s website.
Overview

SAS Business Orchestration Services is highly customizable and enables you to manage message flow and transformation to and from other software systems that expose an API using any of the popular communication protocols. It can be deployed to tackle both real-time and batch processing scenarios. Here are the primary functions when managing the message flow:

- Transformation: Conversion of a message into a format that can be consumed by the target system.
- Enrichment: Adding information to the message that was fetched from other data sources.
- Support for multiple protocols: Acting as a bridge between systems that use different network protocols.
- Store and forward: Queueing messages if the target system is unavailable to receive messages and forwarding the messages once the target system is available.
- Stand-in response: Sending the last known good response for a message if the target system does not respond within a specified time-out period.

SAS Business Orchestration Services is a lightweight Java process that can be installed on hardware that has minimal resources. For highly available deployments,
multiple instances of SAS Business Orchestration Services can be deployed on separate hardware.

SAS Business Orchestration Services enables integration of two or more systems without any changes to the systems that are being integrated. Each system communicates using their existing protocols and using the message structures that it is capable of understanding. SAS Business Orchestration Services acts as the bridge between the systems to perform any necessary transformations and can communicate with each system using that system’s native capabilities.

SAS Business Orchestration Services provides data-driven routing capabilities where target systems and actions to take on each message can be determined based on the content of the message.

SAS Business Orchestration Services provides robust error handling mechanisms and transactional processing capabilities.

For more information about installing SAS Business Orchestration Services, see SAS Business Orchestration Services 10.1: Deployment Guide.

Business Orchestration Services Paths

Here are various default SAS Viya 3.5 paths and SAS Business Orchestration Services paths. If you have customized your installation path in any way in your environment, then use the modified paths that apply to your installation.

- Installation home directory: /opt/sas/viya/home (referred to as VIYA_HOME in this document)
- Configuration directory: /opt/sas/viya/config (referred to as VIYA_CONFIG in this document)
- Log directory: /opt/sas/viya/config/var/log (referred to as VIYA_LOG in this document)
- $BOSS_BIN directory: /opt/sas/viya/home/bin/boss
- $BOSS_CONFIG directory: /opt/sas/viya/config/etc/boss
- $BOSS_LIB directory: /opt/sas/viya/home/lib/boss
- $BOSS_LOG directory: /opt/sas/viya/config/var/log/boss
- $BOSS_SHARE directory: /opt/sas/viya/home/share/boss

For more information about configuring and installing SAS Business Orchestration Services on multiple nodes, see Chapter 5, “SAS Business Orchestration Services Scalability and High Availability,” on page 53.

Environment Setup

SAS Business Orchestration Services requires JDK 1.8. Set the environment variable JAVA_HOME to point to your JDK installation directory. Add the JDK
binaries to the PATH environment variable. The exact directory might vary from system to system. Check your local file system to be sure where Java is installed.

UNIX (bash/sh):
JAVA_HOME=/usr/local/java/jdk1.8.0_102; export JAVA_HOME
PATH=$JAVA_HOME/bin:$PATH; export PATH

UNIX (tcsh):
setenv JAVA_HOME=/usr/local/java/jdk1.8.0_102
setenv PATH $JAVA_HOME/bin:$PATH

Start SAS Business Orchestration Services

Once you have installed SAS Business Orchestration Services and have JDK set up, you can either migrate your previous version of SAS Business Orchestration Services configuration or copy any of the sample configurations in the $BOSS_SHARE directory to the $BOSS_CONFIG directory.

For more information about migrating, see Appendix 1, “Migrating to SAS Business Orchestration Services 10.1,” on page 65.

Copy your license file named setinit.txt into the $BOSS_CONFIG directory.

Start SAS Business Orchestration Services.

$BOSS_BIN/boss.sh start

Besides start, the following commands can also be used with boss.sh:

- help: explains how to use the boss.sh script
- status: shows the status of the process and its process ID
- stop: stops the process
- tail: sends the main process application log to the console

Verify That SAS Business Orchestration Services Is Running

Verify that SAS Business Orchestration Services is running using the following command:

$BOSS_BIN/boss.sh status
Stop SAS Business Orchestration Services

Stop SAS Business Orchestration Services using the following command:

$BOSS_BIN./boss.sh stop

One of the following messages is displayed:

- BOSS Process ID: <process ID>. Stopping it gracefully ...
  - BOSS shutdown complete (n seconds).
- BOSS is not running.

Configure SAS Business Orchestration Services

SAS Business Orchestration Services is written in Java and it uses an open-source Spring Framework and Apache Camel (version 2.24.0). Knowledge of Spring Framework with XML configuration files is highly recommended in order to use SAS Business Orchestration Services. Since Apache Camel is used as the execution engine in SAS Business Orchestration Services, knowledge of Apache Camel concepts and core components is required to use SAS Business Orchestration Services effectively.

For more information about the capabilities that are built into Apache Camel, see Apache Camel Getting Started.

For more information about Spring Framework, see Spring Framework Documentation. Spring Framework has numerous modules. At a minimum, you should be familiar with using Spring IoC container and Spring beans.

Here are the most commonly used Apache Camel components that are already packaged with SAS Business Orchestration Services:

- camel-groovy (to enable Groovy scripting and Groovy beans)
- camel-http4 (useful for calling HTTP or REST endpoints)
- camel-jasypt and camel-crypto (to encrypt data elements in configuration files or the message content)
- camel-jetty9 (useful for providing REST services)
- camel-jms (useful to communicate with message queue systems)
- camel-kafka
- camel-metrics (useful to gather route metrics)
- camel-netty4 (useful for socket-level communications)
- camel-rabbitmq
- camel-sql (useful to interact with relational databases)
For more information about how to configure and use the components, see the component-specific documentation. The $BOSS_SHARE directory contains samples for some of these components.

Most of the configuration that is needed for SAS Business Orchestration Services is done in XML files. Some configuration changes are applied to content mapping without the need to restart the service. Whenever any processing task gets too complex to express in XML, it is recommended that you encapsulate that logic in Groovy code and invoke it from the XML configuration file.

Renewing Your Software License

A valid license file is required to run SAS Business Orchestration Services.

For more information, see Licensing: How To Apply Licenses in SAS® Viya® 3.5 Administration.
Inbound Data Transformation

Inbound Data Transformation Overview

This chapter explains how customer transaction data can be mapped before it is routed to other software services or components like SAS Fraud Management. SAS Fraud Management is used as an example throughout this section.
One of the main functions of SAS Business Orchestration Services is to receive customer transaction requests in native customer data formats, process the requests, and return responses in customer-preferred formats. This customer-centric approach aims to introduce minimal code change to a customer’s existing software, shorten the integration time, and increase speed to market.

SAS Fraud Management requires messages to conform to the SAS Fraud Management specific data format. Incoming transactions from a client’s system are in a different data format from the SAS Fraud Management data format. Therefore, inbound data mapping is needed to convert a client’s transaction into a SAS Fraud Management transaction.

**Note:** In this document, SAS Fraud Management and its data format is being used as an example. The information about inbound and outbound data mapping generally apply to all software services.

There is a wide range of messaging standards that are used by financial institutions and for data exchange among institutions. Even with the emergence of international standards such as FIX, ISO20022, SWIFT, and FpML, many institutions still use their proprietary syntaxes and semantics. As a result, SAS Business Orchestration Services needs to handle inbound data transformation to accommodate customer-specific formats and standard formats.

SAS Business Orchestration Services supports the following common formats:

- Byte streams
- CSV
- JSON
- XML
- ISO8583
- SWIFT MT

Mapping takes place in two steps:

1. Map input data into request field name-value pairs.
2. Map input request field names to SAS Fraud Management field names.

### Mapping for XML Input

Many custom data formats and even some international standard formats such as ISO20022 use an XML format. XML data is self-descriptive. SAS Business Orchestration Services uses XPath and a string value represented by XPath to form name-value pairs. SAS Business Orchestration Services uses a subset of XPath functions to get name-value pairs from the input XML data.

To map the name-value pairs to a SAS Fraud Management transaction, SAS Business Orchestration Services needs additional information:

- Which part of the XML data maps to which SAS Fraud Management transaction field or fields.
- What are the default values for some of the SAS Fraud Management transaction fields.
- How to transform XML data (if needed).
- How to enrich data (if needed).

Sample mapping files can be found in `$BOSS_CONFIG/mappings/inbound/mapping_iso20022.xml`.

Here is part of this file.

```xml
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="https://www.sas.com/sfm/eop"
  xsi:schemaLocation="http://www.sas.com/sfm/eop mapping.xsd">
  <messages name="ISO 20022">
    <message messageType="PAIN" subType="001.001.03" mapperId="iso20022_01">
      <fields>
        <field name="TBT_GROUP_CNT">
          <srcField name="CstmrCdtTrfInitn/GrpHdr/NbOfTxs"/>
        </field>
        <field name="TBT_GROUP_AMT">
          <srcField name="CstmrCdtTrfInitn/GrpHdr/CtrlSum"/>
        </field>
        ...
        <field name="smh_tran_type" default="TRX"/>
        <field name="rqo_tran_time_alt" default="17:14:00"/>
      </fields>
    </message>
  </messages>
</application>
```

In this file, it is important to understand the following concepts:

- A piece of data identified by an XML pathname is mapped to a SAS Fraud Management field.
- A SAS Fraud Management field can be assigned a default value even when there is no corresponding input.
- Mapping files can be version controlled by using name, type, and subtype combinations.
- A unique mapperId can also be used to identify a mapping.

A unique mapperId is used to identify a mapping. With the initial mapping created and the above understanding of the mapping file, you should have no problem making small changes, such as the following:

- adding a new field mapping
- changing default values
- deleting an existing field mapping
- modifying an existing field mapping

The mapping files are monitored by SAS Business Orchestration Services at run time. Changes are picked up and applied at run time. Any mapping errors are written to log files, and the previously loaded mapping file continues to be used.
Mapping for JSON Input

JSON is another popular data format that might be used. Like XML, JSON is also self-describing to a certain degree, hierarchical, and easy to parse. JSON's popularity results from being more compact than XML.

The JSON hierarchical names are used for mapping an input data field to a SAS Fraud Management field.

The mapping file for JSON input is similar to what is used for XML input. In fact, if the JSON input has the same hierarchical names as that of XML input, then the mapping file is identical.

Mapping for CSV Input

For parsing input data in CSV format (using separators such as a comma, tab, tilde, or any other character string), a corresponding comma separated names (CSN) file needs to be configured in the SAS Business Orchestration Services. A CSN file tells SAS Business Orchestration Services about each field name in the incoming CSV input. CSN files need to be placed in the $BOSS_CONFIG/mappings/csv/ directory. You can also find sample CSN files in that directory.

The input CSV data is parsed using a CSN file to produce a field name to value pairs. If you are already familiar with the sample mapping file used in "Mapping for XML Input" on page 8, then you see that the names are used in `<srcField name="xxx"/>`. In the mapping for XML and JSON input, the XPath notation is used for the srcField name, but it is not a requirement imposed by the mapping file. Simple names work equally well in this case.

You need to create the mapping file with correct `<srcField name="xxx"/>` mapped to the SAS Fraud Management `<field name="xxx"/>`.

Here is an example of a CSN:

```
# This sample CSV mapping information contains a line for unique key for CSV formats, and
# another line for comma separated names.
# This is the CSV format for a ping request.
# The input CSV data could be: 04.03.08,1,CMD
# although the separator does not have to be a comma, it could be a - for example.
sample1
version,needResponse,tran type
```

For the sample CSN, here is an example corresponding mapping file:

```xml
<?xml version="1.0"?>
<!-- Sample mapping equivalent to a ping request. -->
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="https://www.sas.com/sfm/epc"
  xsi:schemaLocation="http://www.sas.com/sfm/epc mapping.xsd">
  <messages name="Ping">
    <message messageType="ping" subType="1.0" mapperId="ping">
      <fields>
        <field name="smh_msg_version" default="04.04.02">
```
If you are not familiar with what input field names should be mapped to the SAS Fraud Management field names, then contact your SAS Fraud Management support personnel.

By default, SAS Business Orchestration Services prefers that incoming CSV data use a simple delimiter (for example, a comma) and that the delimiter is not part of the real data. This ensures that the most efficient data parsing is done for input data and results. If the incoming CSV data cannot use this simple format, then SAS Business Orchestration Services supports many other commonly used formats, such as the CSV format that you can export from the following software:

- InformixUnload
- MongoDBCsv
- MySQL
- Oracle
- PostgreSQLCsv
- RFC4180

Mapping a Byte Stream

Another commonly used input format is a binary data stream that can be converted into a text string. Unlike CSV data, this text string contains no information to indicate where each input field starts or ends. A separate schema file (or equivalent) is needed to describe how the data needs to be interpreted based on column positions.

By default, SAS Business Orchestration Services uses a simple file format to provide a schema. If you want to use a different format, then contact your SAS Fraud Management support personnel. Here is a sample schema file, which can be found at $BOSS_CONFIG/mappings/fbs/sample.fbs.

# This file contains a fixed block schema for processing fixed block stream data.
# fbsId must be specified first, and it needs to be unique.
# Each additional line describes a field, and it has three parts:
# Field name

```xml
<application>
  <message>
    <fields>
      <field name="version"/>
    </fields>
  </message>
</application>
```
Here is a sample input text string:

04.04.02CMD1

Using the previous sample schema, the parsing result is:

version = "04.04.02"
tran type = "CMD"
needResponse = "1"

Once the name-value pairs are produced, the rest of the processing is similar to what has been discussed in previous sections.

Mapping for Other Data Formats

Other data formats can be supported by SAS Business Orchestration Services with small code changes. Typically, you need to create a new bean processor.

The new bean processor’s main function is to convert input data into name-value pairs. Once the name-value pairs are created, the rest of the processing is similar to what has been discussed in previous sections.

Data Transformation

Previous sections described how different types of input data can be mapped to SAS Fraud Management transactions. This section describes how each request field or fields can be mapped to a SAS Fraud Management field or fields with or without data transformation. Mappings from the sample mapping files are used in the examples in this section.

Direct Mapping

Direct mapping is the easiest case where the request field value can be directly used or mapped to a SAS Fraud Management field value. The format of the mapping is to put the name of the SAS Fraud Management field name first and assign it a value from a field in the incoming transaction. Here is direct mapping sample:

```xml
<field name="rua_4byte_string_002">
  <srcField name="FTSTOLIQ_TRN_SUF"/>
</field>
```
Mapping By Transforming One Request Field

In the following example, the request field value must be parsed before it is assigned to a SAS Fraud Management field. SAS Business Orchestration Services allows users to use a Groovy script to transform a value before it is assigned to a SAS Fraud Management field.

```xml
<field name="rua_ind_004" default="U">
  <srcField name="FTSTOLIQ_STATUS">
    <groovy>
      return value.substring(0, 1)
    </groovy>
  <srcField>
</field>
```

Lines 73 through 79 show that only the first character from FTSTOLIQ_STATUS should be assigned to rua_ind_004.

```xml
<field name="rua_8byte_string_001">
  <srcField name="FTSTOLIQ_VAL_DATE">
    <groovy>
      return '20' + value
    </groovy>
  <srcField>
</field>
```

Lines 84 through 90 show that the value for FTSTOLIQ_VAL_DATE should be prefixed with 20 before it is assigned to the rua_8byte_string_001 field.

In the above mapping example, the variable named `value` represents the run-time value for the field name given in the `srcField` tag.

The run-time Groovy method has the following signature:

```
def getValue(String value, Map<String, String> records, Map<String, Object> cache)
```

In most cases, simple string methods should be sufficient for converting the passed in value. However, SAS Business Orchestration Services does not restrict you from using more advanced Groovy functions. How you use the other parameters, records, and cache are discussed in later sections.

Mapping Involving Multiple Request Fields

In some cases, the value that is assigned to a SAS Fraud Management field might depend on multiple input request fields. The following shows a multiple input request:

```xml
<field name="rua_10byte_string_002">
  <srcField name="FTSTOLIQ_DR_FED_ABA">
    <groovy>
      if (records['FTSTOLIQ_DR_ACCT_TYPE'] == 'M')
        return value
      else
        return ''
    </groovy>
  <srcField>
</field>
```

In the above mapping example, the variable named `value` represents the run-time value for the field name given in the `srcField` tag.

The run-time Groovy method has the following signature:

```
def getValue(String value, Map<String, String> records, Map<String, Object> cache)
```

In most cases, simple string methods should be sufficient for converting the passed in value. However, SAS Business Orchestration Services does not restrict you from using more advanced Groovy functions. How you use the other parameters, records, and cache are discussed in later sections.
In this example, the rua_10byte_string_002 field depends on FTSTOLIQ_DR_FED_ABA and FTSTOLIQ_DR_ACCT_TYPE.

If the request field FTSTOLIQ_DR_ACCT_TYPE has a value of M, then the value for field FTSTOLIQ_DR_FED_ABA is assigned to the SAS Fraud Management field named rua_10byte_string_002. If it has another value, then it assigns an empty string value to the SAS Fraud Management field named rua_10byte_string_002.

The use of the passed in Map<String, String> records is from the Groovy method signature:

def getValue(String value, Map<String, String> records, Map<String, Object> cache)

In this example, two request field names are used. The records map contains complete name-value pairs from the input data, so all request fields are accessible. It is even possible to modify the map, although that is not recommended.

If you need to modify the map to store some intermediate processing results, then you should use the cache map, which is discussed in a later section.

Mapping One Request Field to Multiple SAS Fraud Management Fields

It is possible that one request field might need to be mapped to multiple SAS Fraud Management fields. SAS Business Orchestration Services permits a srcField with the same name to show up multiple times.

In the following example, the first 35 characters of FTSTOLIQ_DR_PARTY are assigned to the dua_40byte_string_004 field. The next 35 characters are assigned to the rua_30byte_string_001 field.
Mapping User Variable Segments

You can define user variable segments in SAS Fraud Management. For more information about how to define and deploy user variable segments, see SAS Fraud Management: SAS Rules Studio User’s Guide. SAS Business Orchestration Services supports user variable segments, and allows for the mapping of incoming message field or fields in user variable segments.

Once user variable segments have been defined and deployed in SAS Fraud Management, an addendum file named `<messageAPIVersionNumber>.-addendum.xml` is generated. You need to put this addendum file in the `$BOSS_CONFIG` directory so that SAS Business Orchestration Services becomes aware of the custom segments and fields.

The following example shows what an entry in the addendum file looks like. In the SAS Business Orchestration Services mapping file, you can use either the field name or the alias. Most users prefer the alias, since it is more user friendly.

```xml
<field name="i00_str_00000_008" offset="32" format="$CHAR8." buildId="50106">
    <alias>i00_priority_code</alias>
    <description>description messages</description>
</field>
```

Data Validation in Mapping

SAS Business Orchestration Services enables you to insert validation in various stages of request processing. The mapping file is a convenient place to insert some validation rules.

In the following example, field `FTSTOLIQ_TRN_DATE` is expected to have a length of 6. When this is not true, the request processing throws an exception, and it is handled by the exception logic to send the proper response back to the caller.

```groovy
if (value == null || value.length() != 6) {
    throw new Exception("Invalid input data for TRN_DATE! " + value)
}
return '20' + value
```

Be aware that this validation is for the run-time field value only. Do not confuse it with the validation of the mapping file itself. The XSD for the mapping file is available, but it is for a different purpose.
Using Mapping Files in a Route

Previous sections talked about mapping files for different request input formats. This section shows how those mapping files can be used.

Using RequestDispatcher

The RequestDispatcher bean is available in SAS Business Orchestration Services after you have configured it. It handles data formats that are discussed in previous sections, and it can even be expanded to handle other formats by registering other handlers using the setHandlers(Map<String, Consumer<Exchange> handlers) method.

In the following example, line 79 exposes a REST URI, and lines 74 through 78 shows example usages. The request data is sent via an HTTP post. The mapperId in the URI tells the requestDispatcher which mapping should be used.

```xml
<post uri="/transaction/{format}"
  <route id="generalRoute" startupOrder="55">
    <convertBodyTo type="java.lang.String"/>
    <bean ref="odeUtils" method="decodeWebString"/>
    <to uri="direct:requestDispatcher"/>
    <bean ref="outboundMapper"/>
  </route>
</post>
```

```xml
<route id="Dispatcher" startupOrder="33">
  <from uri="direct:requestDispatcher"/>
  <doTry>
    <-- create Txn object -->
    <bean ref="requestDispatcher" method="process"/>
    <bean id="StoreUnqKeyFor_format" ref="uniqueKeySaver"/>
    <to uri="disruptor:sendTransactionToODE?timeout=100&waitForTaskToComplete=Always"/>
    <doCatch id="Catch Txn Processing Error(format)" id="Catch Custom Timeout(format)" id="Catch SendToODE Error(format)"
      <exception>com.sas.finance.fraud.ol.exceptions.TransactionProcessingException</exception>
      <exception>com.sas.finance.fraud.ol.exceptions.EnrichTimeoutException</exception>
      <exception>org.apache.camel.ExchangeTimedOutException</exception>
      <to uri="id="Stand-in-Response(format)" url="direct:standInResponse"/>
    </doCatch>
    <doCatch id="SendErrorHandler(format)" uri="direct:handleSendError"/>
```
Using Custom Processors

If the processors that are available in SAS Business Orchestration Services do not fit your needs, then you can create custom processors to handle your requests. In this case, you have more coding work to do, but it gives you more freedom in implementing your data handling logic.

Each mapping is represented by an XmlMapper object at run time, and XmlMapper can be retrieved using the following API based on the mapperId that you defined in your mapping XML file.

```java
IBMXmlMappingsRepository.getInstance.getXmlMapperByMapperId(String mapperId)
```

---

Data Enrichment

Data Enrichment Overview

This section discusses data enrichment by retrieving additional data from sources other than input requests. “Data Transformation” on page 12 covers different scenarios for converting request data fields to SAS Fraud Management fields. If your data handling involves only request data, not retrieving data from other sources, see “Data Transformation” on page 12.

Data enrichment can be done using one or both approaches:

- using a mapping file
- using a bean processor or processors

The following sections detail both approaches.

Data Enrichment Using a Mapping File

Overview

The inbound mapping file is a convenient place to configure data enrichment. SAS Business Orchestration Services supports the following resources. It can also be extended to support more resource types.

- relational database management systems (RDBMS)
- Redis servers
Data Enrichment Using an RDBMS

You might store additional information that is retrieved to enrich a transaction request in a relational database. Enrichment data are typically stored in tables, and can be queried based on some key value or values from an input request.

TIP You need to place an additional JDBC driver JAR file and any transitive dependencies in the $BOSS_LIB directory.

The following configuration uses a database query result to assign a SAS Fraud Management field named rua_8byte_string_003.

Line 225 says that the database named derby is being used, and it is configured in $BOSS_CONFIG/spring/dataSource.xml.

Line 244 uses the passed in jdbcTemplate parameter to run the query. The jdbcTemplate parameter comes from Spring Framework.

The name attribute in line 240 is optional. When it is specified, it contains a run-time value for the request field identified by name.

The dataSourceRepo bean can take multiple entries for its constructor argument sources. This means that multiple databases can be configured and used at the same time for data enrichment purposes.
Here is the exact Groovy method signature for RDBMS data enrichment:

def getValue(JdbcTemplate jdbcTemplate, String value, Map<String, String> records, Map<String, Object> cache)

Data Enrichment Using a Redis Server

You might store additional information that is retrieved to enrich a transaction request in a Redis server. Enrichment data is stored in a name-value cache, and it can be queried based on some key value or values from an input request.

TIP  You need to place an additional Redis client JAR file (Jedis) and any transitive dependencies in the $BOSS_LIB directory.

Redis provides a NoSQL storage alternative to a classical RDBMS for horizontal scalability and speed. In terms of implementation, key-value stores represent one of the largest (and oldest) members in the NoSQL space.

The following configuration uses a Redis server query result to assign a SAS Fraud Management field named rua_8byte_string_002.

106  <-- redisField to show enrich from Redis -->
107  <-- a Map of request NV pairs and Spring RedisTemplate<?, ?> are accessible inside Groovy -->
108  <-- rscName is mandatory which identifies a Redis server to use. -->
109  <-- name and default attributes are optional. If name is given, then its value can be conveniently accessed inside the Groovy block by a variable named "value" -->
110  <-- There are two ways to store query results for use in subsequent field mappings, to minimize number of Redis accesses:
111        1. Using the passed in "records" Map<String,String>, which has NV pairs from transaction request.
112        a. to update existing fields
113        b. to insert new name-value
114        2. Using the passed in "cache" Map<String, Object>, which is for data sharing purpose.
115           For example,
116                  cache.put('key1', 'v1')
117                  cache.get('key3', new Double(100.001))  -->
118  <field name="rua_8byte_string_002">
119     <redisField rscName="server1" name="FTSTOLIQ_ORIG_DATE" default="20170203">
120        <groovy>
121           return redisTemplate.opsForValue().get('myKey');
122        </groovy>
123     </redisField>
124  </field>

Line 120 shows that an rscName of server1 is being used.

The redisTemplate at line 122 is an instance of RedisTemplate from the Spring Data Redis package, which is an API for accessing Redis. Beware that the configuration of the Spring bean stringRedisTemplate can point to a Redis server cluster. This enables scalability and high availability. In addition, Spring Data Redis supports both Jedis and Lettuce as underlying libraries, which provide the same API for users.

server1 is configured in $BOSS_CONFIG/spring/ol-redis.xml.

30  <-- Use Spring Data Redis -->
RedisTemplateRepository can be configured with multiple instances of RedisTemplates. Therefore, it is possible for a mapping file to enrich data using multiple Redis servers.

Here is the exact Groovy method signature for Redis server enrichment:

```groovy
def getValue(RedisTemplate<?, ?> redisTemplate, String value, Map<String, String> records, Map<String, Object> cache)
```

Data Enrichment Using Other Resources

The same patterns used in mapping for an RDBMS, Redis server, and free text mining can be used to implement enrichment using other resources. For more information, contact your SAS Fraud Management support personnel.

Data Enrichment Using Bean Processors

Data enrichment can occur at different stages of request processing, and it can be done using different resources. Enrichment using a mapping file occurs right before the SAS Fraud Management transaction object is created. Data enrichment using custom bean processors is more flexible than enrichment using a mapping file.

You can create custom bean processors from scratch or you can extend SAS Business Orchestration Services classes. Once created, you can insert a custom bean into the request processing flow, and it can implement logic for accessing any other services.

The com.sas.finance.fraud.ol.enrich package provides a base interface, an abstract class, and the following sample enrichment examples that use a relational database and a Redis server:

```java
public interface IEnrich<T,R>;
public abstract class AbstractEnricher<T,R> implements IEnricher<T,R>;
public abstract class DbEnricher<T,R> extends AbstractEnricher<T,R>;
```

Data Enrichment Considerations

Data Enrichment Efficiency Using Service-Level Agreements

SAS Business Orchestration Services uses error-handling mechanisms such as time-outs to ensure data enrichment efficiency.
For each request processing, there is typically a service-level agreement (SLA) time constraint such as 100ms. If a normal response cannot be given in that amount of time, then a pre-agreed response needs to be sent to the caller. For more information, see “Configuring Stand-In Response” on page 38. Each node in the processing routes is responsible for overall time.

Data Enrichment Efficiency Using a Mapping File

When data enrichment is done using a mapping file, SAS Business Orchestration Services has already optimized the related components by pre-loading, pooling, or both.

It is possible that `<dbField>` and `<redisField>` might be used multiple times in the same mapping file. This means that the related component and API are invoked multiple times. For example, if `<dbField>` is used 10 times in a mapping file, and each time it calls `jdbcTemplate.queryForObject()`, then the database is accessed 10 times per request for data enrichment alone.

Reduce the number of queries as much as possible. Use the cache map to store query results for later use.

The scope for the cache map can be per request or per mapping file. The performance improvement is significant when cache is used properly.

Data Enrichment Efficiency Using a Bean Processor

When custom processor beans are implemented and used for data enrichment, you must ensure their efficiency.

Data Enrichment Error Handling

You might encounter errors with data enrichment. For example, you might be unable to reach resources, data might be missing, an operation might time out, and so on. The logic needed to handle errors can be very different for each case or customer.

Errors can be divided into recoverable errors and irrecoverable errors. Irrecoverable errors remain as errors each time that you retry. Recoverable errors are temporary and might not occur on the next try.

From an application business logic point of view, data enrichment operations can be divided into two categories:

- Mandatory enrichment
  
  When mandatory enrichment fails to complete successfully, the request should be marked as failed. The normal request routing should stop.

- Optional enrichment
  
  When operational enrichment fails, the request processing should continue as usual.

With correct message routing configuration, both categories can be handled.
Configuring Outbound Data Mapping

The outbound data mapping generates response messages for client systems in expected formats. SAS Business Orchestration Services has a few built-in outbound data mappers, and you can also create custom bean processors if more customized behavior is preferred.

Outbound Mapper Using Mapping File

SAS Business Orchestration Services uses outbound mapping XML files to collect and convert response information from different sources. Those sources can be any of the following:

- input request data
- transaction responses from SAS Fraud Management
- constant values
- data enrichment resources

The mapping results can be returned as a map of name-value pairs, XML, or JSON. It can be returned to client systems directly, or transformed into other formats depending on your route design.

First, configure a Spring bean. Here is an example of a bean using a mapperId of ctMasterResp, which identifies a mapping.

```xml
<bean id="OBMCtMaster" class="com.sas.finance.fraud.ol.demos.OutboundMapperUsingXmlMapping">
  <constructor-arg name="mapperId" value="ctMasterResp"/>
  <constructor-arg name="format" value="MAP"/>
</bean>
```

Note: The mapperId value should match with the value in your outbound mapping XML file.

Once the bean is defined, it can be used in a route. Depending on the route design and whether you want the message body to be updated with outbound mapping results, there are two different methods that you can invoke:

1. `<bean ref="OBMCtMaster" method="mapFromExchange"/>`
   
   This method uses the mapping information and creates an outbound result map (if it does not exist), and then puts the result map in a Camel exchange property. This step can be invoked more than once during message routing, so that information can be collected at different stages of message routing.

2. `<bean ref="OBMCtMaster" method="mapFromExchangeAndUpdateBody"/>`
   
   In addition to performing the Method 1 tasks, this method converts the result map into the requested format, and puts it into Camel exchange body. This is usually the last step of outbound mapping using this bean.
Mapping from an Input Request Data

Some applications might require that certain input field value or values be sent back in a response with or without a transformation.

The following example shows how this can be achieved:

```groovy
37     <-- Use an incoming request source field -->
38     <field name="ResponseName">
39        <srcField name="Foo.Bar" default="y">
40           <groovy>
41              return (value == 'FOO')? 'OOF' : value
42           </groovy>
43        </srcField>
44     </field>
```

This mapping tells SAS Business Orchestration Services to look for the input field named Foo.Bar. If its value is FOO, then it returns OOF. Otherwise, it returns its value. The returned value is assigned to a field named ResponseName, and this is part of the response data.

Mapping from Constant Values

Mapping a constant string value to a response field can be configured two ways. You can use either a `<constant>` tag or a default. The following example shows both methods.

```xml
46     <-- Two ways to specify static value -->
47     <field name="Response.Fld01">
48        <constant>
49           <![CDATA[plain text here.]]>
50        </constant>
51     </field>
52
53     <field name="Response.Fld02" default="FooBar"/>
```

Mapping from Enrichment Resources

Although it is uncommon to use data enrichment for outbound mapping, the functionality is available. The outbound mapping shares the same mapping file syntax and parsing engine as that of inbound mapping.

For detailed syntax about data enrichment, see “Data Enrichment” on page 17. The main difference is the field name. For inbound mapping, the field name is a SAS Fraud Management field name. However, for outbound mapping, it is the response field name.
Outbound Mapping Using Template Files

Some client applications might expect that response messages are sent in a fixed format that replaces certain fields or tokens for each transaction. In this case, the template file-based mapping can be used.

The template file-based mapping depends on the outbound mapper using a mapping file with the format set to Map. You also need to create a template file and place it in the $BOSS_CONFIG/mappings/outbound/templates/ directory.

Here is an example of a template file:

```
$ cat custom_outbound.template
<Response>
  <ServiceStatus>${Service.Status}</ServiceStatus>
  <ServiceError>${Service.ExceptionMsg}</ServiceError>
  <CardNumber>${CardNumber}</CardNumber>
  <Smh_rtn_code>${Response.Code}</Smh_rtn_code>
  <Request-Date>${Request Date}</Request-Date>
  <Ming-Fields>
    <Person>${FreeText Name}</Person>
    <Organization>${FreeText Organization}</Organization>
    <Location>${FreeText Location}</Location>
  </Ming-Fields>
</Response>
```

The key names that are enclosed in $ {...} are replaced by values from the key-value map.

You also need to configure a Spring bean. Here is an example:

```
<bean id="templateMapper" class="com.sas.finance.fraud.ol.demos.OutboundTemplateMapper">
  <constructor-arg name="templateFilename" value="${demo_outbound_response_template}"/>
</bean>
```

In your route, you also need to add a bean processor after the mapper node that uses the mapping file. Here is an example:

```
<bean ref="OBMCtMaster" method="mapFromExchangeAndUpdateBody"/>
<bean ref="templateMapper"/>
```
Managing SAS Business Orchestration Services

Logging

Logging Overview

SAS Business Orchestration Services uses log4j2 for logging. Log4j2 is flexible and configurable via its configuration file. For SAS Business Orchestration Services, this file is located at $BOSS_CONFIG/log4j2.xml.

The default logging level is set to ERROR, and log files are written to the $BOSS_LOG directory.

For most configurations, you need to change only the value for the root logger level.

```
<Root level="INFO">
   <AppenderRef ref="asyncAppLogAppender"/>
</Root>
```

From the most verbose to least verbose, valid values for the logging level are the following:

- ALL
- DEBUG
- INFO
- WARN
- ERROR
- FATAL
- OFF
SAS Business Orchestration Services recognizes modifications to the log4j2.xml file at run time. There is no need to restart SAS Business Orchestration Services.

Logging Throughput of a Route

You change only one line of your configuration to log the real-time throughput on a route.

```xml
<rest>
  <get uri="/ping">
    <route id="Ping">
      <to uri="log:myRoute01?level=INFO&groupInterval=5000"/>
      <transform><constant>pong</constant></transform>
    </route>
  </get>
</rest>
```

If the global logging level is INFO or lower, then in the log file that you see prints every 5000 milliseconds:

```
myRoute01:159 - Received: 500 new messages, with total 103000 so far. Last group took:
5000 millis which is: 100 messages per second. average: 100
```

Show Time Spent on Each Route Step

One or more routes might fulfill the processing of an incoming request, and each route might contain one or more steps. If you want to see how much time was spent on each step during the processing of a request, then you can turn on MsgHistLogger. Within the route of your choice, you can insert the following:

```xml
<onCompletion parallelProcessing="true">
  <to uri="direct:MsgHistLogger"/>
</onCompletion>
```

If the global logging level is INFO or lower, then in the log file you see messages such as the following:

```
generalRoute-onCompletion2-onCompletion : 0 ms
generalRoute-restBinding5-restBinding : 3 ms
generalRoute-general_route_logger-to : 1 ms
generalRoute-convertBodyTo2-convertBodyTo : 0 ms
generalRoute-bean14-bean : 1 ms
generalRoute-to8-to : 12 ms
Dispatcher-doTry1-doTry : 12 ms
  Dispatcher-bean3-bean : 5 ms
  Dispatcher-StoreUnqKeyFor_format-bean : 0 ms
Dispatcher-to1-to : 6 ms
    coreHandler-onCompletion1-onCompletion : 0 ms
    coreHandler-setTransactionId-bean : 0 ms
    coreHandler-bean8-bean : 1 ms
    coreHandler-validation-bean : 2 ms
    coreHandler-ODE Load Balancer-loadBalance : 0 ms
    coreHandler-odeSimulator01-bean : 0 ms
    coreHandler-bean9-bean : 0 ms
```
coreHandler-cacheResponseForStandIn-bean : 0 ms
generalRoute-bean15-bean : 1 ms
generalRoute-generalRoute-removeHeaders : 0 ms
generalRoute-to7-to : 1 ms
       MsgHistLogger-messageHistoryLogger-bean : 1 ms

The message is indented to separate child routes from parent routes.
Managing Data Flow

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store and Forward</td>
<td>29</td>
</tr>
<tr>
<td>Enabling Store and Forward</td>
<td>30</td>
</tr>
<tr>
<td>Configuring Store and Forward Data Store</td>
<td>32</td>
</tr>
<tr>
<td>Relational Databases</td>
<td>36</td>
</tr>
<tr>
<td>Using Unsupported Data Store</td>
<td>38</td>
</tr>
<tr>
<td>Configuring Stand-In Response</td>
<td>38</td>
</tr>
<tr>
<td>Overview</td>
<td>38</td>
</tr>
<tr>
<td>Enable Stand-In Response</td>
<td>39</td>
</tr>
<tr>
<td>Configuring Stand-In Data Store</td>
<td>39</td>
</tr>
<tr>
<td>Using Supported Data Stores</td>
<td>40</td>
</tr>
<tr>
<td>Configuring Communication between SAS Business Orchestration Services and SAS Fraud Management</td>
<td>43</td>
</tr>
<tr>
<td>Using Sockets for Communication</td>
<td>43</td>
</tr>
<tr>
<td>Using IBM MQ for Communication</td>
<td>44</td>
</tr>
<tr>
<td>Consuming SAS Fraud Management Messages on Downstream Queues</td>
<td>46</td>
</tr>
<tr>
<td>Consuming Messages from Offline Files</td>
<td>47</td>
</tr>
<tr>
<td>File Flow</td>
<td>47</td>
</tr>
<tr>
<td>Single Threaded Mode</td>
<td>47</td>
</tr>
<tr>
<td>Error Handling</td>
<td>48</td>
</tr>
<tr>
<td>Manual Correction</td>
<td>48</td>
</tr>
<tr>
<td>Advanced Option</td>
<td>48</td>
</tr>
<tr>
<td>Camel Routes</td>
<td>48</td>
</tr>
<tr>
<td>Directory Structure</td>
<td>49</td>
</tr>
<tr>
<td>Data Stores</td>
<td>50</td>
</tr>
<tr>
<td>Data Store Overview</td>
<td>50</td>
</tr>
<tr>
<td>Configuring a Data Store</td>
<td>50</td>
</tr>
<tr>
<td>Storing Data</td>
<td>52</td>
</tr>
<tr>
<td>Retrieving Data</td>
<td>52</td>
</tr>
</tbody>
</table>

Store and Forward

The store and forward feature of SAS Business Orchestration Services provides an additional mechanism to deal with service outages or connectivity issues. When SAS Business Orchestration Services is unable to communicate with a target system, it can be configured to store transactions to a persistent store, and forward those transactions when connectivity to the target server is reestablished.
Enabling Store and Forward

Overview

There are two parts to enabling store and forward:

- enabling store
- enabling forward

Enabling Store

Catching exceptions due to SAS Fraud Management connection problems can trigger the store and forward store operations.

```xml
<doTry>
<loadBalance id="ODE Load Balancer" inheritErrorHandler="false">
  <custom ref="odeLoadBalancer"/>
  <to id="ode_host_1" uri="netty4:tcp://{{ode_host_1}}:{{ode_port_1}}?sync=true&encoder=#encoder&decoder=#decoder"/>
  <to id="ode_host_2" uri="netty4:tcp://{{ode_host_2}}:{{ode_port_2}}?sync=true&encoder=#encoder&decoder=#decoder"/>
</loadBalance>
<doCatch>
  <exception>java.net.ConnectException</exception>
  <exception>java.lang.RuntimeException</exception>
  <to uri="direct:handleSendError"/>
  <!-- Throw a fake timeout exception to trigger a stand-in response. -->
  <script>
    <groovy>
      exchange.setException(new org.apache.camel.ExchangeTimedOutException(exchange, {{exception_to_timeout}}, {{exception_to_timeout_msg}}));
    </groovy>
  </script>
</doCatch>
</doTry>
```

The child route invokes the store and forward store method, which stores the failed transactions in the store and forward data store. There are additional store routes depending on the data store that you chose. The additional routes are described in "Configuring Store and Forward Data Store" on page 32.

```xml
<route id="handleSendError" startupOrder="20">
  <from uri="direct:handleSendError" />
  <choice>
    <when>
      <simple>${header.skipSaf} == 'true'</simple>
      <to uri="log:SAFSkipped?level=DEBUG"/>
      <setHeader headerName="safSkipped">
        <constant>true</constant>
      </setHeader>
    </when>
  </choice>
</route>
```
Near the end of the child route definition, a stand-in response is retrieved. For more information, see “Overview” on page 38.

Enabling Forward

There are different ways to trigger the forward operation. Out of the box, SAS Business Orchestration Services offers an implementation that uses a Camel timer to trigger it. The SAF bean has built-in logic to determine whether it is the right time to forward and how fast it can forward transactions. This logic is discussed in later sections.

The `saf_interval_milli` property determines how often store and forward is asked to try forward operations. It is not the rate at which failed transactions are forwarded.
Configuring Store and Forward Data Store

SAS Business Orchestration Services supports a few store and forward data store solutions out of the box. To support other data store solutions, see “Using Unsupported Data Store” on page 38.

In-Memory Data Store

The following example uses an in-memory queue to store failed transactions. It is used primarily for test and demonstration purposes. Do not use it in any production environment.

The following example shows an in-memory configuration.

```xml
<bean id="saf" class="com.sas.finance.fraud.ol.saf.BasicSafImpl">
  <constructor-arg name="waitSecondsAfterLastTimeout" value="30"/>
  <constructor-arg name="waitSecondsAfterLastError" value="60"/>
  <constructor-arg name="maxTxnsPerSecond" value="2000"/>
  <property name="forwardTemplate" ref="producerTemplateForSaf"/>
  <property name="storeTemplate" ref="safStoreTemplate"/>
  <property name="maxQueueSize" value="1000000"/>
  <property name="serviceStatus" ref="odeStatus"/>
  <property name="prefForwardProcessor" ref="uniqueKeySaver"/>
</bean>
```

The Camel template is an object that is used for forwarding transactions to a route when it is time to forward.

The `waitSecondsAfterLastTimeout`, `waitSecondsAfterLastError`, and `odeStatus` are used for determining when to start forwarding transactions. `odeStatus` can tell whether there are any reachable SAS Fraud Management systems. `waitSecondsAfterLastTimeout` specifies the number of seconds to wait after the last time-out before the next forward attempt can be made.

`maxTxnsPerSecond` controls the transaction forwarding rate.

Here is its additional store route:

```xml
<route id="safStoreRoute" startupOrder="22">
  <from uri="direct:safStore"/>
  <bean ref="odeStatus" method="setLastSendErrorTime"/>
</route>
```

Active MQ

The configuration for using Active MQ as a data store is similar to the in-memory data store, except for a few arguments for specifying Active MQ parameters.

The following ActiveMQ client JAR files and their transitive dependencies must be installed in the `$BOSS_LIB` directory.

- activemq-client-5.15.9.jar
- activemq-jms-pool-5.15.10.jar
- activemq-pool-5.15.9.jar
If you have an Active MQ server running already, use `saf_activemq_host` and `saf_activemq_port` to point to your server. `destination` configures the destination queue name for storing transactions.

```xml
<bean id="messageConverter" class="com.sas.finance.fraud.ol.saf.MessageConverterFactory"
    factory-method="createNewMessageConverter">
    <constructor-arg name="clazz" value="com.sas.finance.fraud.transaction.Transaction"/>
</bean>

<bean id="pooledConnectionFactory" class="org.apache.activemq.pool.PooledConnectionFactory"
    init-method="start" destroy-method="stop">
    <property name="maxConnections" value="10" />
    <property name="maximumActiveSessionPerConnection" value="10" />
    <property name="connectionFactory">
        <bean class="org.apache.activemq.ActiveMQConnectionFactory">
            <property name="brokerURL" value="tcp://${saf_activemq_host}:${saf_activemq_port}?wireFormat.maxInactivityDuration=0" />
        </bean>
    </property>
</bean>

<bean id="jmsConfig4ActiceMq" class="org.apache.camel.component.jms.JmsConfiguration">
    <property name="connectionFactory" ref="pooledConnectionFactory" />
    <property name="transacted" value="false" />
    <property name="concurrentConsumers" value="2" />
    <property name="maxConcurrentConsumers" value="10" />
    <property name="deliveryPersistent" value="true" />
    <property name="requestTimeout" value="10000" />
    <property name="cacheLevelName" value="CACHE_CONSUMER" />
</bean>

<bean id="activemq" class="org.apache.camel.component.jms.JmsComponent">
    <property name="configuration" ref="jmsConfig4ActiceMq" />
</bean>

<bean id="saf" class="com.sas.finance.fraud.ol.saf.JmsGenericSaf">
    <constructor-arg name="waitSecondsAfterLastTimeout" value="5"/>
    <constructor-arg name="waitSecondsAfterLastError" value="10"/>
    <constructor-arg name="maxTxsPerSecond" value="1000"/>
    <constructor-arg name="connectionFactory" ref="pooledConnectionFactory"/>
    <constructor-arg name="destination" value="${saf_queue_name}"/>
    <constructor-arg name="receiveWaitMs" value="100"/>
    <constructor-arg name="messageConverter" ref="messageConverter"/>
    <property name="forwardTemplate" ref="producerTemplateForSaf"/>
    <property name="storeTemplate" ref="safStoreTemplate"/>
    <property name="serviceStatus" ref="ODEStatus"/>
    <property name="preForwardProcessor" ref="uniqueKeySaver"/>
</bean>

Here is its additional store route:

```xml
<route id="safStoreRoute" startupOrder="22">
    <from uri="direct:safStore" />
    <doTry>
        <template id="safStoreTemplate"/>
        <property name="storeTemplate" ref="safStoreTemplate"/>
        <property name="serviceStatus" ref="ODEStatus"/>
        <property name="preForwardProcessor" ref="uniqueKeySaver"/>
    </doTry>
</route>
```
IBM MQ

Depending on your configuration scenarios, there are a few overloaded constructors that you can use. SAS Business Orchestration Services expects that you have properly installed and configured an IBM MQ manager or managers.

The following IBM MQ client JAR files and their transitive dependencies must be installed in the $BOSS_LIB directory.

- bcpkix-jdk15on-1.61.jar
- com.ibm.mq.allclient-9.1.3.0.jar
- javax.jms-api-2.0.1.jar

The following example shows an IBM MQ configuration as a store and forward data store.

```xml
<bean id="safWebsphereConnectionFactory" class="com.ibm.mq.jms.MQConnectionFactory">
  <property name="connectionNameList" value="${saf.ibm.mq.connectionNameList}"/>
  <property name="transportType" value="${saf.ibm.mq.transport.type}"/>
  <property name="queueManager" value="${saf.ibm.qm.name}" />
  <property name="channel" value="${saf.ibm.mq.channel.name}" />
</bean>

<bean id="safWebsphereConfig" class="org.apache.camel.component.jms.JmsConfiguration">
  <property name="connectionFactory" ref="safCachedConnectionFactory" />
  <property name="concurrentConsumers" value="1" />
  <property name="transacted" value="false" />
  <property name="cacheLevelName" value="CACHE_CONSUMER" />
</bean>

<bean id="ibmmq" class="org.apache.camel.component.jms.JmsComponent">
  <property name="configuration" ref="safWebsphereConfig" />
</bean>

<bean id="safCredConnectionFactory" class="org.springframework.jms.connection.UserCredentialsConnectionFactoryAdapter">
  <property name="targetConnectionFactory" ref="safWebsphereConnectionFactory" />
  <property name="username" value="${saf.ibm.mq.username}" />
  <property name="password" value="${saf.ibm.mq.password}" />
</bean>

<bean id="safCachedConnectionFactory" class="org.springframework.jms.connection.CachingConnectionFactory">
  <property name="targetConnectionFactory" ref="safCredConnectionFactory" />
  <property name="sessionCacheSize" value="50" />
</bean>
```

---

Chapter 4 / Managing Data Flow
<!-- Use IbmMq message queue based Store and Forward. -->

<bean id="saf" class="com.sas.finance.fraud.ol.saf.JmsGenericSaf">
  <constructor-arg name="waitSecondsAfterLastTimeout" value="5"/>
  <constructor-arg name="waitSecondsAfterLastError" value="10"/>
  <constructor-arg name="maxTxnsPerSecond" value="1000"/>
  <!-- You can use safCachedConnectionFactory for a performance boost-->
  <constructor-arg name="connectionFactory" ref="safCredConnectionFactory"/>
  <constructor-arg name="destination" value="${saf_queue_name}"/>
  <constructor-arg name="receiveWaitMs" value="10"/>
  <constructor-arg name="messageConverter" ref="messageConverter"/>
  <property name="forwardTemplate" ref="producerTemplateForSaf"/>
  <property name="storeTemplate" ref="safStoreTemplate"/>
  <property name="safetyStatus" ref="ODEStatus"/>
  <property name="preForwardProcessor" ref="uniqueKeySaver"/>
</bean>

Here is its additional store route:

<route id="safStoreRoute" startupOrder="22">
  <from uri="direct:safStore" />
  <doTry>
    <to uri="ibmmq:queue:${saf_queue_name}?messageConverter=#messageConverter&disableReplyTo=true"/>
    <doCatch>
      <!-- Exception thrown from here can cause errors recursively. We'd better catch it. -->
      <exception>java.lang.Throwable</exception>
      <to uri="log:?level=ERROR"/>
    </doCatch>
  </doTry>
  <bean ref="odeStatus" method="setLastSendErrorTime"/>
</route>

The bean configuration for cachedConnectionFactory can be found at ibmmq.xml.

Kafka

You can use a Kafka server or cluster for SAF data store. The following example shows a Kafka configuration using the com.sas.finance.fraud.ol.saf.KafkaSafImpl class.

<bean id="saf" class="com.sas.finance.fraud.ol.saf.KafkaSafImpl" init-method="initialize">
  <constructor-arg name="waitSecondsAfterLastTimeout" value="${saf.kafka.wait.afterlasttimeout.sec:10}"/>
  <constructor-arg name="waitSecondsAfterLastError" value="${saf.kafka.wait.afterlasterror.sec:20}"/>
  <constructor-arg name="maxTxnsPerSecond" value="${saf.kafka.max.txnpersec:1000}"/>
  <constructor-arg name="uri" value="${saf.kafka.host}:${saf.kafka.port}"/>
  <constructor-arg name="topic" value="${saf.kafka.topic}"/>
  <constructor-arg name="groupId" value="${saf.kafka.groupid}"/>
  <property name="pollWait" value="${saf.kafka.poll.wait.millisec:100}"/>
  <property name="pollInterval" value="${saf.kafka.poll.interval.millisec:200}"/>
  <property name="pollTimeOut" value="${saf.kafka.poll.timeout.millisec:2000}"/>
  <property name="forwardTemplate" ref="producerTemplateForSaf"/>
  <property name="storeTemplate" ref="safStoreTemplate"/>
  <property name="safetyStatus" ref="odeStatus"/>
  <property name="preForwardProcessor" ref="uniqueKeySaver"/>
  <property name="kafkaValueDeserializer" value="com.sas.finance.fraud.ol.converter.ByteArrayDeserializerForKafka"/>
</bean>
When you deploy multiple instances of SAS Business Orchestration Services, make sure that there are multiple partitions on a topic so that each instance of SAS Business Orchestration Services can handle forwarding concurrently.

For more information about Kafka installation and configuration, see Kafka Documentation.

Here is its additional store route:

```xml
<route id="safStoreRoute" startupOrder="22">
  <from uri="direct:safStore" />
  <doTry>
    <to uri="kafka:{{saf.kafka.topic}}?brokers={{saf.kafka.host}}:{{saf.kafka.port}}&bridgeEndpoint=true&serializerClass=com.sas.finance.fraud.ol.converter.TransactionSerializerForKafka" />
    <doCatch>
      <!-- Exception thrown from here can cause errors recursively. We'd better catch it. -->
      <exception>java.lang.Throwable</exception>
      <to uri="log:?level=ERROR" />
    </doCatch>
  </doTry>
  <bean ref="odeStatus" method="setLastSendErrorTime" />
</route>
```

Relational Databases

Relational databases can be used for SAF storage. However, you are strongly discouraged from using a Relational Database Management System (RDBMS) because it is a non-native message queue system.

Oracle Database

The following Oracle database client JAR files and their transitive dependencies must be installed in the \$BOSS_LIB directory.

**TIP** The following Oracle client JAR files are for use with Oracle Database 19.

- c3p0-0.9.5.2.jar
- mchange-commons-java-0.2.11.jar
- ojdbc8-19.3.0.0.jar
- ons-19.3.0.0.jar
- oraclepki-19.3.0.0.jar
- osdt_cert-19.3.0.0.jar
- osdt_core-19.3.0.0.jar
- simplefan-19.3.0.0.jar
- ucp-19.3.0.0.jar

Here is an example of an Oracle database for SAF storage configuration.

```xml
<bean id="siMqOdeTxnSerializer" class="com.sas.finance.fraud.ol.mq.SiMqOdeTxnSerializer"/>
```
<bean id="dbMqDataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close">
  <property name="driverClass" value="oracle.jdbc.driver.OracleDriver" />  
  <property name="jdbcUrl" value="jdbc:oracle:thin:@${saf.rdbms.oracle.host}:${saf.rdbms.oracle.port}/${saf.rdbms.oracle.dbname}"/>  
  <property name="user" value="${saf.rdbms.oracle.user}"/>  
  <property name="password" value="${saf.rdbms.oracle.password}"/>  
  <property name="maxPoolSize" value="20" />  
  <property name="minPoolSize" value="5" />  
  <property name="maxStatements" value="100" />  
  <property name="testConnectionOnCheckout" value="true" />
</bean>

<bean id="safFactory" class="com.sas.finance.fraud.ol.saf.simq.OracleSafFactory">
  <constructor-arg name="dataSource" ref="dbMqDataSource"/>  
  <constructor-arg name="queueName" value="safDbQueue"/>  
  <constructor-arg name="serializer" ref="siMqOdeTxnSerializer"/>  
  <constructor-arg name="forwardTemplate" ref="producerTemplateForSaf"/>  
  <constructor-arg name="serviceStatus" ref="odeStatus"/>  
  <constructor-arg name="preForwardProcessor" ref="uniqueKeySaver"/>
  <!-- Override the following default values with caution and only if you know what you are doing. -->  
  <!-- property name="waitAfterLastTimeout" value="${saf.rdbms.oracle.wait.afterlasttimeout.sec:10}"/>-->  
  <!-- property name="waitAfterLastError" value="${saf.rdbms.oracle.wait.afterlasterror.sec:20}"/>-->  
  <!-- property name="maxTxnsPerSecond" value="${saf.rdbms.oracle.max.txnpersec:1000}"/>-->  
  <property name="consumerReceiveBatchCount" value="5000"/>
</bean>

<bean id="saf" factory-bean="safFactory" factory-method="createNewSaf"/>

Here is its additional store route:

<route id="safStoreRoute" startupOrder="22">
  <from uri="direct:safStore" />  
  <bean ref="odeStatus" method="setLastSendErrorTime" />  
</route>

### PostgreSQL Database

The following PostgreSQL client JAR files and their transitive dependencies must be installed in the $BOSS_LIB directory:

- c3p0-0.9.5.2.jar
- mchange-commons-java-0.2.11.jar
- postgresql-42.2.5.jar

Here is an example of a PostgreSQL database for SAF storage configuration.

<bean id="siMqOdeTxnSerializer" class="com.sas.finance.fraud.ol.mq.SiMqOdeTxnSerializer"/>
<brn id="dbMqDataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close">
  <property name="driverClass" value="org.postgresql.Driver" />  
  <property name="jdbcUrl" value="jdbc:postgresql://${saf.rdbms.postgres.host}:${saf.rdbms.postgres.port}/${saf.rdbms.postgres.dbname}"/>  
  <property name="user" value="${saf.rdbms.postgres.user}"/>  
  <property name="password" value="${saf.rdbms.postgres.password}"/>  
  <property name="maxPoolSize" value="20" />  
  <property name="minPoolSize" value="5" />  
  <property name="maxStatements" value="100" />  
  <property name="testConnectionOnCheckout" value="true" />
</bean>

<bean id="safFactory" class="com.sas.finance.fraud.ol.saf.simq.PostgresSafFactory"/>
Using Unsupported Data Store

To use data stores that are not supported by SAF, you need to either implement an ISaf interface, or extend the AbstractSafImpl class. If you plan on implementing an unsupported data store, you should discuss this with your SAS Fraud Management support personnel.

Configuring Stand-In Response

Overview

Stand-in response is the last received response from SAS Fraud Management for a given account number. For a valid transaction request from a client system, a response is expected even in scenarios where there is no SAS Fraud Management server available or there is a time-out during the request processing. It is a common practice to respond with a stand-in response in those scenarios.

Here are the two things that you need to configure for a stand-in response:

1. Enable stand-in response.
2. Configure stand-in data store.
Enable Stand-In Response

You enable stand-in response through configuration. The following example shows a configuration to handle a transaction time-out. Within the route, `<doCatch>` catches the time-out exceptions that it cares about, and invokes the child route `direct:standInResponse`.

```xml
<route id="customTransactionHandler" startupOrder="48">
    <from uri="disruptor:customTransaction?concurrentConsumers=8"/>
    <doTry id="Main route(custom)">
        <convertBodyTo type="java.lang.String"/>
        <bean id="decodeWebString" ref="odeUtils" method="decodeWebString"/>
        <bean id=" cust_to_Transaction" ref="inboundMapper" method="customToTransaction"/>
        <bean id="StoreUnqKeyForCustomTxn" ref="uniqueKeySaver"/>
        <!-Steps omitted -->
        <doCatch id="Catch Custom Timeout(custom)">
            <exception>com.sas.finance.fraud.ol.exceptions.EnrichTimeoutException</exception>
            <exception>org.apache.camel.ExchangeTimedOutException</exception>
            <to id="Stand-in-Response(custom)" uri="direct:standInResponse"/>
        </doCatch>
    </doTry>
</route>
```

The following example defines the child route `direct:standInResponse`. It logs the exception, invokes some beans to track the last time-out, and calls the `handleTimeout` logic.

```xml
<route id="standInResponse" startupOrder="21">
    <from uri="direct:standInResponse"/>
    <to id="log warn" uri="log:?level=WARN"/>
    <bean id="safSetTimeout" ref="saf" method="setLastTimeout"/>
    <bean id="safSetTimeoutTS" ref="reporter" method="setAolLastTimeoutTimestamp"/>
    <bean id="requestTimeoutHandler" ref="requestTimeoutHandler" method="handleTimeout"/>
</route>
```

The default implementation of `handleTimeout` gets the stand-in response from the stand-in data store based on the account number in the request. So, the processing of the request continues as if a transaction response was received from SAS Fraud Management. You can alter the flow and logic of the `handleTimeout` method by providing a custom implementation. However, it should not be necessary for the majority of cases.

Configuring Stand-In Data Store

SAS Business Orchestration Services updates the data store when a transaction response is received from SAS Fraud Management. It tries to retrieve a previously stored transaction per account number when it needs to. This is similar to a caching service for a data store.

Here are a few factors to consider about the data store:
Size: The data store size is proportional to the number of unique account numbers. For each account number, there can be one data entry. A data entry contains information such as the last SAS Fraud Management transaction response, account number, and a timestamp.

Persistence: Depending on your application requirements, the selected data store might need to persist data across SAS Business Orchestration Services restarts. SAS Business Orchestration Services offers a few choices out of the box, and they are covered in later sections.

Speed: The choice of data store can affect the SAS Business Orchestration Services’s performance while saving and retrieving to and from data store.

High availability: The data store needs to be reliable for SAS Business Orchestration Services’s stand-in response feature. When a data store becomes unavailable, SAS Business Orchestration Services operates as if the stand-in response feature is turned off.

Accessibility among SAS Business Orchestration Services: In real deployments, multiple instances of SAS Business Orchestration Services might be used. Depending on your business needs, it might be necessary for multiple instances of SAS Business Orchestration Services to share the same data store.

Using Supported Data Stores

SAS Business Orchestration Services puts no restriction on what data stores you can use. However, it does offer a few data stores that are supported out of box.

In-Memory Data Store

The in-memory data store is the simplest and the most efficient stand-in data store that is offered by SAS Business Orchestration Services. It is also easy to configure. The sample configuration for stand-in data store can be found at $BOSS_CONFIG/spring/ol-beans.xml.

```xml
  <bean id="requestTimeoutHandler"
       class="com.sas.finance.fraud.ol.standin.RequestTimeoutHandler">
    <property name="standInDatastore">
      <bean class="com.sas.finance.fraud.ol.standin.CachedResponses"/>
    </property>
  </bean>
```

However, the in-memory data store does not offer persistence, and it is not sharable among instances of SAS Business Orchestration Services. When there are too many unique account numbers, there could be a memory usage issue. Thus, it is not recommended for production use.

Redis Data Store

Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker. It supports various data structures and queries. Redis has built-in replication, Lua scripting, LRU eviction, transactions, and different levels of on-disk persistence, and provides high availability via Redis Sentinel, and automatic partitioning with Redis Cluster. For more information, see Redis.
The following Redis client JAR files and their transitive dependencies must be installed in the $BOSS_LIB directory.

- commons-pool2-2.6.2.jar
- jedis-2.9.0.jar

The following is sample configuration for this stand-in data store.

```xml
<bean id="requestTimeoutHandler"
     class="com.sas.finance.fraud.ol.standin.RequestTimeoutHandler">
    <property name="standInDatastore">
        <bean class="com.sas.finance.fraud.ol.standin.RedisDatastore">
            <constructor-arg name="template" ref="transactionRedisTemplate"/>
            <constructor-arg name="stringTemplate" ref="stringRedisTemplate"/>
            <property name="timestampProvider" ref="timestampProvider"/>
        </bean>
    </property>
</bean>

The following lines configure Redis stand-in data store using a Jedis connection factory:

```xml
<!-- JedisConnectionFactory can be injected with RedisClusterConfiguration to handle clusters. -->
<bean id="jedisConnectionFactory" class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory"
     p:host-name="${redis_host}" p:port="${redis_port}" p:use-pool="true"/>

<bean id="stringRedisTemplate" class="org.springframework.data.redis.core.StringRedisTemplate"
     p:connection-factory-ref="jedisConnectionFactory"/>

<bean id="transactionRedisTemplate" class="org.springframework.data.redis.core.RedisTemplate">
    <property name="connectionFactory" ref="jedisConnectionFactory"/>
    <property name="keySerializer">
        <bean class="org.springframework.data.redis.serializer.StringRedisSerializer"/>
    </property>
    <property name="valueSerializer">
        <bean class="com.sas.finance.fraud.ol.standin.TransactionRedisSerializer"/>
    </property>
</bean>

<!-- By default, rqo_tran_date and rqo_tran_time are being used. It is configurable though. -->
<bean id="timestampProvider" class="com.sas.finance.fraud.ol.standin.ConfigurableDateTimeProvider"/>

The `timestampProvider` bean ensures that the transaction response that is stored in data store is the latest response. By default, the `rqo_tran_date` and `rqo_tran_time` fields are used to calculate a timestamp. As the class name `ConfigurableDateTimeProvider` implies, field names are configurable. Line 58 is equivalent to the following bean configuration. If you want to use different field names, then you can modify the values accordingly.

```xml
<bean id="timestampProvider" class="com.sas.finance.fraud.ol.standin.ConfigurableDateTimeProvider">
    <constructor-arg name="dateFieldName" value="rqo_tran_date"/>
    <constructor-arg name="timeFieldName" value="rqo_tran_time"/>
    <constructor-arg name="dateTimeFormat" value="yyyyMMddHH:mm:ss.SS "/>
</bean>
```
Relational Data Store

SAS Business Orchestration Services also supports using a relational database as stand-in data store, although it might not perform as well as the previous two choices. Performance varies depending on your environment and system settings.

**Note:** You need to put an additional JDBC JAR file for the database that you chose in the `$BOSS_LIB` directory.

The configuration for using a relational database is similar to using a Redis server. Its exact configuration depends on which type of relational database you use, and additional options for connection pooling. Here is an example for a MySQL server.

```xml
<bean id="requestTimeoutHandler" class="com.sas.finance.fraud.ol.standin.RequestTimeoutHandler">
    <!-- Use a relational database as StandIn Data Store -->
    <property name="standInDatastore">
        <bean class="com.sas.finance.fraud.ol.standin.RelationalDbDatastore">
            <constructor-arg name="jdbcTemplate" ref="jdbcTemplate"/>
            <property name="timestampProvider" ref="timestampProvider"/>
        </bean>
    </property>
</bean>

Line 150 uses the `jdbcTemplate` bean, which is a Spring JdbcTemplate instance. Its configuration depends on which type of relational database you use, and additional options for connection pooling. Here is an example for a MySQL server.

```xml
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
    <property name="dataSource" ref="dataSource"/>
</bean>

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close">
    <property name="driverClass" value="com.mysql.jdbc.Driver"/>
    <property name="jdbcUrl" value="${MySql_Server_Port}"/>
    <property name="user">
        <bean class="com.sas.finance.fraud.ol.secure.OLSecureUsername">
            <constructor-arg name="identity" value="${MYSQL_CREDENTIAL}"/>
        </bean>
    </property>
    <property name="password">
        <bean class="com.sas.finance.fraud.ol.secure.OLSecurePassword">
            <constructor-arg name="identity" value="${MYSQL_CREDENTIAL}"/>
        </bean>
    </property>
    <property name="maxPoolSize" value="25"/>
    <property name="minPoolSize" value="10"/>
    <property name="maxStatements" value="100"/>
    <property name="testConnectionOnCheckout" value="true"/>
</bean>
```

The configuration for `user` and `password` might not at first glance be straightforward. For more information, see “Encrypting Credentials” on page 61.

You need to create a table in your relational database that uses the SQL equivalent to the following example, which is for a MySQL database.

```sql
CREATE TABLE StandIn (Id INTEGER AUTO_INCREMENT PRIMARY KEY, AcctNum varchar(50), Response BLOB, Timestamp BIGINT, CONSTRAINT uc_key UNIQUE (AcctNum));
```
Using Unsupported Data Stores

SAS Business Orchestration Services can be extended to support other types of data stores, although only three data stores are supported out of the box.

To support new data store, implement the following SAS Business Orchestration Services interface and configure the Spring bean:

```java
com.sas.finance.fraud.ol.standin.IStandInDatastore
```

Configuring Communication between SAS Business Orchestration Services and SAS Fraud Management

By default, communication between SAS Business Orchestration Services and SAS Fraud Management uses sockets. SAS Business Orchestration Services offers IBM MQ as an alternative.

Using Sockets for Communication

Camel offers several components for socket communication. SAS Business Orchestration Services uses the Camel:Netty4 component to talk to SAS Fraud Management. For example, the following line of a configuration uses a Netty4 socket to communicate with a server listening on port 5018.

```xml
<to id="rdcesx12073" uri="netty4:tcp://rdcesx12073.race.sas.com:5018?sync=true&encoder=#encoder&decoder=#decoder"/>
```

The following options given in the uri:

- `sync=true`: A response is expected for each request sent.
- `encoder=#encoder`: Use the specified encoder bean when sending a request.
- `decoder=#decoder`: Use the specified decoder bean when receiving a response.

There are many other options available for the component.

In a real deployment, more than one socket might be configured to provide load balancing and failover support. For more information, see Chapter 5, “SAS Business Orchestration Services Scalability and High Availability,” on page 53.
Using IBM MQ for Communication

Using IBM MQ for Communication Overview

Although using sockets between SAS Business Orchestration Services and SAS Fraud Management is recommended, using IBM MQ to exchange request and response messages between SAS Business Orchestration Services and SAS Fraud Management works for systems that do not require very high throughput and have very low latency.

The following lists the key configuration differences when you use IBM MQ to communicate with SAS OnDemand Decision Engine.

- Instead of using the Camel:Netty4 component, use Camel:JmsComponent send messages to request the queue.

```xml
<to id="sendMessageToMQ" uri="ibmmq:queue:{{ibm.mq.req.queue.name}}?messageConverter=#messageConverterMQ&disableReplyTo=true"/>
```

The `ibmmq` is an instance of `JmsComponent` configured in `ibmmq.xml`. The `ibm.mq.req.queue.name` is the name of the SAS Fraud Management request queue, which is configured for SAS Fraud Management to listen for transaction requests.

- Use an asynchronous component to get a response from the SAS Fraud Management response queue.

```xml
<to id="asyncMsgRetrieval" uri="mrp:messageReceiver"/>
```

This component uses response messages received from the `ibm.mq.rsp.queue.name` queue. SAS Fraud Management sends response messages to this queue.

```xml
<route id="ODE_RESP_Q_Handler" startupOrder="32">
    <from uri="ibmmq:queue:{{ibm.mq.rsp.queue.name}}?messageConverter=#messageConverterMQ&disableReplyTo=true"/>
    <bean id="storeResponseTxn" ref="txnMsgHandler" method="process"/>
    <bean id="cacheResponseForStandIn" ref="requestTimeoutHandler" method="handleSuccess"/>
</route>
```

Multiple MQConnector versus Single MQConnector

MQConnector is configured on SAS Fraud Management for receiving transaction messages from a request queue and sending processed results to a response queue. SAS Business Orchestration Services sends transaction requests to the request queue. SAS Business Orchestration Services receives transaction responses from the response queue.

It is possible to have multiple SAS Fraud Management systems that point to the same pair of request and response queues, or to configure each SAS Fraud Management system to point to different pair of request and response queues.
Multiple instances of SAS Business Orchestration Services can use one pair of request and response queues, or multiple pairs of request and response queues.

### Configuration for a Single MQConnector

If you use only one pair of queues for multiple SAS Business Orchestration Services, then you need to use a message selector for each instance of SAS Business Orchestration Services to receive the correct responses.

```
<route id="ODE_RESP_Q_Handler" startupOrder="32">
  <from uri="ibmmq:queue:{{ibm.mq.rsp.queue.name}}?messageConsumer=MQ&disableReplyTo=true&selector={{ibm.mq.msg.selector}}"/>
  <bean id="storeResponseTxn" ref="txnMsgHandler" method="process"/>
  <bean id="cacheResponseForStandIn" ref="requestTimeoutHandler" method="handleSuccess"/>
</route>
```

Line 199 specifies the selector. The selector uses one of the message properties that are inserted in the original request message. For each instance of SAS Business Orchestration Services, you need to make sure a unique value is used for `olMsgSelectorIdx` within the SAS Business Orchestration Services group.

```
<bean id="messageConverterMQ" class="com.sas.finance.fraud.ol.converter.OLMessageConverterForOdeMQ">
  <property name="olMsgSelectorIdx" value="${ol.msg.selector.idx}"/>
</bean>
```

Your use of selector could add processing overhead on the message consuming end, thus increasing the overall latency. However, it reduces the number of queues to manage. In an environment where very high throughput is expected, you should consider using multiple instances of MQConnector.

### Configuration for Multiple MQConnectors

In this case, each instance of SAS Business Orchestration Services uses a different pair of request and response queues, so no selector is needed. Then, on the SAS Fraud Management side, it means that multiple instances of MQConnector need to be configured on each SAS Fraud Management system, so that all SAS Fraud Management systems can process requests from all request queues.

Here is the SAS Business Orchestration Services configuration for multiple instances of MQConnector:

```
<route id="ODE_RESP_Q_Handler" startupOrder="32">
  <from uri="ibmmq:queue:{{ibm.mq.rsp.queue.name}}?messageConsumer=MQ&disableReplyTo=true"/>
  <bean id="storeResponseTxn" ref="txnMsgHandler" method="process"/>
  <bean id="cacheResponseForStandIn" ref="requestTimeoutHandler" method="handleSuccess"/>
</route>
```

You can expect better performance from multiple instances of MQConnector. If they cannot provide sufficient performance, then you should use sockets.
Consuming SAS Fraud Management Messages on Downstream Queues

Previous sections focus primarily on configuring SAS Business Orchestration Services to receive and process transaction requests. SAS Business Orchestration Services can be easily configured to consume SAS Fraud Management messages (such as enterprise case management system (ECMI) messages and alerts) on downstream queues, and then process and deliver these messages according to your business needs.

The configuration is typically done by adding a route or routes, so that SAS Business Orchestration Services is able to use multiple transports.

Here is an example route that consumes SAS Fraud Management ECMI messages, processes them, and then sends them to different web services based on message content.

```xml
<route id="SAS Fraud Management_ECMI_CONSUMER">
  <from uri="ibmmq:queue:{{ibm.mq.queue.name}}?mapJmsMessage=false"/>
  <bean id="FulFill Message Processor" ref="NABOutboundPmtBean" method="handleFullFillMessages"/>
  <setHeader headerName="Exchange.CONTENT_TYPE">
    <constant>text/plain</constant>
  </setHeader>
  <choice id="ActionCodeBasedRouting">
    <when>
      <simple>${header.FFCode} == 'auto_action_1',<simple>
        <bean id="XmlToJson" ref="jsonMapperUtil" method="xmlToJson"/>
        <to id="SendToWebService1" uri="{{web.service.destination.1}}?bridgeEndpoint=true">
          <setHeader headerName="Exchange.CONTENT_TYPE">
            <constant>text/plain</constant>
          </setHeader>
        </to>
      </when>
      <otherwise id="NoMatchOnActionCode">
        <to id="NoMatch_SendToQ" uri="activemq:ode.analyst.ffcode.other?disableReplyTo=true"></otherwise>
      </when>
  </choice>
</route>
```

Line 509 specifies the endpoint from which messages are received. In this case, the endpoint is IBM MQ.

Line 511 invokes the bean processor method named handleFullFillMessages for parsing the incoming messages and sets them up for further routing. From Line 517 to 533, it uses content-based route EIP patterns and multicast EIP patterns to route messages to different destinations.
Consuming Messages from Offline Files

File Flow

Files are placed in the landing directory. Files that are complete and ready must be accompanied by marker files with .done appended after the name.

Within each SAS Business Orchestration Services instance, there is one Camel route that is using the file component to pick up files one by one from the landing directory and write them to the inProgress ${nodeId} directory. The inProgress_${nodeId} directory is unique to each SAS Business Orchestration Services instance. The SAS Business Orchestration Services ID can be FQDN, an IP address, or a human-assigned unique ID.

To ensure concurrent control to files from different SAS Business Orchestration Services instances, the Camel file component ___PreMove___ option is used. Before processing, Camel tries to move the file into the inProgress_${nodeId}/temp directory. If the operation is successful, then SAS Business Orchestration Services continues to process the file. If the operation fails, then the file was either moved by another SAS Business Orchestration Services instance or there was some other error.

Once files have been moved to the inProgress_${nodeId} directory, concurrent control is no longer needed. Files are visible only to that particular SAS Business Orchestration Services instance.

Another route with the file component monitors the inProgress_${nodeId} directory and picks up files one by one. It also looks for a ${file.name}.processed file and tries to obtain the last processed line number from it. This operation is done so that processing can continue after a crash. In this way, you have a single route to process both normal files and resumed files.

After a file has been processed successfully, SAS Business Orchestration Services moves it to the backup directory. Camel provides this function out of the box.

Single Threaded Mode

SAS Business Orchestration Services uses a single thread to process a file line by line. It creates a file named file_name.processed in the inProgress_${nodeId} directory. After each line of the record has been successfully processed, SAS Business Orchestration Services writes the line number. There is one record for each line. This way, if SAS Business Orchestration Services crashes while processing a file, it knows after start-up the line number of the last successfully processed record. SAS Business Orchestration Services can resume from the failure point. After successfully processing the entire file, SAS Business Orchestration Services deletes both the file and the file_name.processed file.
Error Handling

File Errors

If an exception occurs during file processing (for example, if there is an IOException or the file contains the wrong encoding), then Camel ends the processing of the entire file and puts the file in the `error_files` directory along with a file that contains the processed line number so that you can review the file.

Line or Record Errors

If an error occurs for a record during file processing (that is, if SAS Business Orchestration Services cannot parse the CSV format for that line), then SAS Business Orchestration Services creates a file named `file_name.record_error` in the `error_record` directory and appends the record so that you can review the file.

Manual Correction

In rare cases, manual correction might be needed. For example, if `boss01` crashed and is not going to recover, then you can go to the `inprogress_boss01` directory and copy the `${file.name}` and `${file.name}.processed` files to another directory (for example, `inprogress_boss02`) and create a `${file.name}.done` file. That way, `boss02` can resume processing that file.

Another even more rare case for manual correction occurs when SAS Business Orchestration Services crashes while trying to pre-move a file into the `inprogress_${nodeId}/temp` directory and while writing to the `inProgress_${nodeId}` directory. You can delete the partially written file in the `inProgress_${nodeId}` directory and move the file in the `inprogress_${nodeId}/temp` to the `landing` directory. Then you can create the `.done` file so that SAS Business Orchestration Services can process the file again. If the corresponding `${file.name}.camelLock` file exists in the `landing` directory, you should remove it. Otherwise, it is an orphaned lock file and prevents SAS Business Orchestration Services from processing `${file.name}`.

Advanced Option

For advanced concurrency control, you can implement the Camel `GenericFileExclusiveReadLockStrategy` interface with a distributed locking service across all SAS Business Orchestration Services instances. This can be backed by a Redis or a GemFire server.

Camel Routes

The following code shows how to define Camel routes.
<route id="offlinefileMove" startupOrder="88">
    <from uri="file://src/test/resources/offlinefiles/landing?delay=1s&delete=true&
        preMoves=../inprogress_${bean:offlineFilesId.toString}/temp/${file:name}
        &readLock=markerFile&
        moveFailed=../error_files&doneFileName=${file:name}.
        done&readLockDeleteOrphanLockFiles=false"/>
    <to uri="file://src/test/resources/offlinefiles?fileName=inprogress_${
        bean:offlineFilesId.toString}/${file:name}&doneFileName=${file:name}.done"/>
</route>

<route id="offlinefilesStart" startupOrder="85" errorHandlerRef="noErrorHandler">
    <from uri="file://src/test/resources/offlinefiles/inprogress_{{boss.offlinefiles.id}}
        ?delay=1s&move=../backup&moveFailed=../error_files&doneFileName=$simple{file:name}.done"/>
    <bean ref="lineRecordFileProcessor"/>
</route>

---

Directory Structure

Here is the directory structure for offline files.

/workingDir # we should have read/write permission this directory and down  
/backup # backup successfully processed files  
/error_files # exception occurs during file processing (ie. IOException).  
"processed" file (if any) indicates how many lines processed before exception. The  
directory is for human inspection only.  
  example_error_file.csv  
  example_error_file.csv.processed  
/error_records # exception occurs during processing single line/record. They are  
appended to a file for human inspection.  
  example_file_with_error_records.csv.error  
/inprogress_boss01.sas.com # file in progress by BOSS 01. "processed" file indicate  
how many lines processed at the moment. In case of BOSS crash, it will start from the  
next line.  
/temp # file preMoved here before writing to  
/inprogress_boss01.sas.com. Files here should be transient fast.  
  example_file_in_progress.csv  
  example_file_in_progress.csv.camelLock  
  example_file_in_progress.csv.done  
  example_file_in_progress.csv.processed  
/inprogress_boss02.sas.com  
/landing # landing directory for files. "done" file is a marker indicate the  
file is complete and ready for processing.  
  example_file_to_be_processed_next.csv  
  example_file_to_be_processed_next.csv.done
Data Stores

Data Store Overview

SAS Business Orchestration Services features, such as enrichment, store and forward, and stand-in response can be configured to use different types of external data stores. Properly selected data stores help improve overall system performance, throughput, and reliability. In addition to these features, a data store can also be used for other purposes, such as storing intermediate processing results.

External data stores are not limited to relational databases. You can also use a more high-performing in-memory data store, such as Redis or Pivotal GemFire Powered by Apache Geode. This section focuses on configuring SAS Business Orchestration Services to use either Redis or GemFire. Redis and GemFire provide similar functionality. SAS Business Orchestration Services does not favor one data store over the other. You choose which data store to use based on things like your existing infrastructure.

Note: The specifics of how you install and configure a Redis and GemFire data store are not covered in this document. For more information, see the documentation for that product.

Configuring a Data Store

SAS Business Orchestration Services supports Pivotal GemFire powered by Apache Geode and Redis.

Pivotal GemFire Powered by Apache Geode

The following example has one GemFire region named region01 acting in PROXY mode. The region is hosted in the server cluster and connects through two locators in myPool.

```xml
<bean id="datastore" class="com.sas.finance.fraud.ol.datastore.gemfire.GeodeFactory"
<gefe:client-cache id="eop-datastore" pool-name="myPool"/>
<gefe:pool id="myPool">
  <gefe:locator host="${datastore.gemfire.locator1.host}" port="${datastore.gemfire.locator1.port}"
  <gefe:locator host="${datastore.gemfire.locator2.host}" port="${datastore.gemfire.locator2.port}"
</gefe:pool>
<gefe:client-region id="region01" cache-ref="eop-datastore" pool-name="myPool" name="region01" shortcut="PROXY"/>
```

The following example code uses the region, region01, that was created above to configure a data store for general use and a data store proxy for use by Apache Camel.

```xml
<bean id="datastore" class="com.sas.finance.fraud.ol.datastore.gemfire.GeodeFactory"
Redis

The following code example configures a stand-alone Redis with Jedis connection factory.

The following Redis client JAR files and their transitive dependencies must be installed in the $BOSS_LIB directory.

- commons-pool2-2.6.2.jar
- jedis-2.9.0.jar

```xml
<bean id="redisStandaloneConfiguration" class="org.springframework.data.redis.connection.RedisStandaloneConfiguration">
    <constructor-arg name="hostName" value="${redis.hostname}" />
    <constructor-arg name="port" value="${redis.port}" />
</bean>

<bean id="jedisConnectionFactory" class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory">
    <constructor-arg name="standaloneConfig" ref="redisStandaloneConfiguration" />
    <property name="usePool" value="true" />
</bean>
```

The following example code creates a Redis backed data store using the Jedis connection factory that was configured above. It also creates a data store proxy for use by Apache Camel.

```xml
<bean id="stringRedisSerializer" class="org.springframework.data.redis.serializer.StringRedisSerializer" />
<bean id="redisTemplate" class="org.springframework.data.redis.core.RedisTemplate">
    <property name="connectionFactory" ref="jedisConnectionFactory" />
    <property name="keySerializer" ref="stringRedisSerializer" />
    <property name="valueSerializer" ref="stringRedisSerializer" />
</bean>

<bean id="redisDataCache" class="com.sas.finance.fraud.ol.datastore.redis.RedisDataCache">
    <constructor-arg name="redisTemplate" ref="redisTemplate" />
</bean>

<bean id="datastoreProxy" class="com.sas.finance.fraud.ol.syf.datastore.DatastoreProxy">
    <constructor-arg name="ds" ref="redisDataCache" />
</bean>
```
Storing Data

The following route removes a transaction from an IBM message queue and converts it to a Java Map interface. The Map data is then stored in the data store with txnId as the key.

```xml
<route id="transaction_to_datastore">
    <from uri="Ibmmq:queue:{ibm.mq.queue.name}?disableReplyTo=true"/>
    <bean id="byte_to_txn" ref="messageConverter" method="byteArrayToTransaction"/>
    <bean id="create_txn_backed_map" beanType="com.sas.finance.fraud.ol.xmlmapping.utils.TransactionBackedMap" method="createTransactionMap"/>
    <bean ref="datastoreProxy" method="store(${body.get('txnId')}, ${body})"/>
</route>
```

Retrieving Data

The following route retrieves data with a key that equals the header.dataKey in a Camel exchange. The header.dataKey can be set before invoking the route. The retrieved data is then stored in the retrieved_data property in the exchange.

```xml
<route id="datastore_retrieve">
    <bean ref="datastoreProxy" method="retrieve(${exchange}, ${header.dataKey}, 'retrieved_data')"/>
</route>
```
SAS Business Orchestration Services Scalability and High Availability

Scalability and High Availability Overview
SAS Business Orchestration Services is scalable both vertically and horizontally to meet throughput and latency requirements. Systems are usually scaled up vertically by using more powerful machines. Horizontal scalability is typically achieved by deploying more physical or virtual nodes.

You can also deploy SAS Business Orchestration Services in different ways to eliminate a single point of failure, such that the resulting system is highly available.

Tuning Performance Vertically
SAS Business Orchestration Services is multi-threaded and can use multiple CPU cores.

SAS Business Orchestration Services can be deployed and configured to do different things, but processing incoming transaction requests is one of the most common uses. How each transaction is routed and processed affects overall performance.

You have seen route configuration examples using Camel components and many other library classes for JMS configuration and database connection pooling. During
a transaction routing, many steps can happen, depending on your business needs. These steps can include (but are not limited to) in-bound mapping, data enrichment by accessing other resources such as a database, communication with SAS Fraud Management, and outbound mapping.

Many areas are tunable, and you should observe the impact after each change. Tunings are highly environment-dependent. There is no single configuration that fits all systems.

Horizontal Scale and High Availability

Horizontal Scale and High Availability Overview

In some cases, you might scale SAS Business Orchestration Services horizontally. Instead of deploying SAS Business Orchestration Services on a more powerful machine to meet increasing requirements, you deploy SAS Business Orchestration Services to multiple, less powerful machines or nodes to meet the same throughput and latency requirements. This deployment provides high availability naturally.

In front of those nodes, there is usually a hardware or software load balancer that distributes incoming requests. Those load balancers, such as F5 and HAProxy, can be deployed in high-availability mode. It does not matter to SAS Business Orchestration Services which upstream systems are used. It can be configured just like stand-alone mode.

Resource Sharing among SAS Business Orchestration Services Nodes

The configuration for each instance of SAS Business Orchestration Services is identical to its stand-alone mode, although the following resources can be shared among SAS Business Orchestration Services:

- Resources being used for enrichment: For more information, see “Data Enrichment” on page 17.
- These shared resources can be RDBMS, Redis, and others.
- Stand-in data store
- SAF data store

Number of SAS Business Orchestration Services Nodes to Use

The number of SAS Business Orchestration Services nodes to use depends on your business needs. For high availability support, at least two nodes are needed.
For more information, contact your SAS Fraud Management support personnel.

Number of SAS Fraud Management Nodes to Use

The number of SAS Fraud Management nodes to use depends on your business needs. For high availability support, at least two nodes are needed. A typical configuration has each SAS Business Orchestration Services load balancing with failover across all SAS Fraud Management nodes. So in each SAS Business Orchestration Services instance, you can use a configuration such as the following:

```xml
<loadBalance id="ODE Load Balancer" inheritErrorHandler="false">
  <custom ref="odeLoadBalancer"/>
  <to id="ode_host_1" uri="netty4:tcp://{{ode_host_1}}:{{ode_port_1}}?sync=true&encoder=#encoder&decoder=#decoder"/>
  <to id="ode_host_2" uri="netty4:tcp://{{ode_host_2}}:{{ode_port_2}}?sync=true&encoder=#encoder&decoder=#decoder"/>
  <to id="ode_host_3" uri="netty4:tcp://{{ode_host_3}}:{{ode_port_3}}?sync=true&encoder=#encoder&decoder=#decoder"/>
</loadBalance>
```

In this configuration, SAS Business Orchestration Services routes transaction requests to each SAS Fraud Management system. If there is a SAS Fraud Management failure, then SAS Business Orchestration Services tries the other SAS Fraud Management systems two times at most. When one or more SAS Fraud Management systems fail, but at least one SAS Fraud Management system is available, the whole system continues to operate, although at a reduced capacity.

In addition to failover capability, SAS Business Orchestration Services customized load balancer also remembers which external services are offline so that it skips them if they are offline. SAS Business Orchestration Services also provides a service health checker that periodically checks the health of external services. It detects the recovery of services.

Here is a typical configuration for a custom load balancer for SAS OnDemand Scoring Engine:

```xml
<bean id="serviceUnitChainGenerator" class="com.sas.finance.fraud.ol.monitor.LbServiceUnitChainGenerator">
  <constructor-arg name="generators">
    <list>
      <bean class="com.sas.finance.fraud.ol.monitor.SendProcessorServiceUnitGenerator"/>
      <bean class="com.sas.finance.fraud.ol.monitor.MockOdeServiceUnitGenerator"/>
    </list>
  </constructor-arg>
</bean>
```

```xml
<bean id="odeLoadBalancerFactory" class="com.sas.finance.fraud.ol.monitor.OdesDynamicFailoverLoadbalancerFactory">
  <constructor-arg name="serviceUnitGenerator" ref="serviceUnitChainGenerator"/>
  <constructor-arg name="healthCheckServiceName" value="ODE"/>
  <constructor-arg name="camelContext" ref="SPM_OL"/>
  <constructor-arg name="serviceStatus" ref="ODEStatus"/>
  <constructor-arg name="periodeInSec" value="30"/>
  <constructor-arg name="id" value="odeLoadBalancerFromBean"/>
  <constructor-arg name="exceptions">
    <list>
      <value>java.net.ConnectException</value>
      <value>java.lang.RuntimeException</value>
    </list>
  </constructor-arg>
</bean>
```
<list>
  <constructor-arg>
    <property name="startListeners">
      <list>
        <ref bean="odeCamelClientsManager"/>
      </list>
    </property>
  </constructor-arg>
</bean>

<bean id="odeLoadBalancer" class="com.sas.finance.fraud.ol.monitor.OdesDynamicFailoverLoadbalancerFactory"
  factory-bean="odeLoadbalancerFactory" factory-method="build"/>
<bean id="odeCamelClientsManager" class="com.sas.finance.fraud.ol.monitor.OdeCamelClientsManager">
  <constructor-arg name="serviceUnitGenerator" ref="serviceUnitChainGenerator"/>
  <property name="odeClientsChangeListeners">
    <list>
      <ref bean="odeProcessor"/>
      <ref bean="myLoggingEventNotifier"/>
    </list>
  </property>
</bean>
SAS Business Orchestration Services Security

Security Overview

SAS Business Orchestration Services offers a wide range of configuration options for security to fit different deployment scenarios. SAS Business Orchestration Services can be configured to offer secured services, and it can be configured to consume secured services from other third-party applications. SAS Business Orchestration Services also offers different ways to protect customer credential information.

Due to the flexibility of SAS Business Orchestration Services in its service offering and the wide range of third-party applications that it supports, it is impossible to show security configurations for all of them. Some typical security configurations are covered in this chapter.

If your security concerns are not addressed in this chapter, you should consult with your SAS Fraud Management implementation team.

Configuring HTTPs

To secure SAS Business Orchestration Services when it offers HTTP, REST services, or both, complete these steps:
1 Configure SSL context parameters. You need to provide properties that are enclosed in `{{ }}` in the properties file.

```xml
<sslContextParameters id="sslContextParameters" xmlns="http://camel.apache.org/schema/spring">
  <secureSocketProtocols>
    <!-- Do NOT enable SSLv3 (POODLE vulnerability) -->
    <secureSocketProtocol>TLSv1</secureSocketProtocol>
    <secureSocketProtocol>TLSv1.1</secureSocketProtocol>
    <secureSocketProtocol>TLSv1.2</secureSocketProtocol>
  </secureSocketProtocols>
  <keyManagers keyPassword="{{passPhraseForKeyRest}}">
    <keyStore resource="{{keyStoreForRest}}" password="{{passPhraseForKeyStoreRest}}"/>
  </keyManagers>
  <trustManagers>
    <keyStore resource="{{trustStoreForRest}}" password="{{passPhraseForTrustStoreRest}}"/>
  </trustManagers>
</sslContextParameters>
```

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyStoreForRest</td>
<td>Full path to the keystore file.</td>
</tr>
<tr>
<td>passPhraseForKeyRest</td>
<td>Password for the key.</td>
</tr>
<tr>
<td>passPhraseForKeyStoreRest</td>
<td>Password to open the keystore file.</td>
</tr>
<tr>
<td>passPhraseForTrustStoreRest</td>
<td>Password to open the truststore file.</td>
</tr>
<tr>
<td>trustStoreForRest</td>
<td>Full path to truststore file.</td>
</tr>
</tbody>
</table>

If you are not sure about the attributes, then talk to your security administrator who created the keystore and truststore for you.

2 Use SSL context parameters. Open `$BOSS_CONFIG/spring/camel-context-simple.xml` and make sure that the `ol-security.xml` file has been imported.

```xml
<import resource="ol-security.xml"/>
```

If the `<restConfiguration>` tag is being used, and you want to turn on SSL for all REST services configured under that tag, then you can include the following code:

```xml
<endpointProperty key="sslContextParametersRef" value="sslContextParameters"/>
```

If your HTTP was configured using a regular `<route>` tag, then you can use the `sslContextParameters` tag such as the following:

```xml
<route id="jettyRouteSample">
  <from url="jetty:{{https://0.0.0.0:9098}}/sampleservice?sslContextParametersRef=#sslContextParameters"/>
  <transform><constant>sample service using jetty httpsample service using jetty https</constant></transform>
</route>
```
3 Verify your HTTPS after you have made these changes, restart the SAS Business Orchestration Services, and test it using either a browser or the following command from the command line:

Curl -k -v https://localhost:9098/sampleservice; echo;

---

Configuring Basic Authentication

HTTPS is typically used in conjunction with basic authentication. SAS Business Orchestration Services uses Spring Security to support user authentication and role-based authorization.

If you are familiar with Spring Security, then the configuration of authentication and authorization in SAS Business Orchestration Services should be straightforward.

To configure basic authentication, complete these steps:

1 **Configure** `userDetailsService` and `authorizationPolicy`. The default in-memory `userDetailsService` configuration looks like the following:

   ```xml
   <!-- Use in-memory user info -->
   <security:user-service id="userDetailsService">
     <!--All the password are: jimspassword-->
     <security:user name="name" password="enter a password" authorities="ROLE_USER"/>
     <security:user name="name1" password="enter a password" authorities="ROLE_USER"/>
   </security:user-service>
   
   By default, `BCryptPasswordEncoder` is used for encoding the password, but you can change it to some other encoder if you want to.
   
   ```xml
   <bean id="passwordEncoder" class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"/>
   ```
   
   The authorization policy can be configured for a role like the following:
   
   ```xml
   <authorizationPolicy id="user" access="ROLE_USER"
       xmlns="http://camel.apache.org/schema/spring-security"/>
   ```

2 **Use** `authorizationPolicy` in routes. If you are using the `<restConfiguration>` tag in your configuration and you want to enable basic authentication, then you can include the following lines:

   ```xml
   <endpointProperty   key="filtersRef" value="springSecurityFilterChain"/>
   ```

   Inside the `<route>` tag, insert the following lines:

   ```xml
   <policy ref="user">
     <!--Your pre-existing routing steps -->
   </policy>
   ```

   With the `<policy>` tag, only an authenticated user with the ROLE_USER authority can access routes within the `<policy>` tag.
Depending on your deployment environment, it might be necessary for you to secure communication between a SAS Business Orchestration Services server or servers and an IBM MQ server or servers.

You should have the connectivity between MQ and SAS Business Orchestration Services working before turning on SSL. Once you have the connectivity working, you or your MQ administrator can configure SSL on the MQ server.

To enable SAS Business Orchestration Services to communicate with MQ securely, complete these steps:

1. Import the certificate from the MQ server into your truststore file.

2. Configure `sslContextParameters`. This step is the same as what is described in “Configuring HTTPS” on page 57. If the same truststore is being used for both HTTPS setup and MQ connectivity, then the same `sslContextParameters` can be used.

3. Use `sslContextParameters` on the MQ connection. When configuring IBM MQ, set the `SSLCipherSuite` and `SSLSocketFactory` properties.

   ```
   <!-- Turn it on for connection to MQ with SSL on. 
   <bean id="sslContext" factory-bean="sslContextParameters" factory-method="createSSLContext"/>
   <bean id="socketFactory" factory-bean="sslContext" factory-method="getSocketFactory"/>
   <bean id="sslStoreSetter" class="com.sas.finance.fraud.ol.utils.SystemPropertiesSetter">
     <constructor-arg name="nameValuePair">
       <map>
         <entry key="javax.net.debug" value="true"/>
         <entry key="com.ibm.mq.cfg.useIBMCipherMappings" value="false"/>
       </map>
     </constructor-arg>
   </bean>
   -->
   
   <property name="SSLCipherSuite" value="${ibm.mq.cipherSuite}"/>
   <property name="SSLSocketFactory" ref="socketFactory"/>
   -->
   
   Make sure that `ibm.mq.cipherSuite` has the correct value in the `camel-context.properties` file. If you are not sure what value to use, consult your MQ administrator.

Configuring SSL with SAS OnDemand Decision Engine

By default, a SAS Business Orchestration Services server communicates with SAS OnDemand Decision Engine using a socket. Between them, a SAS proprietary binary data stream is transmitted.

When secure socket is enabled for SAS Fraud Management, you need to configure secure socket in SAS Business Orchestration Services so that it communicates with SAS Fraud Management securely. For more information about secure socket configuration in SAS Fraud Management, see SAS Fraud Management: Installation and Configuration Guide.

To configure secure socket in SAS Business Orchestration Services, complete these steps:

1. Obtain a certificate from the SAS Fraud Management system that SAS Business Orchestration Services is going to communicate with.

2. On the machine where your SAS Business Orchestration Services is installed, import the certificate into the truststore.

   You can use keytool to do it, for example:
   ```
   keytool -import <path to trust store file> -storepass changeit -alias MyCertificate -file <certificate file>
   ```

3. In the `camel-context.properties` file, point the `sslKeyStoreFile` property and `sslTrustStoreFile` property to the truststore file.

4. In the `camel-context.properties` file, set `sslEnabled` to `true`.

5. Make sure that your route configuration uses the correct URI.

   `<to id="sfm1" uri="netty4:tcp://<host><port>?sync=true&ssl={{sslEnabled}}&passphrase={{sslPassPhrase}}&encoder=#encoder&decoder=#decoder&trustStoreFile=#sslTsf&keyStoreFile=#sslKsf"/>

Encrypting Credentials

SAS Business Orchestration Services provides two mechanisms to hide sensitive configuration information, such as the user name and password. One method stores the sensitive information in a metadata server, and the second method uses Jasypt.

Configure Credentials Using the Metadata Server

Credentials are typically needed for SAS Business Orchestration Services to access other systems, such as a relational database or a queue manager. To avoid putting clear-text passwords in a configuration file or files, SAS Business Orchestration
Services uses the system of record database to store credentials. You can define a SAS Business Orchestration Services credential store bean.

```xml
13  <!-- Credential store can be either OMR or local text file. -->
14  <bean id="olCredentialStore" class="com.sas.finance.fraud.ol.secure.OLSecureCredentialStore">
15     <!-- <constructor-arg name="filePath" value="${credentialFilePath}"/> -->
16     <constructor-arg name="host" value="${OMR_HOST}"/>
17     <constructor-arg name="port" value="${OMR_PORT}"/>
18     <constructor-arg name="domain" value="${OMR_DOMAIN}"/>
19  </bean>
```

The `com.sas.finance.fraud.ol.secure.OLSecureCredentialStore` also supports the use of a local credential file for scenarios where the system of record database is not available or is not used.

Once the credential store has been configured, it can be used in other bean configurations.

"Relational Data Store" on page 42 has an example for configuring a user name and password for accessing a database. The following example shows part of that configuration. `user` and `password` are configured using the `OLSecureUsername` and `OLSecurePassword` beans. In this example, the previous configured credential store is used.

```xml
32  <property name="user">
33     <bean class="com.sas.finance.fraud.ol.secure.OLSecureUsername">
34        <constructor-arg name="identity" value="${MYSQL_CREDENTIAL}"/>
35     </bean>
36  </property>
37  <property name="password">
38     <bean class="com.sas.finance.fraud.ol.secure.OLSecurePassword">
39        <constructor-arg name="identity" value="${MYSQL_CREDENTIAL}"/>
40     </bean>
41  </property>
```

Configure Credentials Using Jasypt

To configure credentials using Jasypt, complete these steps:

1. Modify `$BOSS_CONFIG/spring/camel-context-simple.xml`.

```xml
<bean class="org.apache.camel.component.jasypt.JasyptPropertiesParser">
   <!-- Use either plain, JVM property, or env property. -->
   <!-- <property name="password" value="eopphrase"/> -->
   <!-- <property name="password" value="sys:EOPENCPHRASE"/> -->
   <property name="password" value="sysenv:EOPENCPHRASE"/>
</bean>
```

2. Encrypt plaintext passwords in the `camel-context.properties` file.

   a. cd `$BOSS_BIN`
   b. export EOPENCPHRASE=<mypassphrase>
   c. For each password that you want to encrypt, issue the following command:

   ```
   ./encrypt.sh <mypassword>
   ```

Source text: `mypassword`
Encrypted text: 2R0KeB17xP1PldjT6jJErqaoWntdJliVQ

The output from this command provides encrypted text that you can use to replace the plaintext `<mypassword>` with the encrypted text from the output. For example:

```
posPhraseForKeyStoreRest=ENC(2R0KeB17xP1PldjT6jJErqaoWntdJliVQ)
```

3 Restart SAS Business Orchestration Services.

4 (Optional) Clear your command history and unset the exported environment variable to make sure that no one can see your EOPENCPHRASE.

   To clear your command history, use the `history -c` command.

   To unset an exported environment variable, use the `unset EOPENCPHRASE` command.
Appendix 1

Migrating to SAS Business Orchestration Services 10.1

Migration Overview

The following appendix details the process to upgrade to SAS Business Orchestration Services 10.1.

**TIP** Most of the additional JAR files that are referenced in this appendix can be found in the [Maven Repository](https://mvnrepository.com) website. You can acquire any vendor-specific JAR files that are not available on this website by contacting the vendor support team.

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Migration Overview</td>
<td>65</td>
</tr>
<tr>
<td>Configuration Files</td>
<td>66</td>
</tr>
<tr>
<td>Remove Unwanted Configuration Information</td>
<td>66</td>
</tr>
<tr>
<td>Upgrade Log4j to Log4j2</td>
<td>66</td>
</tr>
<tr>
<td>Install ActiveMQ and Store and Forward Using ActiveMQ</td>
<td>67</td>
</tr>
<tr>
<td>ActiveMQ Store and Forward Configuration</td>
<td>67</td>
</tr>
<tr>
<td>IBM MQ and Store and Forward Using IBM MQ</td>
<td>68</td>
</tr>
<tr>
<td>IBM MQ Store and Forward Configuration</td>
<td>68</td>
</tr>
<tr>
<td>Dynamic Load Balancer</td>
<td>70</td>
</tr>
<tr>
<td>PostgreSQL Database</td>
<td>71</td>
</tr>
<tr>
<td>Oracle Database</td>
<td>72</td>
</tr>
<tr>
<td>Redis</td>
<td>72</td>
</tr>
<tr>
<td>Offline Files Processing</td>
<td>72</td>
</tr>
<tr>
<td>Kafka Message Queue</td>
<td>73</td>
</tr>
<tr>
<td>Custom Outbound Mapping</td>
<td>73</td>
</tr>
</tbody>
</table>
Configuration Files

The installation directory structure has been changed as part of the changes that were made for SAS Business Orchestration Services 10.1. The typical installation path for SAS Business Orchestration Services 10.1 is /opt/sas/viya.

Before you make the changes that are explained in the following sections, the configuration files from previous installations need to be manually copied into the new configuration directory located at /opt/sas/viya/config/etc/boss. When copying the configuration files to the new directory, leave the setinit.txt file because it is already in the /opt/sas/viya/config/etc/boss directory.

Remove Unwanted Configuration Information

Remove or comment out the following configuration information if it can be found in your ol-beans-simple.xml file.

- `<import resource="ol-sasesp.xml"/>
- `<import resource="ol-mining.xml"/>
- `<import resource="ol-tokenization.xml"/>
- `<import resource="ol-threatmetrix.xml"/>
- `<bean id="transformerRepo" class="com.sas.finance.fraud.ol.transform.CustomTransformerRepository">
      <constructor-arg name="customerMappingsDir" value="${custom_mappings_folder}"/>
      <constructor-arg name="trimBlockDefinitions" value="true"/>
    </bean>

Remove the data mining mapping file named mappings/inbound/BNY_SampleMapping_2.xml.

Upgrade Log4j to Log4j2

If you have not modified the log4j.xml file, then copy the log4j2.xml file to the $BOSS_CONFIG directory.

If you have modified the log4j.xml file, then see Configuring Log4j 2 in “Migrating from Log4j 1.x.”
The following SAS Business Orchestration Services logs have been relocated to the following directories:

- Application log: /opt/sas/viya/config/var/log/boss/application.log
- Reports log and command history log: /opt/sas/viya/config/var/log/boss
- Standard out log: /opt/sas/viya/config/var/log/boss/boss.log

Remove the following bean from the ol-beans-simple.xml file (by default), if applicable.

```xml
<bean id="log4jWatcher" class="com.sas.finance.fraud.ol.utils.Log4jChangeWatcher">
    <constructor-arg name="log4jFile" value="log4j.xml" />
</bean>
```

Install ActiveMQ and Store and Forward Using ActiveMQ

If ActiveMQ is used either for message ingestion or outgoing (including but not limited to store and forward), the following ActiveMQ client JAR files and their transitive dependencies must be installed in the $BOSS_LIB directory.

- activemq-client-5.15.9.jar
- activemq-jms-pool-5.15.10.jar
- activemq-pool-5.15.9.jar
- commons-pool2-2.6.2.jar
- geronimo-j2ee-management_1.1_spec-1.0.1.jar
- geronimo-jms_1.1_spec-1.1.1.jar
- geronimo-jta_1.0.1B_spec-1.0.1.jar
- hawtbuf-1.11.jar

ActiveMQ Store and Forward Configuration

Replace the following ActiveMQ SAF bean configuration with the bean configuration that follows:

```xml
<bean id="saf" class="com.sas.finance.fraud.ol.saf.ActiveMqSafImpl">
    <constructor-arg name="waitSecondsAfterLastTimeout" value="5"/>
    <constructor-arg name="waitSecondsAfterLastError" value="10"/>
    <constructor-arg name="maxTxnsPerSecond" value="1000"/>
    <constructor-arg name="uri" value="tcp://${saf_activemq_host}:${saf_activemq_port}?wireFormat.maxInactivityDuration=0"/>
    <constructor-arg name="destination" value="${saf_queue_name}"/>
    <constructor-arg name="receiveWaitMs" value="100"/>
    <constructor-arg name="messageConverter" ref="messageConverter"/>
    <property name="forwardTemplate" ref="producerTemplateForSaf"/>
```
<property name="storeTemplate" ref="safStoreTemplate"/>
<property name="serverStatus" ref="ODEStatus"/>
<property name="preForwardProcessor" ref="uniqueKeySaver"/>
</bean>

<bean id="safPooledConnectionFactory"
    class="org.apache.activemq.pool.PooledConnectionFactory" init-method="start"
    destroy-method="stop">
    <property name="maxConnections" value="10" />
    <property name="maximumActiveSessionPerConnection" value="10" />
    <property name="connectionFactory">
        <bean class="org.apache.activemq.ActiveMQConnectionFactory">
            <property name="brokerURL" value="tcp://${saf_activemq_host}:${saf_activemq_port}?wireFormat.maxInactivityDuration=0" />
        </bean>
    </property>
</bean>

<bean id="saf" class="com.sas.finance.fraud.ol.saf.JmsGenericSaf">
    <constructor-arg name="waitSecondsAfterLastTimeout" value="5"/>
    <constructor-arg name="waitSecondsAfterLastError" value="10"/>
    <constructor-arg name="maxTxnsPerSecond" value="1000"/>
    <constructor-arg name="connectionFactory" ref="safPooledConnectionFactory"/>
    <constructor-arg name="destination" value="${saf_queue_name}"/>  
    <constructor-arg name="receiveWaitMs" value="100"/>
    <constructor-arg name="messageConverter" ref="messageConverter"/>
    <property name="forwardTemplate" ref="producerTemplateForSaf"/>
    <property name="storeTemplate" ref="safStoreTemplate"/>
    <property name="serverStatus" ref="ODEStatus"/>
    <property name="preForwardProcessor" ref="uniqueKeySaver"/>
</bean>

Make sure that the bean IDs in ref are consistent.

---

**IBM MQ and Store and Forward Using IBM MQ**

If IBM MQ is used either for message ingestion or outgoing (including but not limited to store and forward), the following IBM MQ client JAR files and their transitive dependencies must be installed in the $BOSS_LIB directory.

- `bcpkix-jdk15on-1.61.jar`
- `com.ibm.mq.allclient-9.1.3.0.jar`
- `javax.jms-api-2.0.1.jar`

---

**IBM MQ Store and Forward Configuration**

Replace the following ActiveMQ SAF bean configuration with the bean configuration that follows:

```xml
<bean id="saf" class="com.sas.finance.fraud.ol.saf.IbmMqSafImpl">
```

---
<constructor-arg name="waitSecondsAfterLastTimeout" value="5"/>
<constructor-arg name="waitSecondsAfterLastError" value="10"/>
<constructor-arg name="maxTxnsPerSecond" value="1000"/>
<constructor-arg name="connectionFactory" ref="websphereConnectionFactory"/>
<constructor-arg name="username" value="${saf.ibm.mq.username}"/>
<constructor-arg name="password" value="{saf.ibm.mq.password}"/>
<constructor-arg name="queueName" value="${saf_queue_name}"/>
<constructor-arg name="receiveWaitMs" value="10"/>
<constructor-arg name="messageConverter" ref="messageConverter"/>
<property name="forwardTemplate" ref="producerTemplateForSaf"/>
<property name="storeTemplate" ref="safStoreTemplate"/>
<property name="serviceStatus" ref="ODEStatus"/>
<property name="preForwardProcessor" ref="uniqueKeySaver"/>
</bean>

<bean id="safWebsphereConnectionFactory" class="com.ibm.mq.jms.MQConnectionFactory">
    <property name="connectionNameList" value="${saf.ibm.mq.connectionNameList}"/>
    <property name="transportType" value="${saf.ibm.mq.transport.type}" />
    <property name="queueManager" value="${saf.ibm.qm.name}" />
    <!-- Turn it on for MQ SSL. -->
    <property name="SSLCipherSuite" value="${saf.ibm.mq.cipherSuite}" />
    <property name="SSLSocketFactory" ref="socketFactory" />
</bean>

<bean id="safWebsphereConfig" class="org.apache.camel.component.jms.JmsConfiguration">
    <property name="connectionFactory" ref="safCachedConnectionFactory" />
    <property name="concurrentConsumers" value="1" />
    <property name="transacted" value="false" />
    <property name="cacheLevelName" value="CACHE_CONSUMER" />
</bean>

<!-- MQQueueConnectionFactory does not take username & password, so use this adapter to work around it. -->
<bean id="safCredConnectionFactory" class="org.springframework.jms.connection.UserCredentialsConnectionFactoryAdapter" init-method="afterPropertiesSet">
    <property name="targetConnectionFactory" ref="safWebsphereConnectionFactory" />
    <property name="username" value="${saf.ibm.mq.username}" />
    <property name="password" value="${saf.ibm.mq.password}" />
</bean>

<!-- caching or pooling is needed for performance -->
<bean id="safCachedConnectionFactory" class="org.springframework.jms.connection.CacheConnectionFactory">
    <property name="targetConnectionFactory" ref="safCredConnectionFactory" />
    <property name="sessionCacheSize" value="50" />
    <!-- You may override the default ExceptionListener -->
    <property name="exceptionListener" ref="jmsExceptionListener" />
</bean>

<!-- Use IbmMq message queue based store & forward. -->
<bean id="saf" class="com.sas.finance.fraud.ol.saf.JmsGenericSaf">
    <constructor-arg name="waitSecondsAfterLastTimeout" value="5"/>
    <constructor-arg name="waitSecondsAfterLastError" value="10"/>
    <constructor-arg name="maxTxnsPerSecond" value="1000" />
    <!-- You can use safCachedConnectionFactory for a performance boost -->
    <constructor-arg name="connectionFactory" ref="safCredConnectionFactory" />
    <constructor-arg name="destination" value="${saf_queue_name}" />
    <constructor-arg name="receiveWaitMs" value="10" />
</bean>
Make sure that the bean IDs in ref are consistent.

Dynamic Load Balancer

Replace the following configuration with the bean configuration that follows. (By default, it is in the ol-beans-simple.xml file.)

```xml
<bean id="odePingTxnGenerator" class="com.sas.finance.fraud.ol.monitor.OdePingTxnGenerator"/>
<bean id="odeProcessorLivenessChecker" class="com.sas.finance.fraud.ol.monitor.ServiceLivenessChecker" init-method="init">
    <constructor-arg name="serviceName" value="ODE"/>
    <constructor-arg name="context" ref="SFM_OL"/>
    <constructor-arg name="serviceStatus" ref="ODEStatus"/>
    <constructor-arg name="periodInSec" value="30"/>
    <constructor-arg name="healthExchangeGenerator" ref="odePingTxnGenerator"/>
</bean>

<bean id="endpointUriExtractor" class="com.sas.finance.fraud.ol.monitor.OdeEndpointUriExtractor"/>
<bean id="odeLoadBalancer" class="com.sas.finance.fraud.ol.monitor.DynamicFailOverLoadBalancer">
    <constructor-arg name="exceptions">
        <list>
            <value>java.net.ConnectException</value>
            <value>java.lang.RuntimeException</value>
        </list>
    </constructor-arg>
    <constructor-arg name="livenessChecker" ref="odeProcessorLivenessChecker"/>
    <property name="endpointUriExtractor" ref="endpointUriExtractor"/>
    <property name="roundRobin" value="true"/>
    <property name="camelContext" ref="SFM_OL"/>
    <property name="id" value="odeLoadBalancerFromBean"/>
    <property name="maximumFailoverAttempts" value="1"/>
    <property name="startListeners">
        <list>
            <ref bean="odeCamelClientsManager"/>
        </list>
    </property>
</bean>

<bean id="odeCamelClientsManager" class="com.sas.finance.fraud.ol.monitor.OdeCamelClientsManager">
    <property name="odeClientsChangeListeners">
        <list>
            <ref bean="odeProcessor"/>
            <ref bean="odeStatsArchiver"/>
            <ref bean="myLoggingEventNotifier"/>
        </list>
    </property>
</bean>
```
If a PostgreSQL database is used in any form, then the following PostgreSQL client JAR files and their transitive dependencies must be installed in the $$BOSS_LIB$$ directory.

- c3p0-0.9.5.2.jar
Oracle Database

If an Oracle database is used in any form, then the following Oracle database client JAR files and their transitive dependencies must be installed in the $BOSS_LIB directory. The following Oracle client JAR files are for database 19.

- c3p0-0.9.5.2.jar
- mchange-commons-java-0.2.11.jar
- ojdbc8-19.3.0.0.jar
- ons-19.3.0.0.jar
- oraclepki-19.3.0.0.jar
- osdt_cert-19.3.0.0.jar
- osdt_core-19.3.0.0.jar
- simplefan-19.3.0.0.jar
- ucp-19.3.0.0.jar

Redis

If Redis is used in any form, then the following Redis client JAR files and their transitive dependencies must be installed in the $BOSS_LIB directory.

- commons-pool2-2.6.2.jar
- jedis-2.9.0.jar

Offline Files Processing

Locate the scheduler=quartz2 option in your offline context configuration. Replace that value with scheduler=spring.
Kafka Message Queue

If Kafka is used in Camel routes for store and forward, then replace the following configuration with the bean configuration that follows.

```xml
<to uri="kafka:{{saf.kafka.host}}:{{saf.kafka.port}}?
bridgeEndpoint=true&topic={{saf.kafka.topic}}
&serializerClass=com.sas.finance.fraud.ol.converter.TransactionSerializerForKafka"/>
```

```xml
<to uri="kafka:{{saf.kafka.topic}}?brokers={{saf.kafka.host}}:
{{saf.kafka.port}}&bridgeEndpoint=true&serializerClass=
com.sas.finance.fraud.ol.converter.TransactionSerializerForKafka"/>
```

Custom Outbound Mapping

If a bean or beans are created out of the class named `com.sas.finance.fraud.ol.demos.OutboundMapperUsingXmlMapping`, then replace the constructor using mapperId with constructor using the following type and subtype.

There is a mapping file with the `mapperId=Obm1MapperId`, the `type=Obm1Type`, and the `subtype=Obm1SubType`. It is used to create the following bean:

```xml
<bean id="Obm1" class="com.sas.finance.fraud.ol.demos.OutboundMapperUsingXmlMapping">
    <constructor-arg name="mapperId" value="Obm1MapperId"/>
    <constructor-arg name="format" value="XML"/>
</bean>
```

Replace the information in the mapping file with the following information:

```xml
<bean id="Obm1" class="com.sas.finance.fraud.ol.demos.OutboundMapperUsingXmlMapping">
    <constructor-arg name="type" value="Obm1Type"/>
    <constructor-arg name="subType" value="Obm1SubType"/>
    <constructor-arg name="format" value="XML"/>
</bean>
```