SAP Cloud Platform Integration

Pipeline steps

Mayur Belur Mohan, SAP
April 09, 2019

PUBLIC

THE BEST RUN SAP

Pipeline Steps

- Pools
- Message Transformers
- Message Routing
- Message Validators
- Message Persistence
- Security Elements
- Events

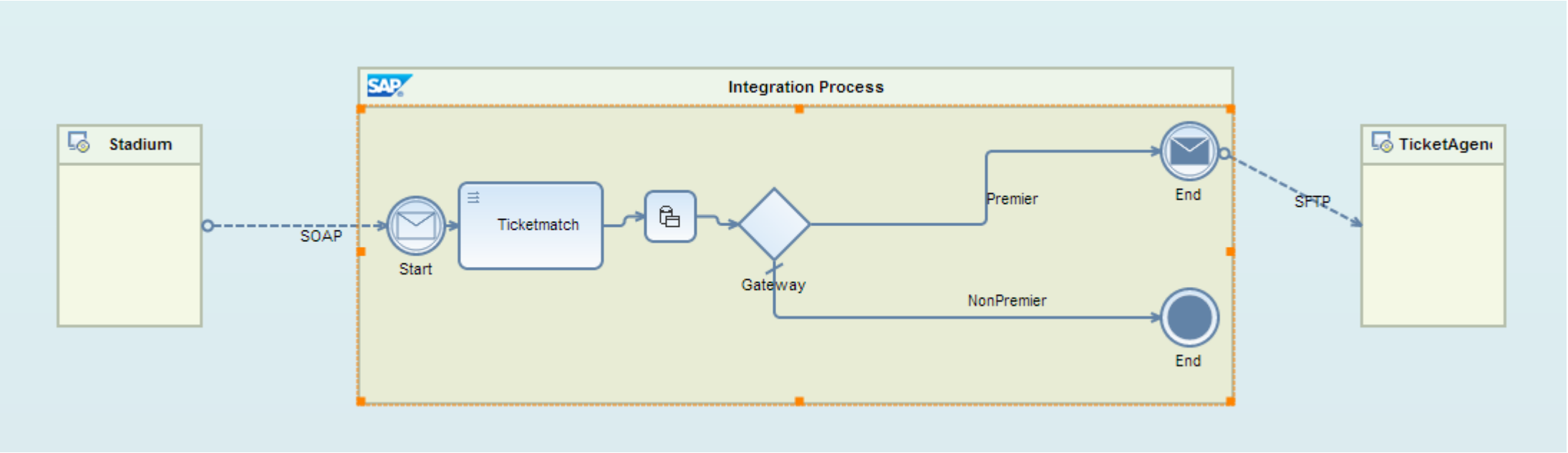
Pools

- Integration Process
- Local Integration Process
- Exception Subprocess

Pools: Integration Process

Integration Process

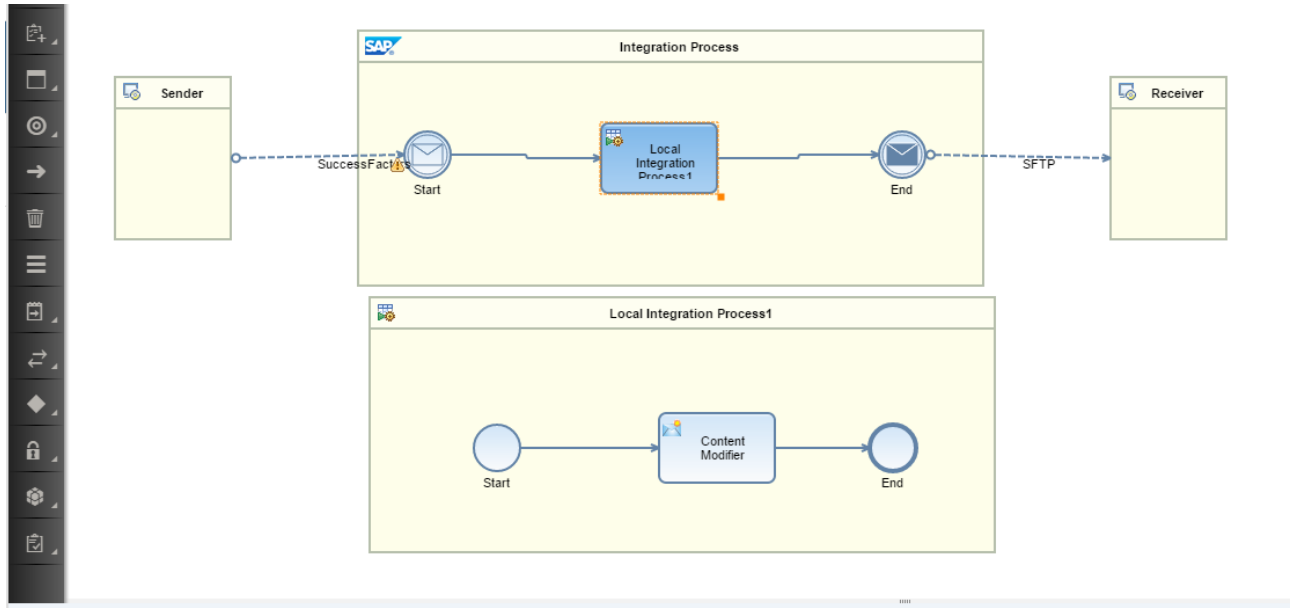
An integration process is an executable, cross-system process for processing messages. In an integration process, all the process steps that are to be executed and the parameters relevant for controlling the process are defined



Pools: Local Integration Process

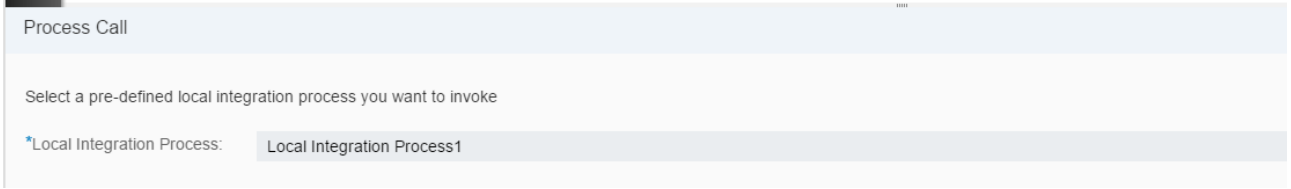
Local Integration Process

You can use the Local Integration Process to simplify your integration process. It allows you to break down the main Integration process into smaller fragments by using local integration processes.



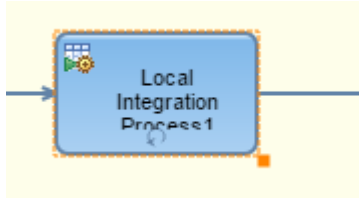
Process Call

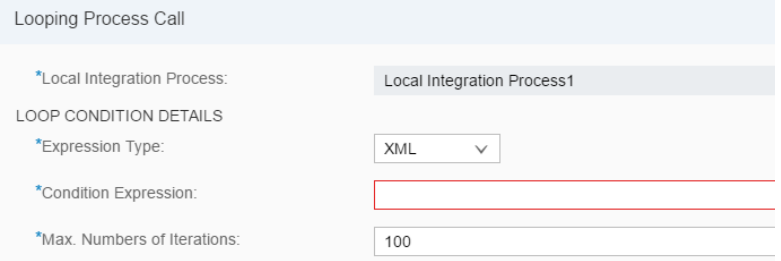
You can use the Process Call step to call the Local Integration Process from the main Integration process.



Looping Process Call

You can use this step to call a Local Integration Process n number of times, based on the XPath condition defined.





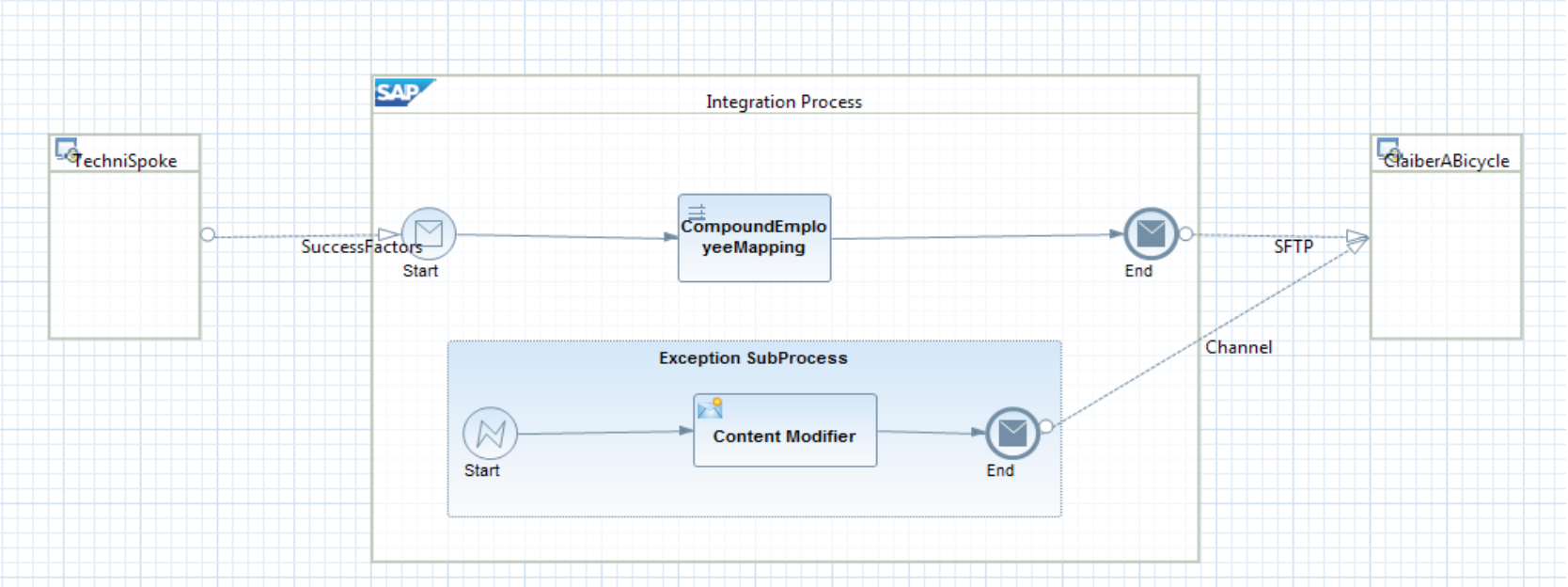
Pools: Exception Subprocess

Exception Subprocess

You can use this task if you want to catch any thrown exception in the integration process or local integration process and perform additional processing on it.

Note:

- You can use End Message event to wrap the exception in a fault message and send it back to the sender, in the payload.
- You can use Error End event to throw the exception to default exception handlers.
- Message Processing Log will be in error state even if a user catches an exception and performs additional processing on it.



Message Transformers

- Mapping
- Encoder
- Decoder
- Filter
- Content Modifier
- Converter
- Script

Message Transformers: Mapping

Message Mapping

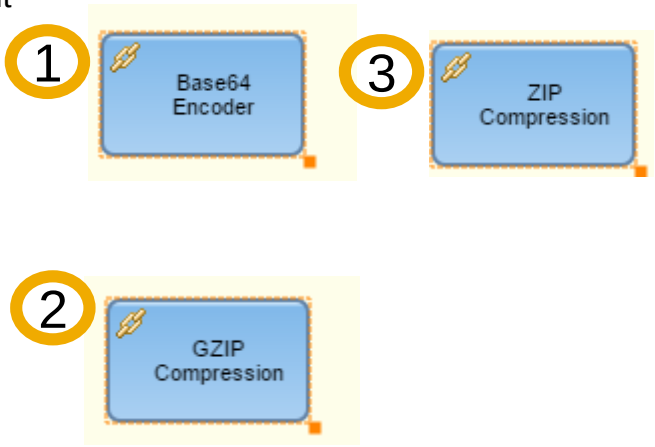
- This Process Step is used to create a message mapping when the receiver system accepts a different message format than that of the sender system. The message mapping defines the logic that maps input message structures with the required format of output message structures. Also called as graphical mapping. Editors are available in both eclipse as well as Cloud Platform Integration WebUI

XSLT Mapping

- XSLT Mapping is used to transform the format of the sender message to receiver message. XSLT mapping is created in xml editors like altova, xml spy and then imported in project.

Message Transformers: Encoder

This Process step is used to encode messages by using an encoding scheme to secure any sensitive message content during transfer over the network.



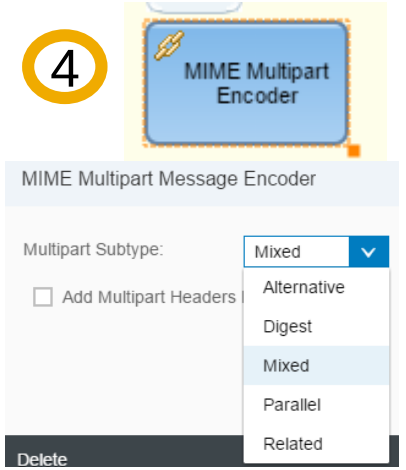
In the Properties view, following encoding schemes can be selected from the dropdown list:

1 **Base64 Encode:** Allows you to encode the message content using base64.

2 **GZIP Compress :** Allows you to compress the message content using GNU zip (GZIP).

3 **ZIP Compress:** Allows you to compress the message content using zip.

4 **MIME multipart encode**: Allows you to transform the message content into a MIME multipart message



Example

Consider the input XML payload structure to the encoder:

<message> Input for encoder</message>

If you select Base64 Encode, the output message would look like

PG1lc3NhZ2U+DQoJSW5wdXQgZm9yIGVuY29kZXINCjwvbWVzc2FnZT4NCg==

Message Transformers: Decoder

This Process step is used to decode the message received over the network to retrieve original data

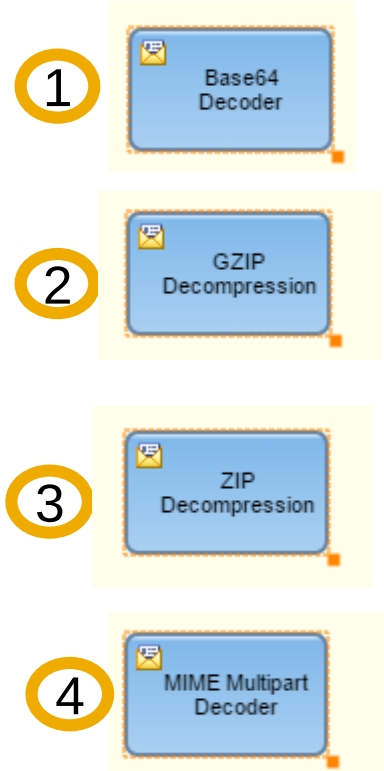
In the Properties view, following decoding schemes can be selected from the dropdown list.

1 **Base64 Decode:** Allows you to decode base64-encoded message content.

2 **GZIP Decompress :** Allows you to decompress the message content using GNU zip (GZIP).

3 **ZIP Decompress:** Allows you to decompress the message content using zip.

4 **MIME multipart decoder:** Allows you to transform a MIME multipart message into a message with attachments



Example

Let us suppose that an input message to the decoder is a message encoded in Base64 that looks like this:

PG1lc3NhZ2U+DQoJSW5wdXQgZm9yIGVuY29kZXINCjwvbWVzc2FnZT4NCg==

The output message of the decoder would be as follows:

<message> Input for encoder</message>

Message Transformers: Filter

This process step is used to filter information by extracting a specific node from the incoming message.

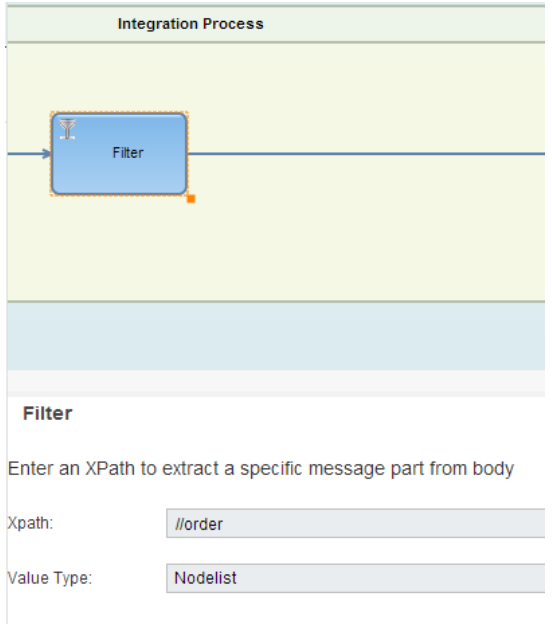
1 In the Properties view, enter an Xpath to extract a specific message part from the body. For example, in the Xpath field, enter **/ns0:MessageBulk/Message/MessageContent/text() .**

.

2 Returns the value of Node specified

.

3 Returns parent Node and child Nodes



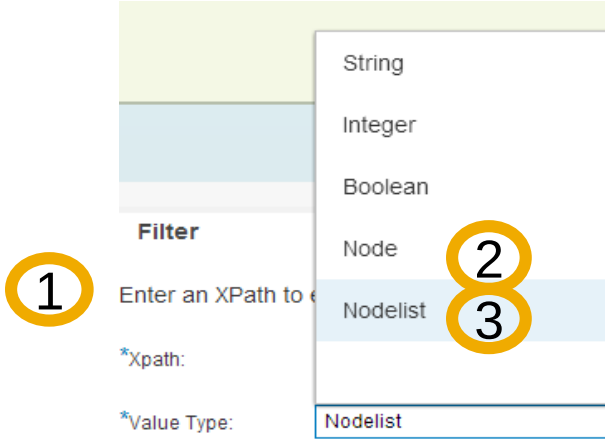
Example:

Consider an XML payload structure-

<Message><orders> <order> <clientId>I0001</clientId> <count>100</count> </order> <order> <clientId>I0002</clientId> <count>10</count> </order></orders></Message>

If you enter an Xpath- **/Message/orders/order/count/text()**.

The output of the Content Filter would be- **10010**



Message Transformers: Content Modifier

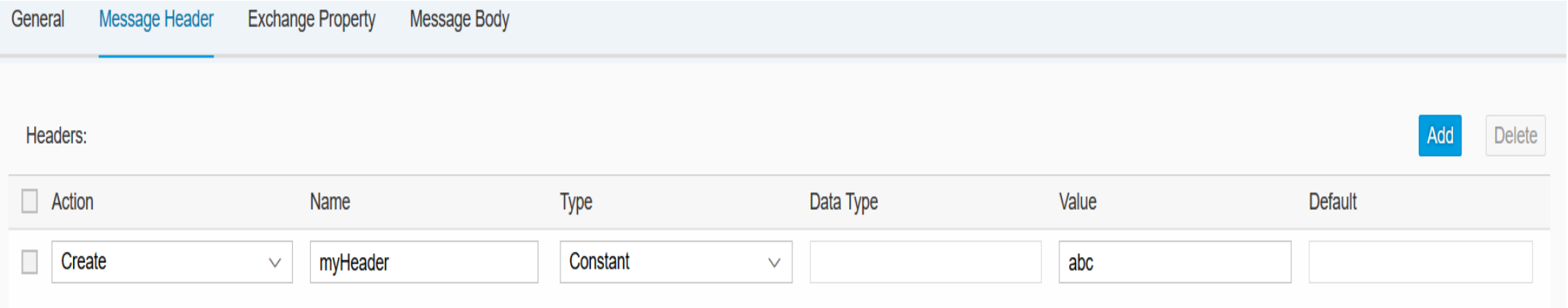
This Process step is used to modify the content of the incoming message by providing additional information in the header or body of a message before sending it to the receiver.

Message Header – Create headers, set values for existing headers or delete them from messages by using constants, another header, an XPath, a property , an external parameter, by forming an expression, a local variable or a global variable. To access Header, we can use ${**header.<Header Name>**}

Exchange Property Tab - Similar to headers, user can also define properties with different value types as explained above, by selecting Exchange Property tab in Properties view.

Note: Header values can be lost post external system call, whereas properties will be available for complete message execution. During outbound communication headers will be handed over to all message receivers and integration flow steps whereas properties will remain within integration flow and will not be handed over to receivers.

Message Body - In the Body tab of the Content Modifier, you specify the content expected in the outgoing message. To access Body, we can use ${**in.body**}



Message Transformers: Content Modifier

Example of Content Modifier usage:

Suppose the incoming message has the following information:

<order> <book> <BookID>A1000</BookID><Count>5<Count></book></order>

- Header Tab:

Name	Type	Value
vendor	constant	ABC Corp
delivery date	constant	25062013

- Body Tab: Keep a placeholder for the header information to modify the content as shown below:

<invoice><vendor>${**header.vendor**}</vendor>${**in.body**}<deliverydate>${**header.delivery_date**}</delivery></invoice>

- Output would look like this:

```
<invoice>
<vendor>ABC Corp</vendor>
    <order>
    <book>
        <BookID>A1000</BookID>
<Count>5<Count>
</book>
</order>
<deliverydate>25062013</deliverydate>
</invoice>
```

Message Transformers: Converter

If you have an input message in CSV/XML/JSON format, you need to convert it into XML or JSON or CSV format to use it in the subsequent steps of the integration flow. You can use the converter to achieve this message transformation.

Converter step can be used for the below conversions.

- CSV to XML conversion
- XML to CSV conversion
- XML to JSON conversion
- JSON to XML conversion
- XML to EDI conversion
- EDI to XML conversion

Converter: CSV to XML

If you have an input message in CSV format, you need to convert it into XML format to use it in the subsequent steps of the integration flow. You can use the CSV to XML converter to achieve this message transformation.

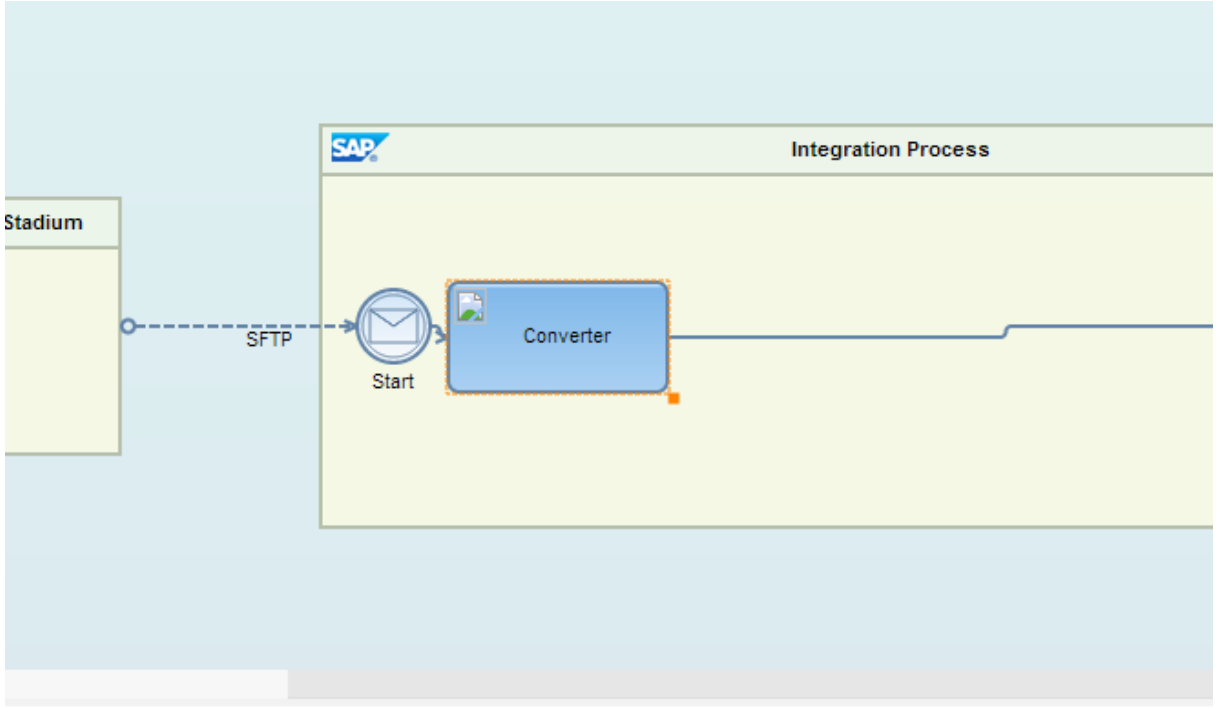
In the Properties view, enter the details based on the descriptions for the fields given in the table below.

1. Choose Browse and select the file path to the XML schema file that is used as the basis for message transformation. The XML file schema format is used as the basis for creation of the payload. This file has to be in the package source.main.resources.xsd

2. XPath in the XML Schema File where the content from CSV has to be placed.

3. The corresponding record in the CSV file that has to be considered for conversion. This entry is the first text in every new record of the CSV. Note: If this value is not specified then all the records would be considered for conversion.

4. The field separator used in the CSV file



CSV to XML Converter

Parameters to convert CSV to XML

1	XML Schema File Path:	/Project_Group_XX_Exercise_7/src/main/resources/xsd/Stadium_XX.xsd
2	XPath field location to replace:	StadiumMessage/FileName
3	Record identifier in CSV:	TICKET
4	Field Separator in CSV:	,

Converter: CSV to XML example

Example

Consider a CSV file that is in the following format:

COMPETENCY,Role Specific,Computer Skills,"Skilled in the use of computers, adapts to new technology, keeps abreast of changes, learns new programs quickly, uses computers to improve productivity."

Consider the Schema XML File in the following format:

```
<?xml  version="1.0" encoding="UTF8"?>
<CompetencyList><Competency><id></id><name></name><description></description></Competency></CompetencyList>
```

Value for the field XPath of Record Identifier in XSD is given as CompetencyList/Competency.

After it is processed by the CSV to XML converter, the XML output appears in the following format:

```
<?xml version="1.0" encoding="UTF-8"?>
<CompetencyList><Competency><id>Role Specific</id><name>Computer Skills</name><description>Skilled in the use of computers, adapts to new technology, keeps abreast of changes, learns new programs quickly, uses computers to improve productivity.</description></Competency></CompetencyList>
```

Converter: XML to CSV

If you have an input message in XML format, you need to convert it into CSV format to use it in the subsequent steps of the integration flow. You can use the XML to CSV converter to achieve this message transformation.

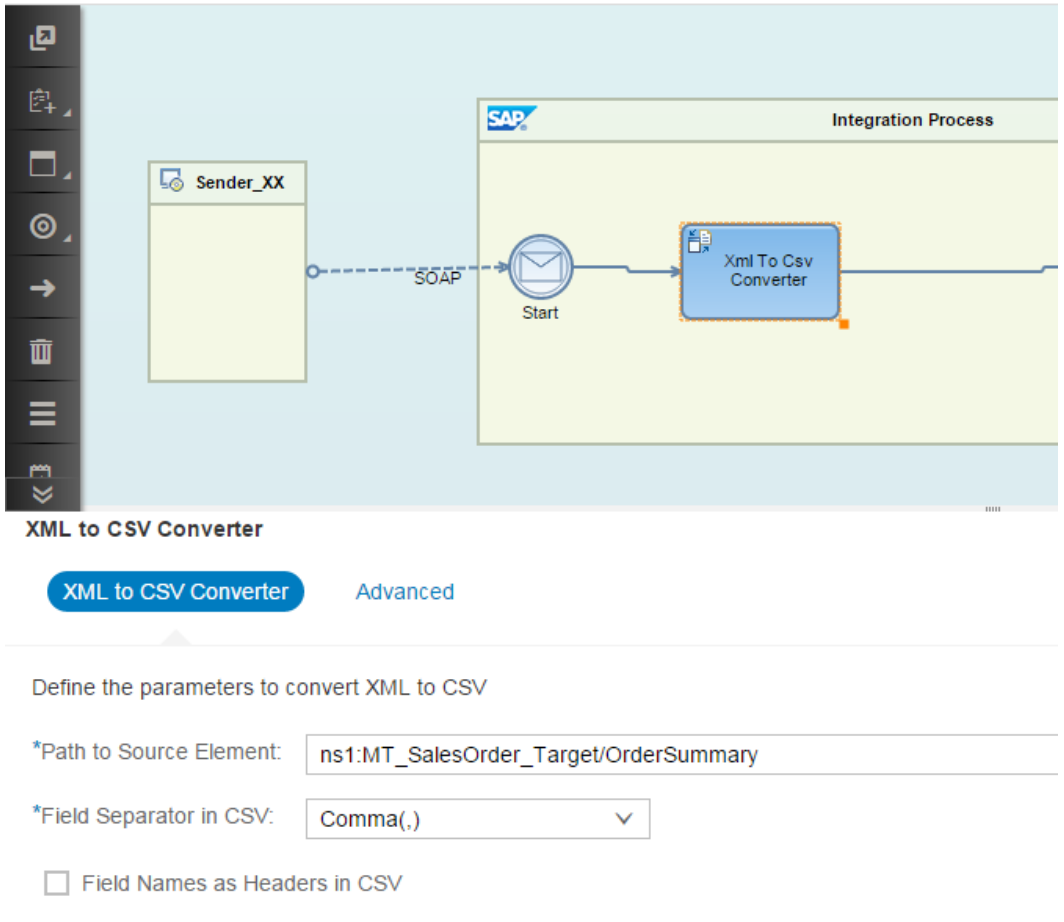
In the Properties view, enter the details based on the descriptions for the fields given in the table below.

- Provide the Xpath of the Source element which you want to convert to the CSV

1 XPath in the XML Schema File where the content from CSV has to be placed.

2 Field Separator for the CSV file. Each element will be separated with the Field Separator in the output file

3 Click on the checkbox if you want the element names of the XML in the output file



Converter: XML to CSV example

Example

Consider a XML file that is in the following format:

```
  <?xml version="1.0" encoding="UTF-8"?>
- <ns4:MT_SalesOrder_Target xmlns:ns4="http://salesorder.sap.com">
  - <OrderSummary>
        <OrderNumber>12345</OrderNumber>
        <CustomerName>Ajay</CustomerName>
        <OrderAmount>200</OrderAmount>
        <Currency>USD</Currency>
        <TaxAmount>20</TaxAmount>
    </OrderSummary>
  - <OrderSummary>
        <OrderNumber>12346</OrderNumber>
        <CustomerName>Ajay</CustomerName>
        <OrderAmount>200</OrderAmount>
        <Currency>USD</Currency>
        <TaxAmount>20</TaxAmount>
    </OrderSummary>
  </ns4:MT_SalesOrder_Target>
```

Value for the field path for the source element.

ns1:MT_SalesOrder_Target/OrderSummary

After it is processed by the XML to CSV converter, the CSV output appears in the following format:

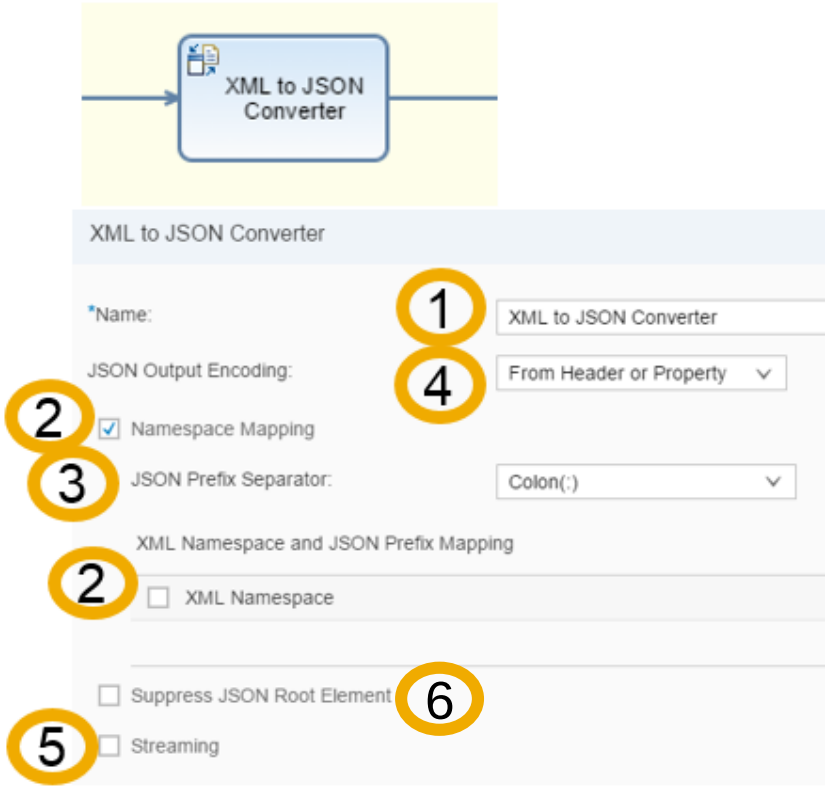
```
12345,Ajay,200,USD,20
12346,Ajay,200,USD,20
```

Converter: XML to JSON

If you have an input message in XML format, you need to convert it into JSON format to use it in the subsequent steps of the integration flow. You can use the XML to JSON converter to achieve this message transformation.

In the Properties view, enter the details based on the descriptions for the fields given in the table below.

- 1 Enter the name of the Convertor step.
- 2 Mapping of XML namespace to the JSON prefix. These parameters are only filled if “Use Namespace mapping” is selected. The JSON namespace/prefix must start with a letter and can contain 0-9,aA-zZ.
- 3 JSON prefix separator to separate the JSON prefix from local part. Colon(:), Comma (,), Dot(.), Pipe(|), Semicolon(;) and space( ) are supported. The value used must never be used in the JSON prefix or local name
- 4 Enter the JSON output encoding. The default value is from *Header or Property.*
- 5 Streaming: Click on checkbox for large xml messages. The streaming has the option of converting all or specific elements of incoming XML document.
- 6 Each individual tag of an XML document is processed consecutively, one after the other, independent from where in the overall structure the tag occurs and how often it occurs in the structure (multiplicity).
- Choose this option to create the JSON message without the root element tag.



Converter: XML to JSON

The conversion from XML to JSON follows the following rules

- Elements with mixed content (for example. <A>mixed1_value<B>valueB</B>mixed2_value</A>) are not supported. You will get incorrect results for XML to JSON: {"A":{"B":"valueB","$":"mixed1_valuemixed2_value"}}.
- No namespace declaration is written into the JSON document
- Tabs, spaces, and new lines between elements and attributes are ignored
- If you have an element with a namespace but without an XML prefix whose namespace is not contained in the XML-Namespace-to-JSON-Prefix map, you get an exception: <A xmlns="http://test"/> -> IllegalStateException Invalid JSON namespace: http://test.

Converter: JSON to XML

If you have an input message in JSON format, you need to convert it into XML format to use it in the subsequent steps of the integration flow. You can use the JSON to XML converter to achieve this message transformation.

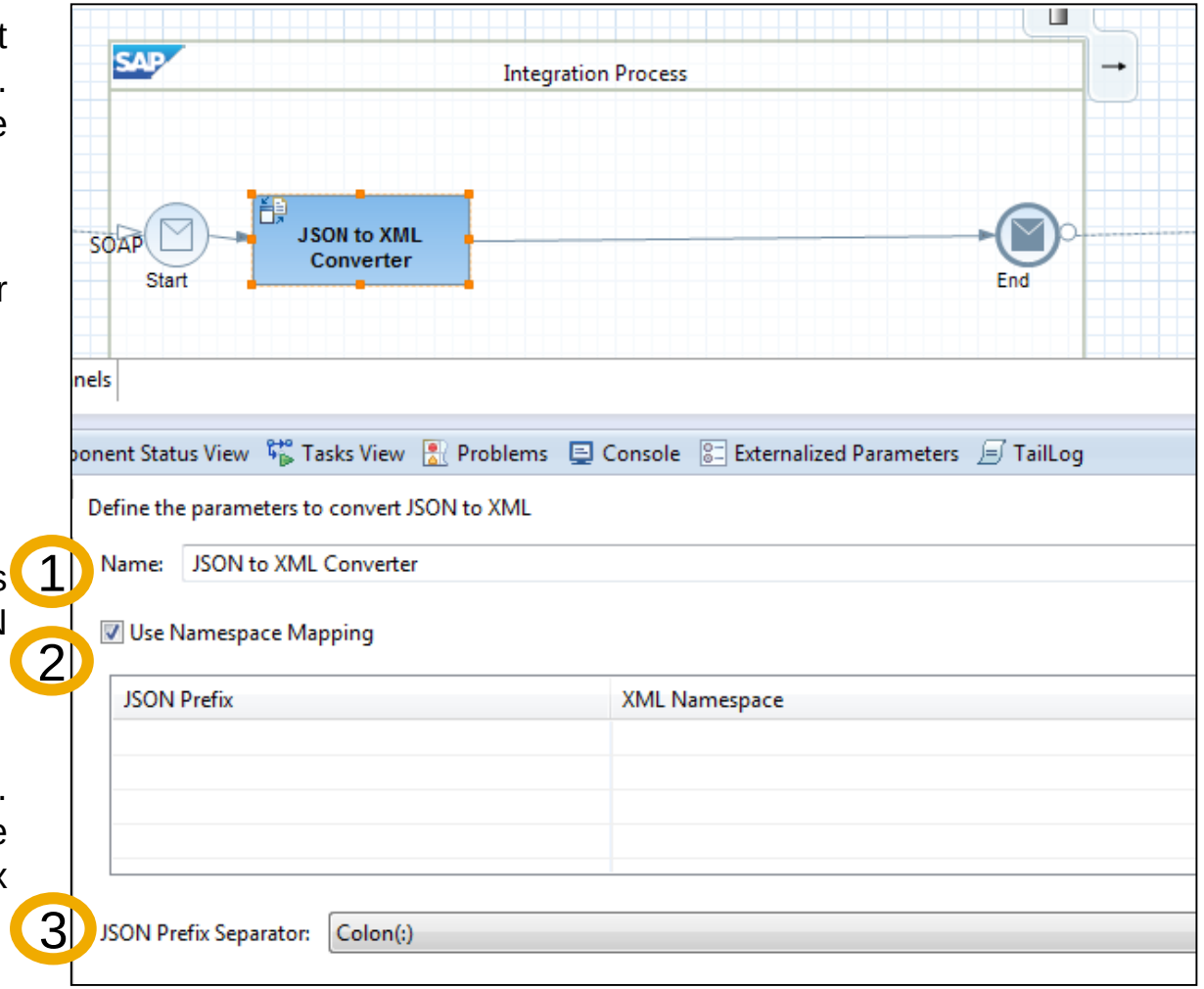
In the Properties view, enter the details based on the descriptions for the fields given in the table below.

1

2 Enter the Name of the convertor step.

3 Mapping of XML namespace to the JSON prefix. These parameters are only filled if “Use Namespace mapping” is selected. The JSON namespace/prefix must start with a letter and can contain 0-9,aA-zZ

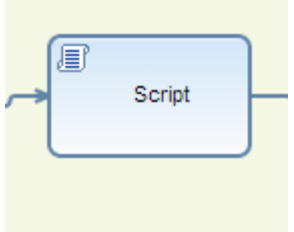
- JSON prefix separator to separate the JSON prefix from local part. Colon(:), Comma (,), Dot(.), Pipe(|), Semicolon(;) and space( ) are supported. The value used must never be used in the JSON prefix or local name



Message Transformers: Script

This Process Step is used to execute a java or groovy script for message processing or transformation.

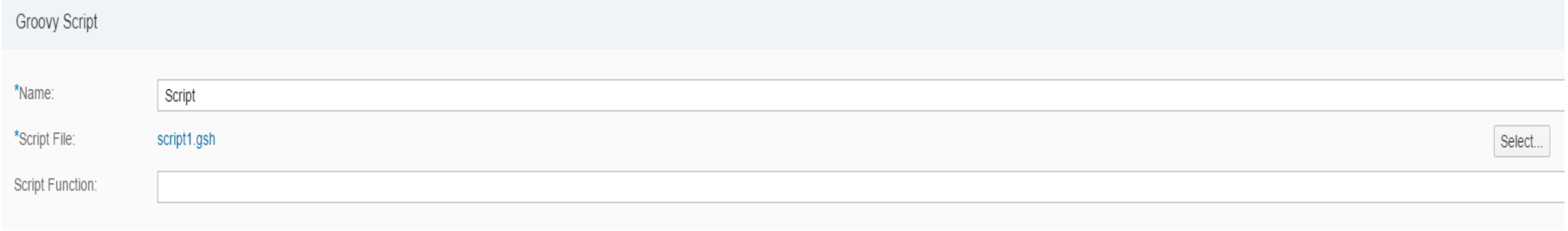
You can add or modify Header, Body and Property using below interfaces on the "message" object.



- public java.util.Map<java.lang.String,java.lang.Object> getHeaders()
- public void setHeaders(java.util.Map<java.lang.String,java.lang.Object> exchangeHeaders)
- public void setHeader(java.lang.String name, java.lang.Object value)

- public java.lang.Object getBody()
- public void setBody(java.lang.Object exchangeBody)

- public java.util.Map<java.lang.String,java.lang.Object> getProperties()
- public void setProperties(java.util.Map<java.lang.String,java.lang.Object> exchangeProperties)○
- public void setProperty(java.lang.String name, java.lang.Object value)

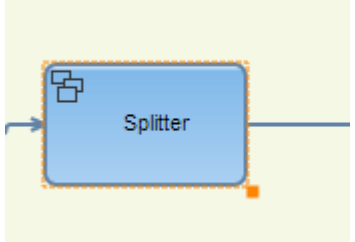


Message Routing

- Splitter
- Router
- Multicast
- Join
- Gather
- Aggregator

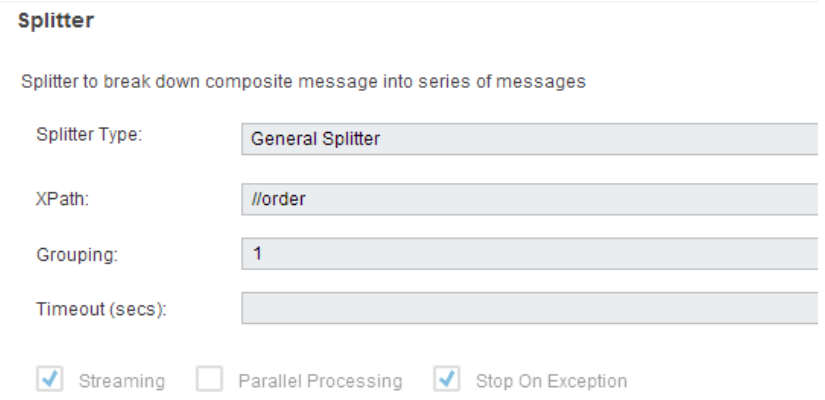
Message Routing: Splitter

This Process Step is used to break down a composite message into a series of individual messages and send them to a receiver. You split the composite message based on the message type, for example, IDoc or PKCS#7/CMS Signature-Content, or the manner in which the message is to be split, for example, general or iterative splitter



In the Properties view, select a splitter type.

- **IDoc Splitter** is used for composite IDoc messages to be split into a series of individual IDoc messages with the enveloping elements of the composite IDoc message. As od now, output of this step should be Idoc adapter only.
- **PKCS#7/CMS Signature-Content Splitter** is used when an agent sends a PKCS7 Signed Data message that contains signature and content. This splitter type breaks down the signature and content into separate files.
- **General Splitter** splits a composite message comprising of N messages into N individual messages each containing one message with the enveloping elements of the composite message.
- **Iterative Splitter** splits a composite message into a series of messages without copying the enveloping elements of the composite message.
- **EDI splitter** splits inbound bulk EDI messages, and during processing you can configure the splitter to validate and acknowledge the inbound messages.



Splitter

Example

The following XML payload structure is input into the Splitter:

<orders> <order> <clientId>I0001</clientId> <count>100</count> </order> <order> <clientId>I0002</clientId> <count>10</count> </order></orders>

If we use the XML node order as a token, the XML payload is split into two output files.

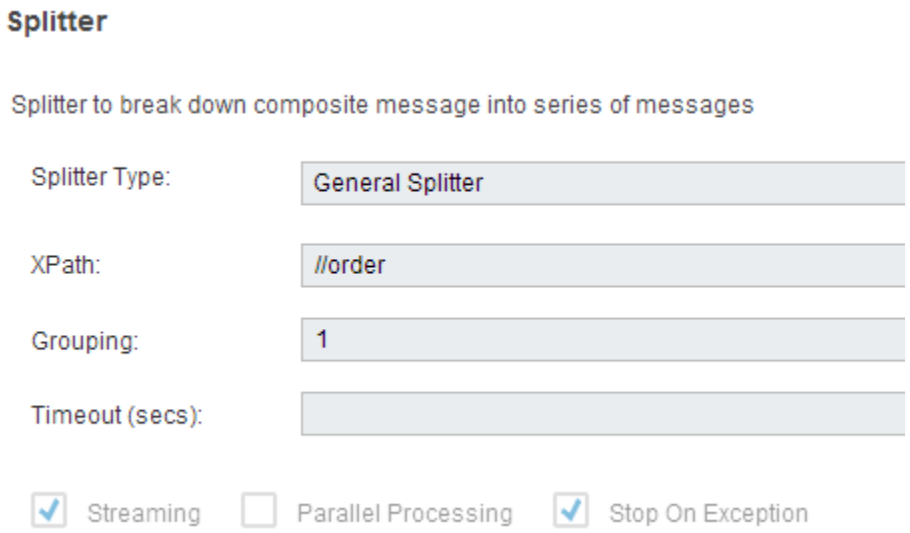
Output1.xml contains:

<order> <clientId>I0001</clientId> <count>100</count></order>

Output2.xml contains:

<order> <clientId>I0002</clientId> <count>10</count></order>

.



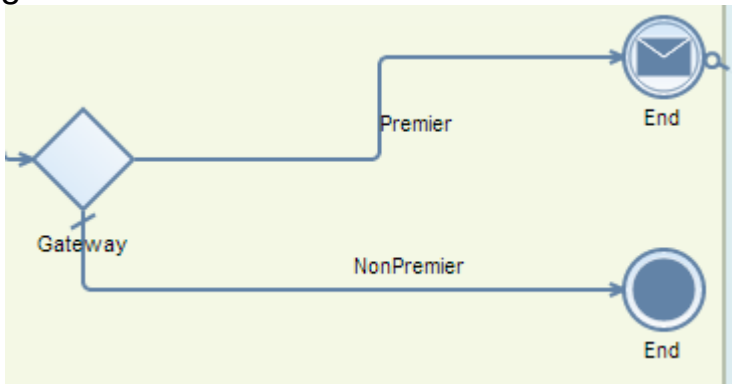
Message Routing: Router

This Process Step is used to specify conditions based on which the messages are routed to a receiver or an interface during runtime. If the message contains XML payload, you form expressions using the Xpath-supported operators. If the message contains non-XML payload, you form expressions using the operators shown in the table below:

Example:

A condition with expression ${header.SenderId} regex '1.*' routes all the messages having

Sender ID starting with 1'.



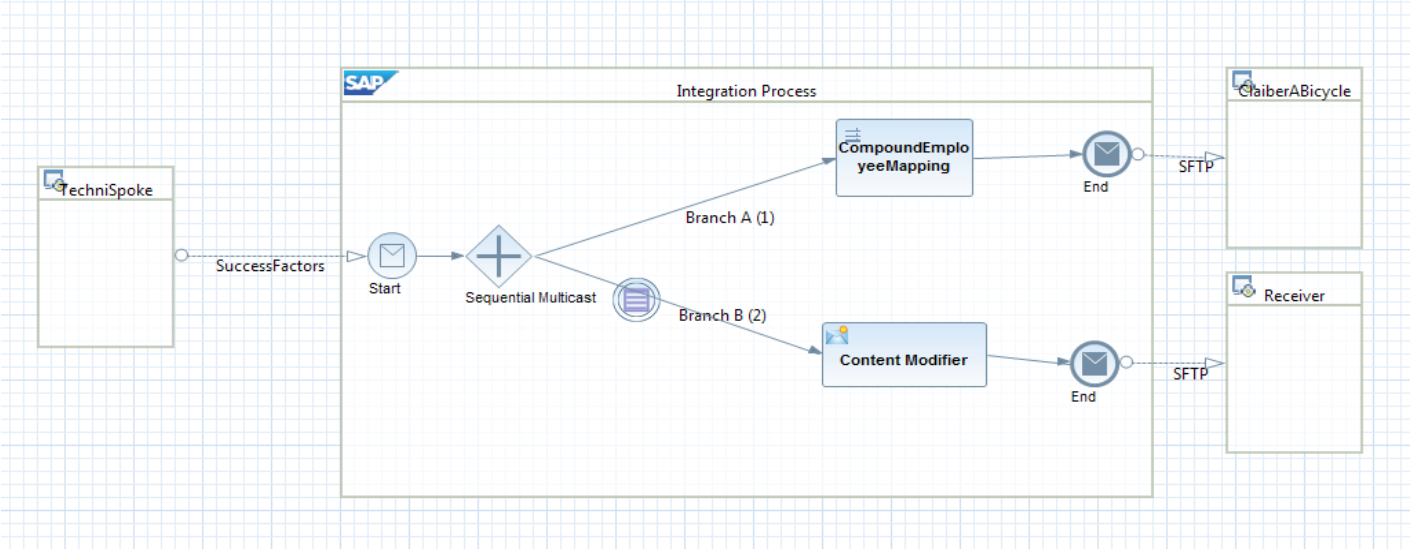
Operator	Example
=	${header.SenderId} = '1'
!=	${header.SenderId} != '1'
>	${header.SenderId} > '1'
>=	${header.SenderId} >= '1'
<	${header.SenderId} < '1'
<=	${header.SenderId} <= '1'
and	${header.SenderId}= '1' and ${header.ReceiverId} = '2'
or	${header.SenderId}= '1' or ${header.ReceiverId}= '2'
contains	${header.SenderId} contains '1'
not contains	${header.SenderId} not contains '1'
in	${header.SenderId} in '1,2'
not in	${header.SenderId} not in '1,2'
regex	${header.SenderId} regex '1.*'
not regex	${header.SenderId} not regex '1.*'

Message Routing: Multicast

Multicast enables you to send the same message to more than one receiver. This allows you to perform multiple operations on the same message in a single integration flow. Without multicast, you require separate integration processes to perform different operations on the same incoming message.

1 In case of the Sequential Multicast you have an option to define the order in which the messages can be executed.

- You would have to use the Join and Gather steps in case the messages from the different branches have to be combined to one single message based on an aggregation algorithm.



1

Specify the sequence of execution of branches

Sequence Order	Sequence Name
1	Branch A
2	Branch B

Message Routing: Join and Gather

The *Join* element enables you to bring together the messages from different routes before combining them into a single message. You use this in combination with the *Gather* element. *Join* only brings together the messages from different routes without affecting the content of messages.

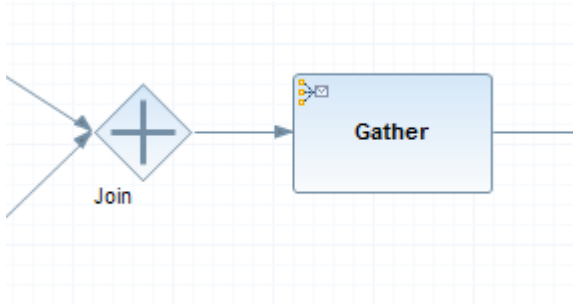
The *Gather* step enables you to merge more than one message flow in an integration process. You define some conditions to merge data based on the type of messages that you are gathering using the *Gather* step.

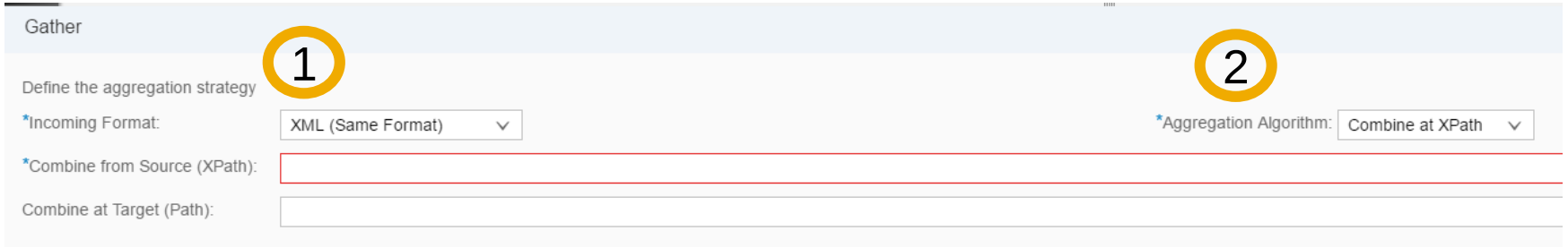
1 You can choose to gather:

- XML messages of different format, XML messages of the same format, Plain text messages

2 Based on this, you choose the strategy to combine the two messages.

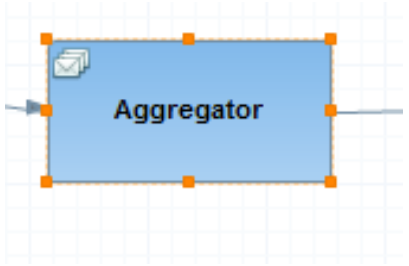
- For XML messages of the same format, you can combine without any conditions or specify the XPath to the node at which the messages have to be combined.
- For XML messages of different formats, you can only combine the messages.
- For plain text messages, you can only specify concatenation as the combine strategy.





Message Routing: Aggregator

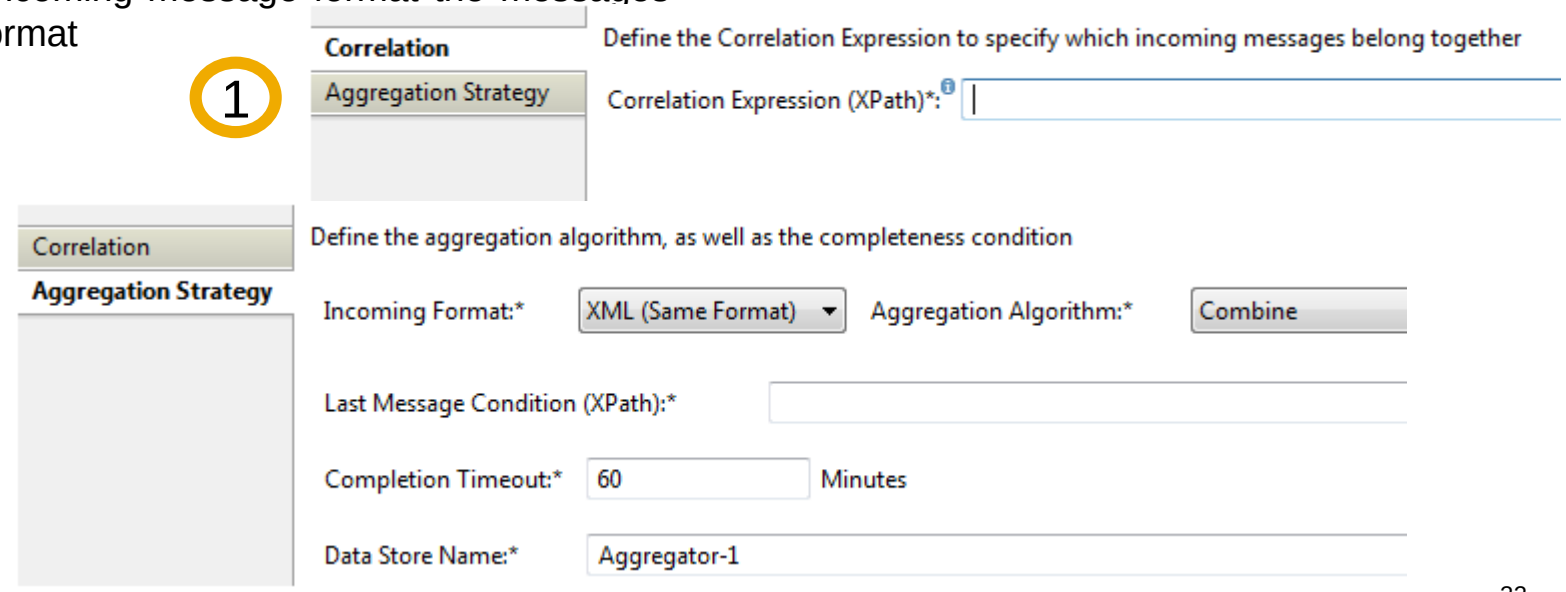
The Aggregator Step allows merging of multiple messages in the sequence flow based on an Aggregation Strategy and a Correlation Condition



In the Properties view, select a splitter type.

1. **Correlation:** Defines the condition that is used to correlate the different messages in the integration flow

2. **Aggregation Strategy:** Based on the type of incoming message format the messages can be combined to a multi-mapping message format



Message Validators

- Validator

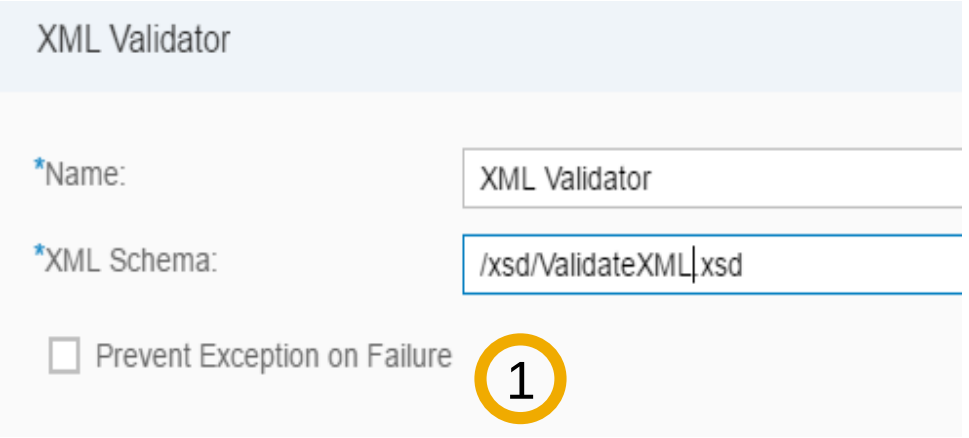
Message Validators: XML Validator

Validating Message Payload against XML Schema

The XML validator validates the message payload in XML format against the configured XML schema.

This step is used to assign XML schema (XSD files) to validate the message payload in a process step. The validator checks the message payload against configured XML schema, and report discrepencies in message payload. If the validation fails, the system stops the whole message processing by default.

1 If you want to continue the processing even if the system encounters error while validating, then select the check box Prevent Exception on Failure.

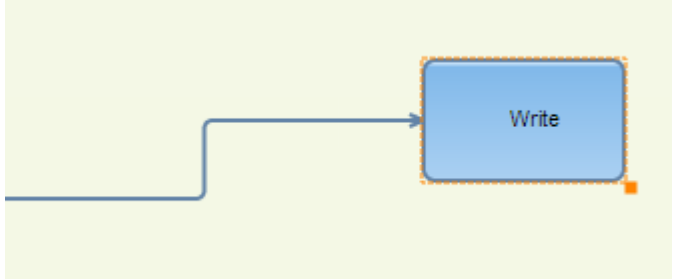


Message Persistence

- Data store Operations
- Write Variables

Datastore Operations

In common scenarios, SAP Cloud Platform Integration receives messages from one participant and forwards them to other participants. There can be scenarios when, SAP CPI cannot directly communicate with the participant, maybe due to firewall settings or the participant has not hosted any endpoint that can be invoked from SAP CPI. Instead of forwarding the messages to the unreachable participant, it can store the messages in a transient datastore and the participant periodically polls the transient datastore for the stored messages



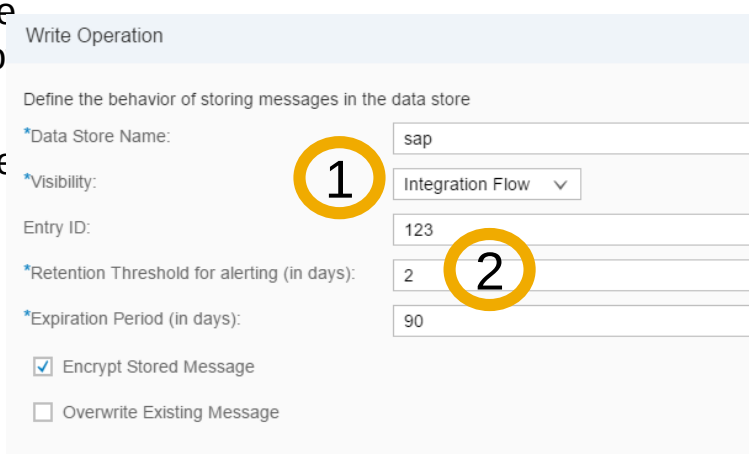
Note: The transient datastore temporarily stores messages for later processing.

Write operation: Using this operation, messages from the sender participants are stored into the datastore. You need to configure the behavior of the stored message, for example, the number o days in which the stored messages are deleted and so on.

1 You can define the unique ID in the header SapDataStoreId. If the ID is not defined, the datastore component generates an ID and sets the header.

- **Global:** The datastore can be shared across all integration flows deployed on one tenant.
- **Integration Flow(Local):** The datastore can only be used by one integration flow.

2 Specifies a time period in which message should be fetched, before an alert is raised. Default 2 days



Note: If you select the option of Overwrite Existing Message, it enables overwriting of an existing persisted message with the same ID.

Datastore Operations

2 **Get operation:** Using this operation, the receiver fetches a message from the datastore using the existing datastore name and message ID specified while storing the messages. You can directly specify the value of message ID or provide an expression such as, ${header.<headername containing message ID>} or ${xpath.<node containing message ID>}.

Note: If you are specifying the message ID in the header SapDataStoreId as well as the Properties view, the value specified in the Properties view takes the precedence.

3 **Select operation:** Using this operation, the receiver fetches messages in bulk from the datastore. You configure the parameters that define the behavior of fetching the messages per poll from the datastore. You can choose to allow the messages to be deleted once the receiver completes fetching the messages. Otherwise, the message remains in the datastore unless it is explicitly deleted using the Delete operation. The format of the fetched message is:

<?xml version="1.0" encoding="UTF-8"?>

<messages> <message id="[id.1]"> [content.1] </message>

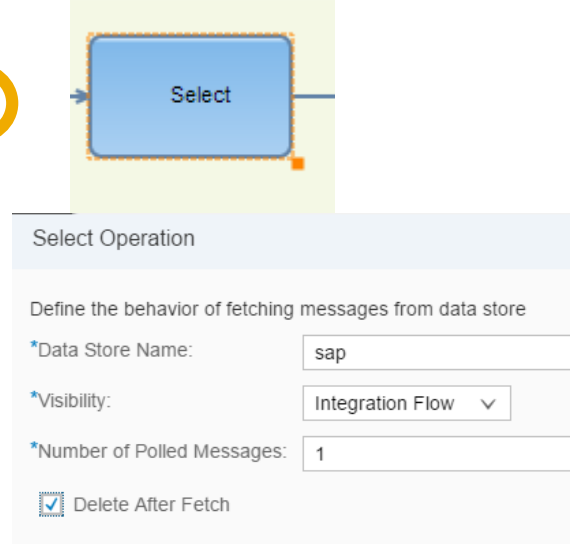
<message id="[id.2]"> [content.2] </message> <messages>

This operation fetches several messages at a time, and creates one bulk message. You can specify the number of messages to be fetched per poll either on the Properties view of Select operation or set the header SapDataStoreMaxResults.

Note: If you are specifying the number of messages to be fetched per poll in the header as well as the Properties view of Select operation, the header will take the precedence

2

3

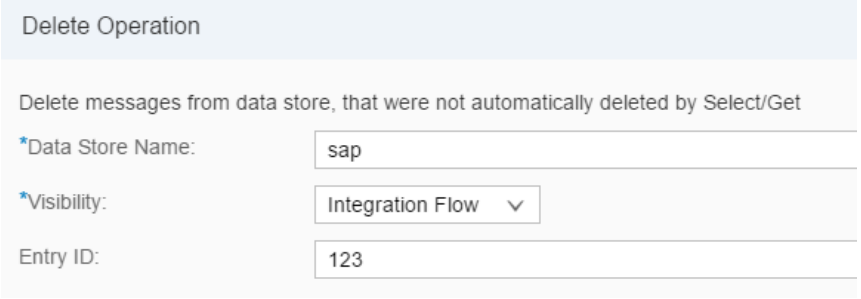


Datastore Operations

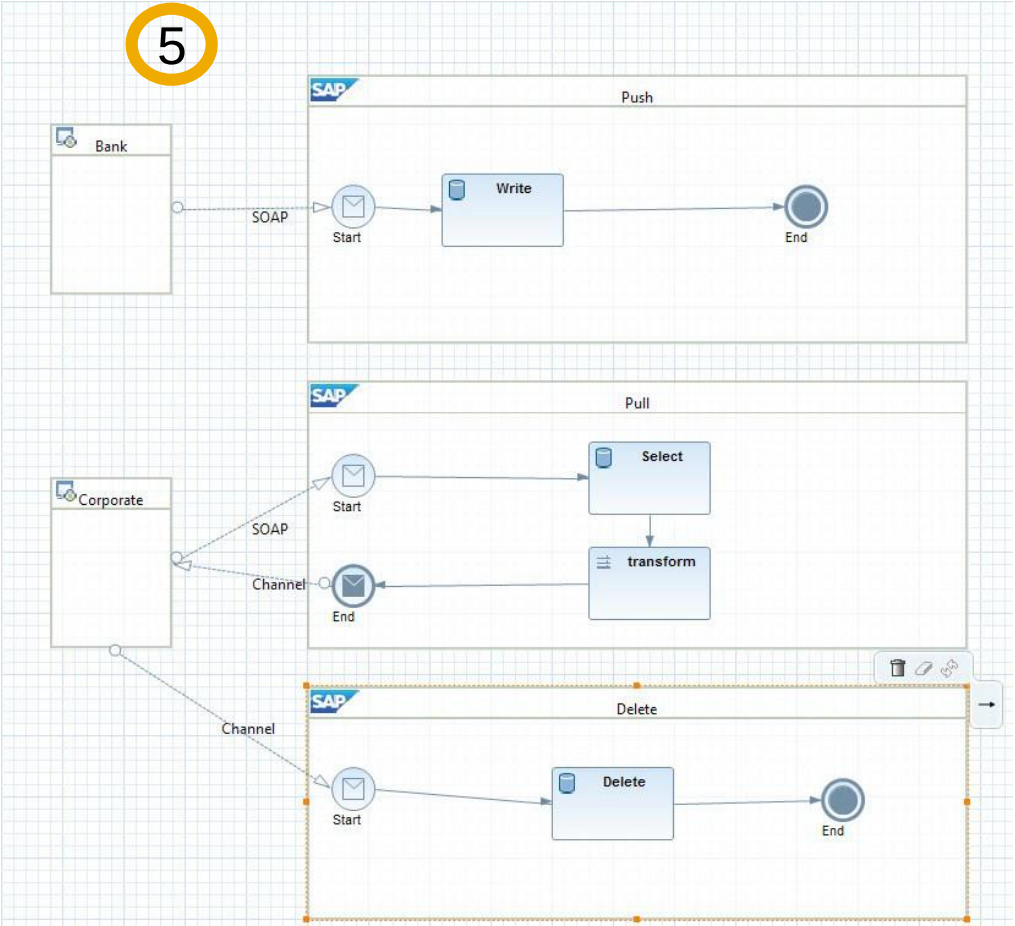
4 **Delete operation:** Using this operation, you can trigger the deletion of message(s) from the datastore. You can specify the message ID of the message to be deleted either in the header SapDataStoreId or in the Message ID field of the Properties view. Instead of directly providing the message ID, you can also enter an expression such as ${header.<headername containing message ID>} or ${xpath.<node containing message ID>} in the Message ID field. To delete multiple messages, use an Xpath condition with org.w3c.dom.NodeList in the SapDataStoreId header.

5 Example of Datastore usage is shown in the Screenshot

4

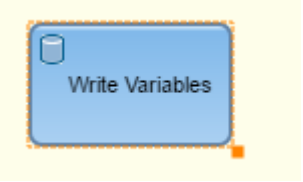


5



Write Variables- Setting Global Variables

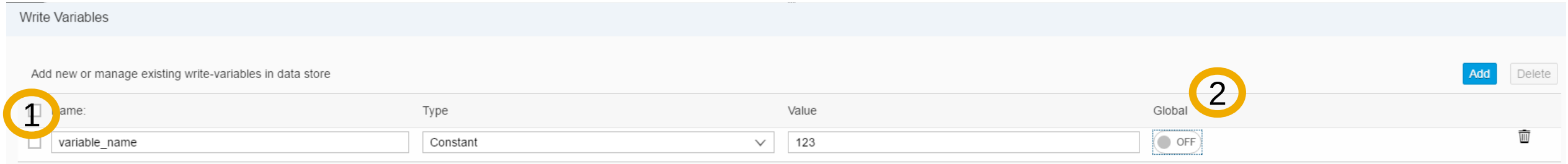
The Write variable Step allows integration developers to store values that can used across message flows within the same iFlow OR across different iFlows within a Tenant.



1 Define a variable that can be used across message flows within the same iFlow. You can access the value stored using Write Variableles from a Content Modifier step.

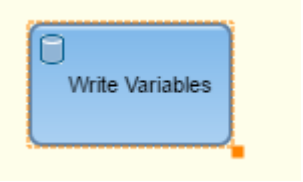
2 You can use this option to allow access to the variable across different iFlows within the same Tenant

Note: The variable is accessible in all iFlows if Global is turned On(WebUI) or you tick the check box "*Global Scope*"(Eclipse)



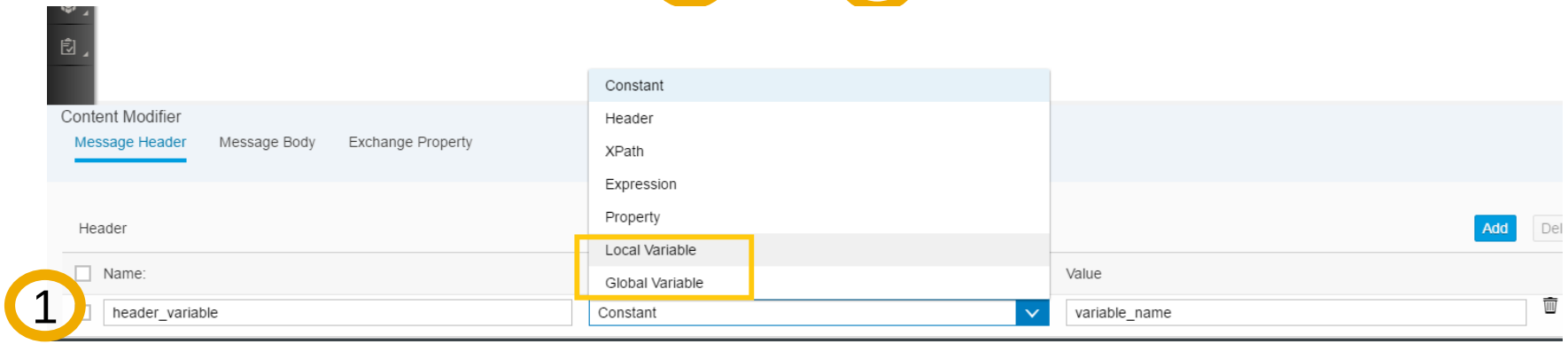
Write Variables- Accessing Global Variables in iFlow

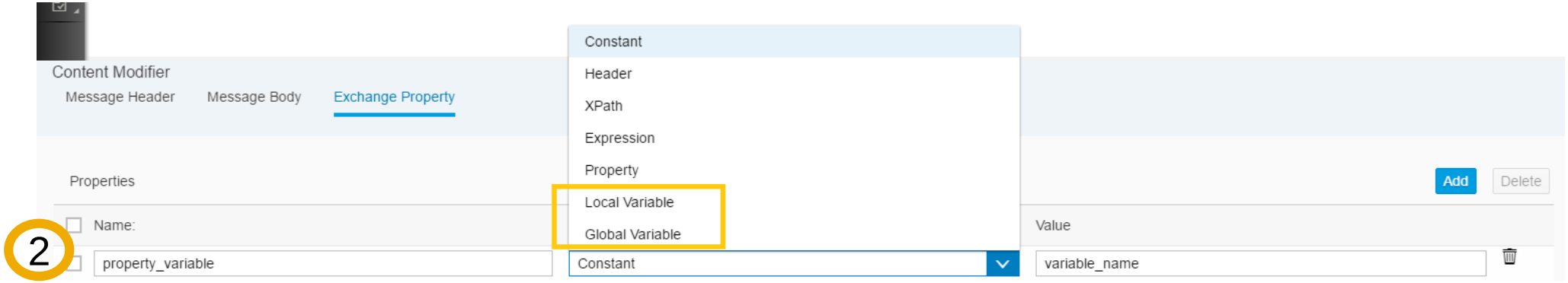
The Write variable Step allows integration developers to store values that can used across message flows within the same iFlow OR across different iFlows within a Tenant.



These variables can be accessed in iFlow using Header or Properties in a content modifier

1 2





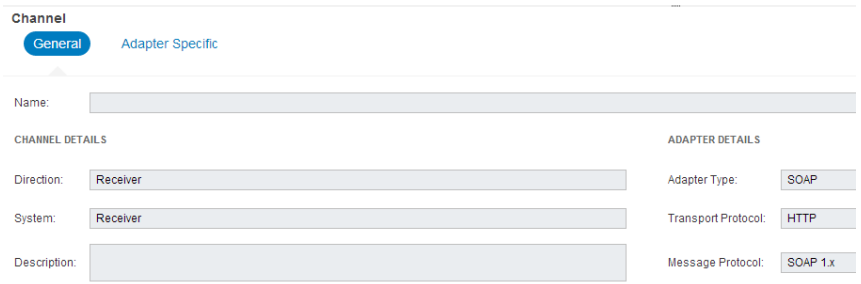
Tasks

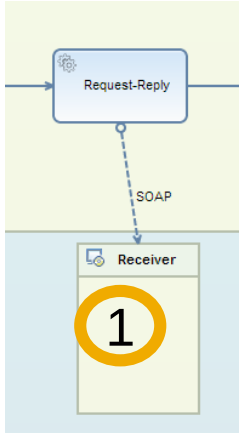
- Service Call
  - Request Reply
  - Content Enricher
  - Send
- Process Call

Tasks: Service Call > Request-Reply

Service Call is used to call an external system. Such calls enable transaction of data from or to the target system. It can be used for following types of operations:

1. **Request-Reply**: This variant of Service Call is used to enable request and reply interactions between sender and receiver systems. The response received as reply is passed on to the next step.





Note: For Details about above mentioned parameters, kindly refer to HANA Cloud Platform, integration service Adapters Slides

Tasks: Service Call > Content Enricher (Combine)

The content enricher adds the content of a payload with the original message in the course of an integration process. This converts two separate messages into a single enhanced payload. This feature enables you to make external calls during the course of an integration process to obtain additional data, if any.

Consider the first message in the integration flow as the original message and the message obtained by making an external call during the integration process as the lookup message. You can choose between two strategies to enrich these two payloads as a single message:

- **Combine**

Consider the following original and lookup messages.

Original Message

```
<EmployeeList>
        <Employee>
                <id>111</id>
                <name>Santosh</name>
                <external_id>ext_111</external_id>
        </Employee>
        <Employee>
                <id>22</id>
                <name>Geeta</name>
                <external_id>ext_222</external_id>
        </Employee>
</EmployeeList>
```

Lookup Message

```
<EmergencyContacts>
        <contact>
                <c_id>1</c_id>
                <c_code>ext_111</c_code>
                <isEmergency>0</isEmergency>
                <phone>9999</phone>
                <street>1st street</street>
                <city>Gulbarga</city>
        </contact>
        <contact>
                <c_id>2</c_id>
                <c_code>ext_111</c_code>
                <isEmergency>1</isEmergency>
                <phone>1010</phone>
                <street>23rd Cross</street>
                <city>Chitapur</city>
        </contact>
        <contact>
                <c_id>3</c_id>
                <c_code>ext_333</c_code>
                <isEmergency>1</isEmergency>
                <phone>007</phone>
                <street></street>
                <city>Raichur</city>
        </contact>
</EmergencyContacts>
```

Content Enrichment using **Combine**

Enriched Message

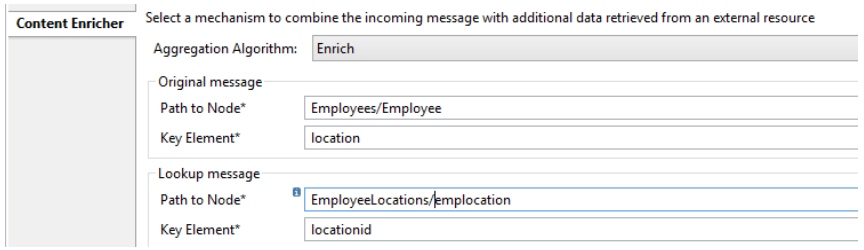
```
<multimap:messages xmlns:multimap="http://sap.com/xi/XI/SplitAndMerge">
<message1>
<EmployeeList>
        <Employee>
                <id>111</id>
                <name>Santosh</name>
                <external_id>ext_111</external_id>
        </Employee>
        <Employee>
                <id>22</id>
                <name>Geeta</name>
                <external_id>ext_222</external_id>
        </Employee>
</EmployeeList>
</message1>
<message2>
<EmergencyContacts>
        <contact>
                <c_id>1</c_id>
                <c_code>ext_111</c_code>
                <isEmergency>0</isEmergency>
                <phone>9999</phone>
                <street>1st street</street>
                <city>Gulbarga</city>
        </contact>
        <contact>
                <c_id>2</c_id>
                <c_code>ext_111</c_code>
                <isEmergency>1</isEmergency>
                <phone>1010</phone>
                <street>23rd Cross</street>
                <city>Chitapur</city>
        </contact>
        <contact>
                <c_id>3</c_id>
                <c_code>ext_333</c_code>
                <isEmergency>1</isEmergency>
                <phone>007</phone>
                <street></street>
                <city>Raichur</city>
        </contact>
</EmergencyContacts>
</message2>
</multimap:messages xmlns:multimap="http://sap.com/xi/XI/SplitAndMerge">
```

Tasks: Service Call > Content Enricher (Enrich)

- **Enrich**

Enrich offers you control on how you can merge the original and lookup message. Unlike Combine, it doesn't directly append the lookup structure to the source structure, rather it enhances source structure with Lookup structure placing the respective nodes at enrich points(defined as key in Enrich strategy) in an intelligent way.

In the same example as for Combine, we consider the node <ext_111> as the reference to enrich the original message with the lookup message.



Content Enrichment using **Enrich**

Original Message

```
<EmployeeList>
        <Employee>
                <id>111</id>
                <name>Santosh</name>
                <external_id>ext_111</external_id>
        </Employee>
        <Employee>
                <id>22</id>
                <name>Geeta</name>
                <external_id>ext_222</external_id>
        </Employee>
</EmployeeList>
```

Lookup Message

```
<EmergencyContacts>
        <contact>
                <c_id>1</c_id>
                <c_code>ext_111</c_code>
                <isEmergency>0</isEmergency>
                <phone>9999</phone>
                <street>1st street</street>
                <city>Gulbarga</city>
        </contact>
        <contact>
                <c_id>2</c_id>
                <c_code>ext_111</c_code>
                <isEmergency>1</isEmergency>
                <phone>1010</phone>
                <street>23rd Cross</street>
                <city>Chitapur</city>
        </contact>
        <contact>
                <c_id>3</c_id>
                <c_code>ext_333</c_code>
                <isEmergency>1</isEmergency>
                <phone>007</phone>
                <street></street>
                <city>Raichur</city>
        </contact>
</EmergencyContacts>
```

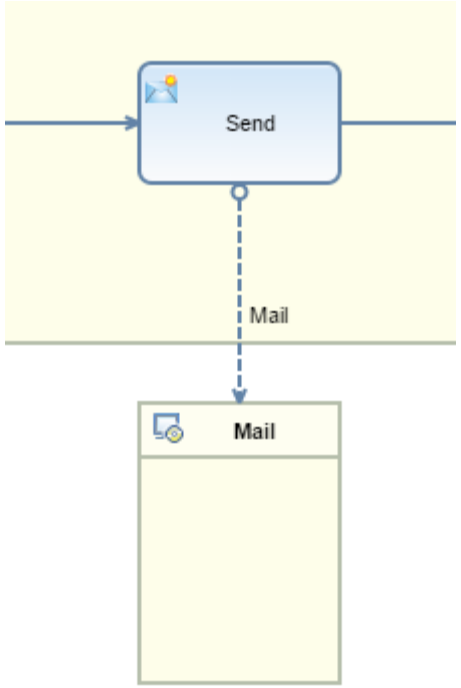
```
<EmployeeList>
        <Employee>
                <id>111</id>
                <name>Santosh</name>
                <external_id>ext_111</external_id>
                <contact>
                        <c_id>1</c_id>
                        <c_code>ext_111</c_code>
                        <isEmergency>0</isEmergency>
                        <phone>9999</phone>
                        <street>1st street</street>
                        <city>Gulbarga</city>
                </contact>
                <contact>
                        <c_id>2</c_id>
                        <c_code>ext_111</c_code>
                        <isEmergency>1</isEmergency>
                        <phone>1010</phone>
                        <street>23rd Cross</street>
                        <city>Chitapur</city>
                </contact>
        </Employee>
        <Employee>
                <id>22</id>
                <name>Geeta</name>
                <external_id>ext_222</external_id>
        </Employee>
</EmployeeList>
```

Tasks: Service Call > Send

Send step is used to configure a service call to a receiver system for scenarios and adapters where no reply is expected.

This step can be used in combination with the following adapter types (for the channel between the send step and the receiver):

- Mail adapter
- SFTP adapter



Events

- Start Message
- End Message
- Terminate Message
- Error Start Event
- Error End Event
- Escalation
- Start Event
- End Event
- Timer

Events

1 **Start Message** - First step of an Integration Process, in case, it is triggered by Sender and not scheduled (with Timer)

2 **End Message –** Last step of an Integration Process before message is delivered to the Receiver

3 **Terminate Message -** Used to terminate the process without sending message to the Receiver.

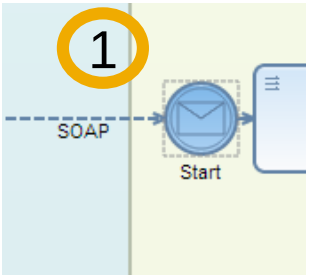
4 **Error Start Event -** Used only in the Exception SubProces and is used as the start in case any exception occurs in the Integration Process where it is embedded.

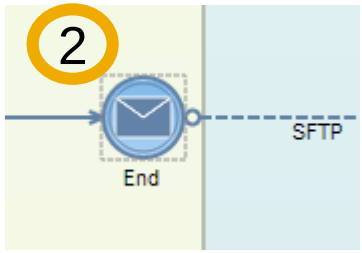
5 **Error End Event -** Used to throw the exception back to default exception handlers.

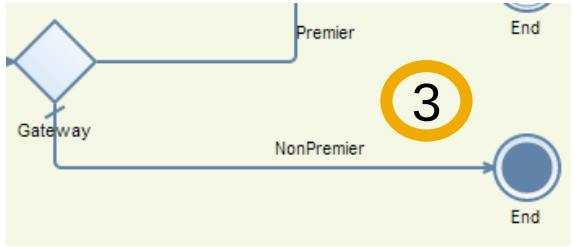
6 **Escalation -** Stops message processing. For synchronous messages, an error messages is sent to the sender.

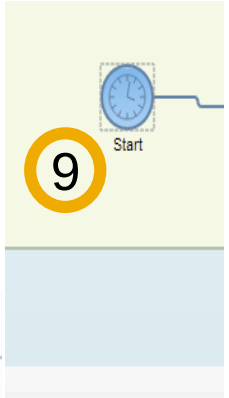
7 **Start Event -** Start of Local Integration process 8 **End Event -** End of local Integration process

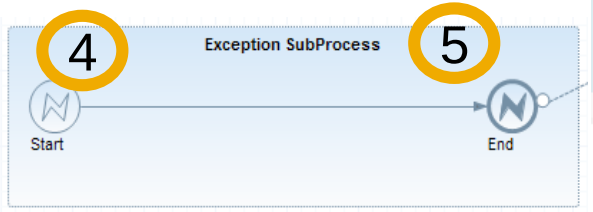
9 **Timer-** User to configure a process to automatically start in a particular schedule,

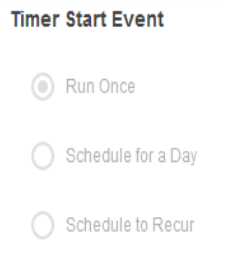












Security Elements

- Encryptor
- Decryptor
- Message Signer
- Signature Verifier

Security Elements: Content Encryptor & Content Decryptor

Content Encryptor

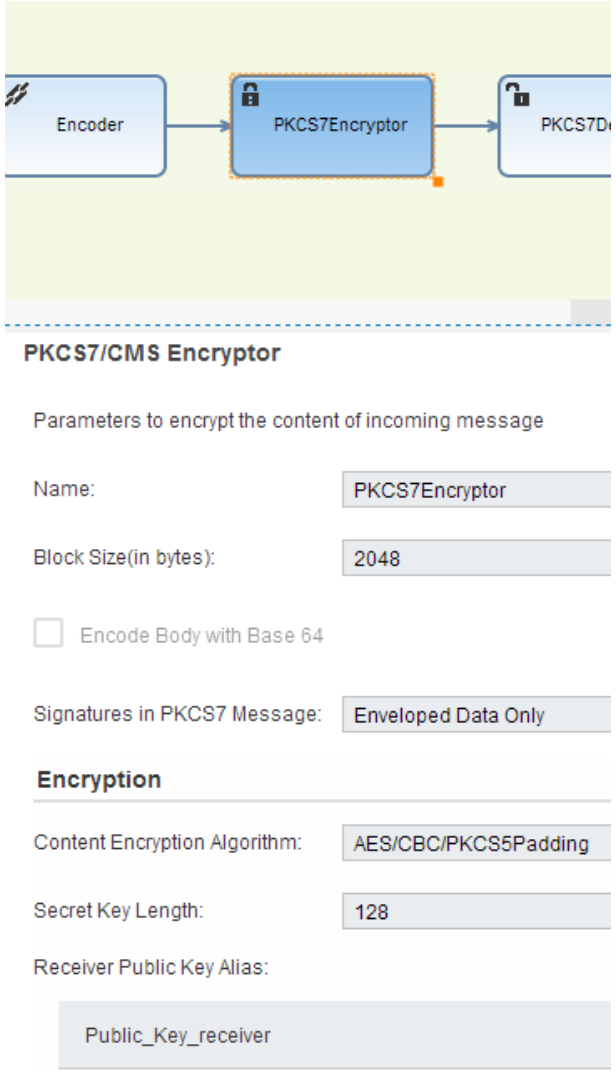
This Process Step is used to protect the message content from being altered while it is being sent to other participants on the cloud, by encrypting the content.

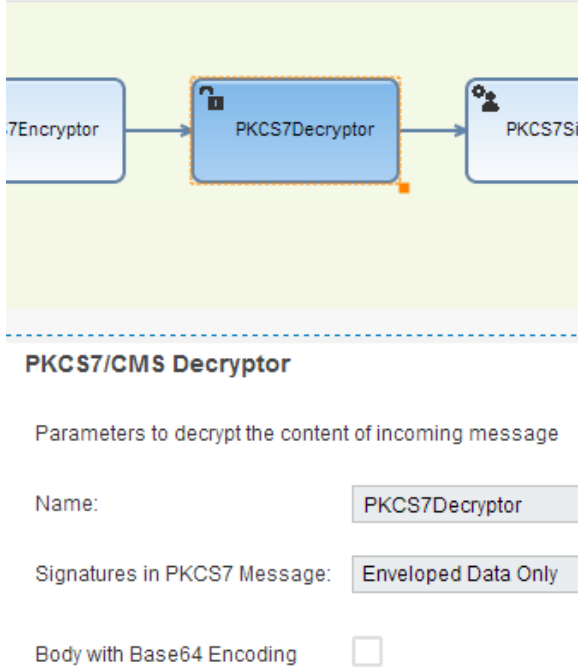
In addition to encrypting the message content, you can also sign the content to make your identity known to the participants and thus ensure the authenticity of the messages you are sending. This task guarantees your identity by signing the messages with one or more private keys using a signature algorithm

Content Decryptor

This Process Step is used to decrypt messages received from a participant on the cloud.

Note that the related keystore must contain the private key, otherwise decryption of the message content will not work. It is also possible to verify the signature of a SignedAndEnvelopedData object to ensure that the received signed message is authentic.



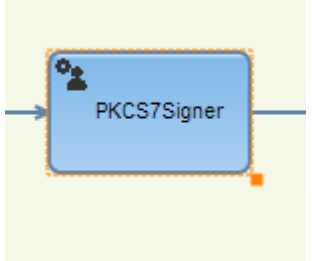


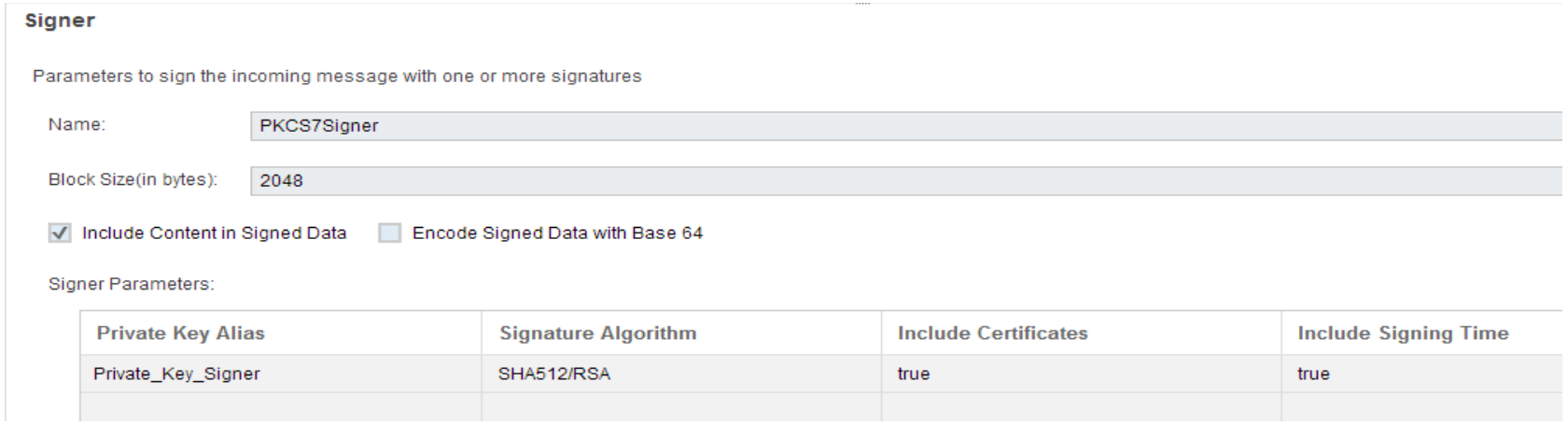
Security Elements: Message Signer

This Process Step is used to make identity of a Sender known to the participants and thus ensure the authenticity of the messages you are sending on the cloud. This task guarantees your identity by signing the messages with one or more private keys using a signature algorithm.

- **Working with PKCS#7/CMS Signer**

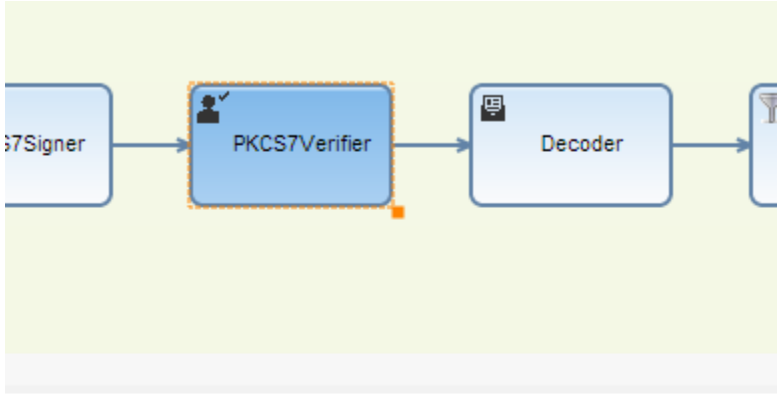
In the integration flow model, you configure the PKCS#7/CMS signer by providing one or more private key aliases. The signer uses the alias name to get the private keys of type DSA or RSA from the keystore. You also specify the signature algorithm for each key type, which is a combination of digest and encryption algorithms, for example, SHA512/RSA or SHA/DSA. The PKCS#7/CMS signer uses the algorithm to generate corresponding signatures. The data generated by the signer is known as the Signed Data.

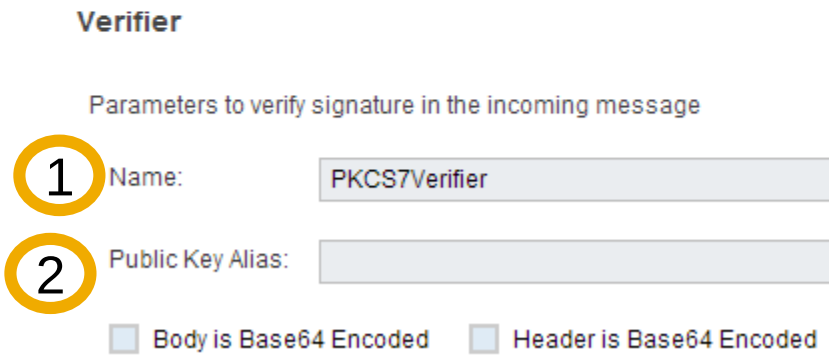




Security Elements: Signature Verifier

This Process step is used to ensure that the signed message received over the cloud is authentic.





1 In the integration flow model, you configure the Verifier by providing information about the public key alias, and whether the message header or body is Base64-encoded, depending on where the Signed Data is placed. For example, consider the following two cases:

- If the Signed Data contains the original content, then in the Verifier you provide the Signed Data in the message body
- If the Signed Data does not contain the original content, then in the Verifier you provide the Signed Data in the header SapCmsSignedData and the original content in the message body.

2 The Verifier uses the public key alias to obtain the public keys of type DSA or RSA that are used to decrypt the message digest. In this way the authenticity of the participant who signed the message is verified. If the verification is not successful, the verifier informs the user by raising an exception.

Under Public Key Alias you can enter one or multiple public key aliases for the Verifier.

Note: In general, an alias is a reference to an entry in a keystore. A keystore can contain multiple public keys. You can use a public key alias to refer to and select a specific public key from a keystore.

Thank you.

Contact information:

F name L name
Title
Address
Phone number

Partner logo

THE BEST RUN SAP®

Follow us

www.sap.com/contactsap

THE BEST RUN SAP®

SAP folgen auf

www.sap.com/germany/contactsap

THE BEST RUN SAP®