

SAP BTP Integration Suite

Contents

About:.....	.2
Scenario 1: Encoder or Decoder2
Scenario 2: “Combine” and “Enrich” options by using Odata Protocol (iFlow) (.....	.4
Scenario 3: CSV to XML conversion	11
Scenario 4: Filtering	13
Scenario 5: Filter and display the output for records that contain the “contact” field or a specific tag/field in the input data.	16
Scenario 6: How to Capture the Source and Target logs by using Groovy Script.	18
Scenario 7: Data Store and Fetch	23
Scenerio 8: Creating 2 iflows and Call 1 Iflow from anotehr IFlow	28
Scenario 9: XPath in Cloud Integration	32
Scenario 10: Converter (JSON to XML)	34

About:

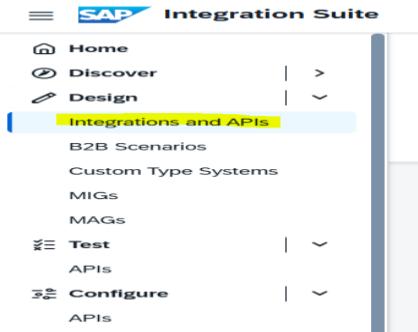
This document provides a step-by-step guide on creating iFlows using the SAP BTP Integration Suite. Due to confidentiality and security restrictions, details from the live project cannot be shared. Therefore, this Proof of Concept (POC) document demonstrates similar scenarios and examples for reference purposes.

Scenario 1: Encoder or Decoder

Description: Use the iFlow which will convert normal input to Encode format

Login integration suite

Select Design → Integration and APIs → Create Package



Save the Package after filling the mandatory details. Output will be like below:

A screenshot of the SAP Integration Suite interface showing a package named 'Encoder and Decoder'. The 'Overview' tab is selected. The package details include: Description: 'Encoder and Decoder'; Package ID: 'EncoderandDecoder'; Supported Platform: 'Cloud Integration'; Category: 'Integration'; Created: '08 Oct 2025'; Created by: 'koppusrao@gmail.com'; Last Modified: '08 Oct 2025'; Last Modified by: 'koppusrao@gmail.com'. The vendor is listed as 'Mode: Editable' and the version is '1.0.0'. The URL in the browser bar is 'Integrations and APIs / Encoder and Decoder /'.

Create the I flow by clicking Artifacts → Add → Integration Flow

A screenshot of the SAP Integration Suite interface showing the 'Encoder and Decoder' package artifacts. The 'Artifacts' tab is selected. A modal dialog is open at the top right with buttons for 'Save', 'Export', 'Cancel', and 'Delete Package'. Below the tabs, there is a search bar and a 'Filter Artifacts' dropdown. On the left, there is a list of artifact types: Value Mapping, Imported Archives, API, Message Mapping, Function Libraries, Integration Flow (which is highlighted with a yellow box), Integration Adapter, Script Collection, Service Interface, Data Type, and Message Type. The URL in the browser bar is 'Integrations and APIs / Encoder and Decoder /'.

Integrations and APIs / Encoder and Decoder /

Encoder and Decoder

Add Integration Flow

Create Upload

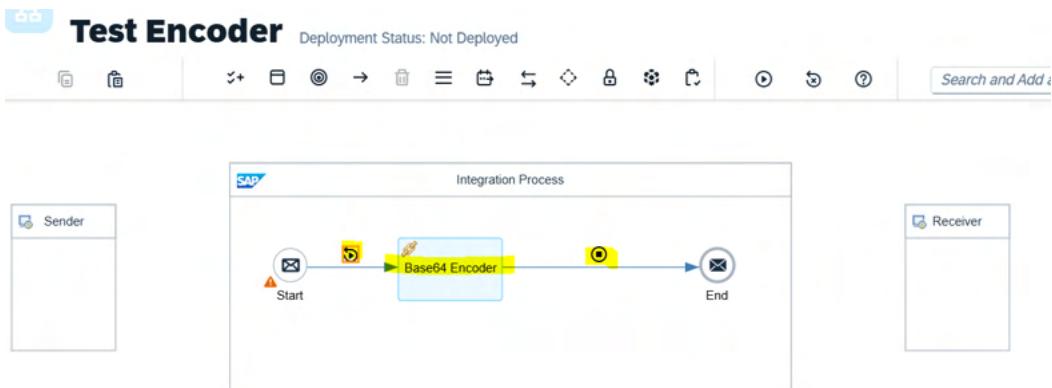
Name: * Test Encoder
ID: * Test_Encoder
Runtime Profile: Cloud Integration

Description: Test Encoder

Sender: Sender
Receiver: Receiver

Add **Add and Open in Editor** **Cancel**

Add from Plotter i.e., “Bae64 Encoder” and add “simulator start” and Simulator End” in flow



Add the information / input in “Simulator start body”

Add Simulation Input

Header

Name	Value
No data	

Properties

Name	Value
No data	

Body

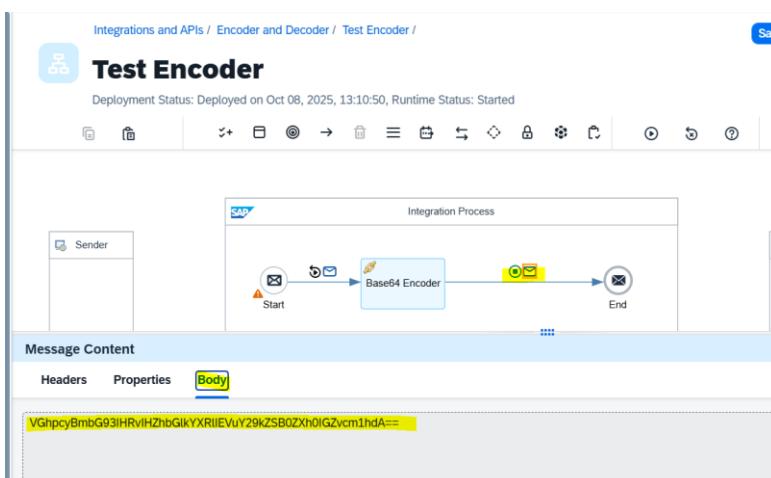
File:

Content: **This flow to validate Encode text format**

Upload from File System **OK** **Cancel**

Save and deploy. Click “Run Simulator” from plotter (from the top menu)

Output:



Note: Cross check the encode data from URL (<https://www.base64decode.org/>)

The screenshot shows a web-based encoder and decoder tool. At the top, there's a section titled "Decode from Base64 format" with instructions: "Simply enter your data then push the decode button." Below this is a text input field containing the base64-encoded string "VGhp...". To the right of the input field are several configuration options: "Source character set" set to "UTF-8", "Decode each line separately", "Live mode OFF" (which is checked), and a "DECODE" button. A yellow box highlights the "DECODE" button. Below the input area, another yellow box highlights the text "This flow to validate Encode text format".

Scenario 2: “Combine” and “Enrich” options by using Odata Protocol (iFlow) (SAP_BTP_CPI | OData With Content Enricher)

Real-Time Scenario:

The input data is received from multiple sources or two different files, such as “Product” and “Supplier.”

- Scenario 1: Merge both datasets sequentially — first the Product data, followed by the Supplier data — using the “Combine” option in the iFlow.
- Scenario 2: Merge the data within the same tag based on a common field (e.g., ID) using the “Enrich” option in the iFlow.

Data source:

<https://services.odata.org/V2/OData/Odata.svc/>

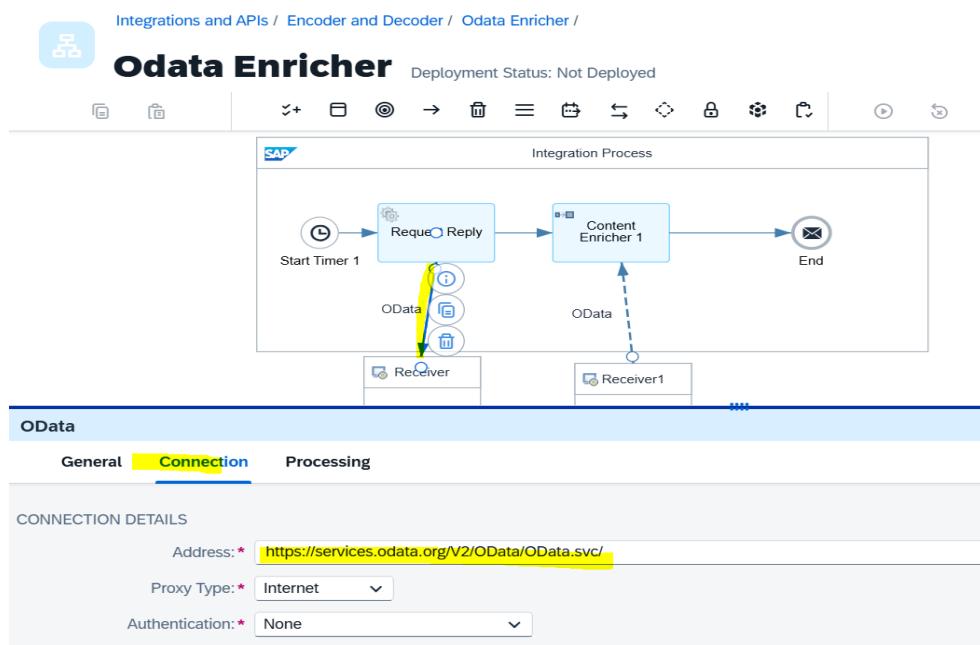
This screenshot shows a browser window with the URL "services.odata.org/V2/OData/Odata.svc/" in the address bar. The page content is an XML document representing an OData service. It includes sections for "workspace", "Products", "Categories", and "Suppliers". A yellow box highlights the URL in the address bar.

```
<service xmlns="http://www.w3.org/2007/app" xmlns:atom="http://www.w3.org/2005/Atom" xml:base="https://services.odata.org/V2/OData/Odata.svc/">
  <workspace>
    <atom:title>Default</atom:title>
    <collection href="Products">
      <atom:title>Products</atom:title>
    </collection>
    <collection href="Categories">
      <atom:title>Categories</atom:title>
    </collection>
    <collection href="Suppliers">
      <atom:title>Suppliers</atom:title>
    </collection>
  </workspace>
</service>
```

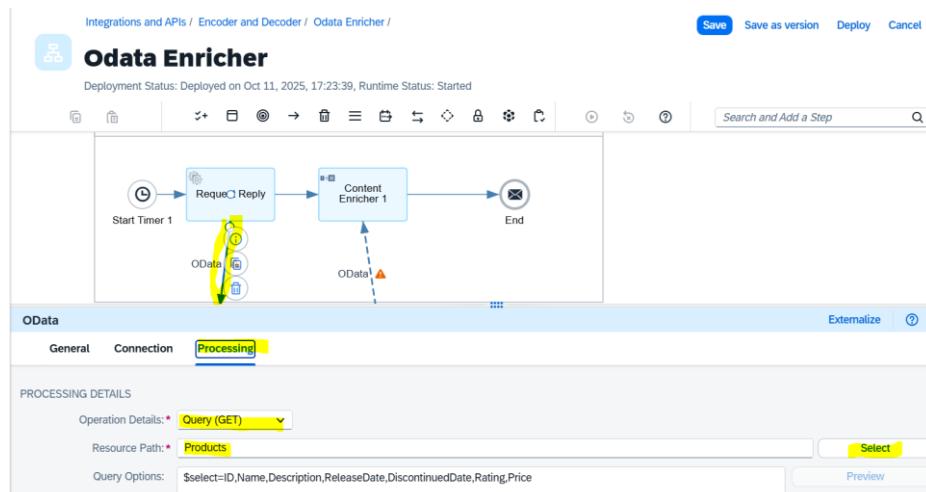
Go to “integration and API” → Add new integration flow like below.

This screenshot shows the "Encoder and Decoder" tool's "Add Integration Flow" dialog. The dialog has fields for "Name" (set to "Odata_Enricher"), "ID" (set to "Odata_Enricher"), "Runtime Profile" (set to "Cloud Integration"), "Description" (containing "Odata Enricher"), "Sender" (set to "Sender"), and "Receiver" (set to "Receiver"). A yellow box highlights the "Name" and "ID" fields. At the bottom of the dialog are buttons for "Add", "Add and Open in Editor", and "Cancel".

Remove “sender” and Receiver flows and add “timer” in “integration Process”



Click on “Processing” → “select”

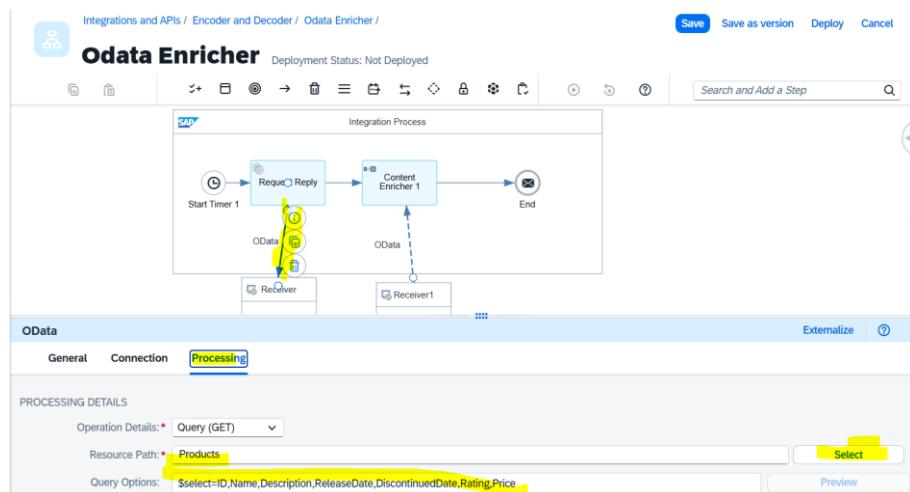


select “product” from drop down → “select all field”

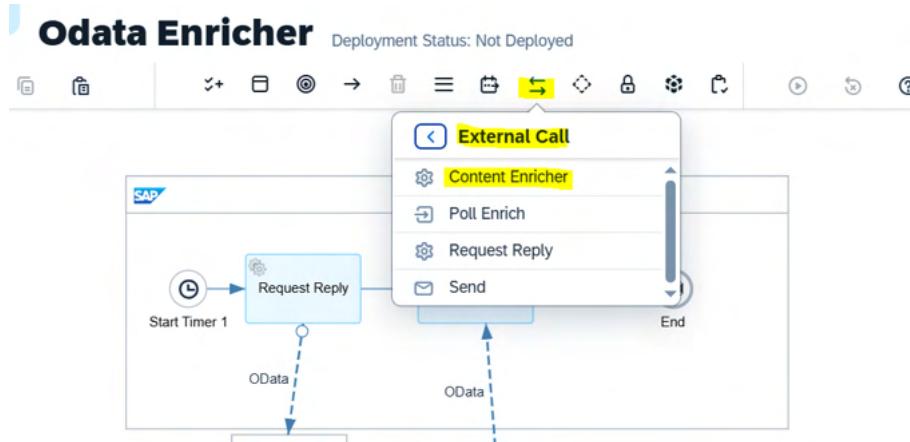
The screenshot shows the Model Operation interface. It consists of two main steps: "1. Connect to System" and "2. Select Entity & Define Operation". In Step 1, "Connect to System", the "Connection Source" is set to "Local EDMX File" and the "EDMX File" is "services_odata_org_V2_OData_OData.svc.edmx". In Step 2, "Select Entity & Define Operation", the "Operation" is "Query (GET)", the "Select Entity" is "Products", and the "Select All Fields" checkbox is checked. There is also a "Fields Filter" input field.

Click finish

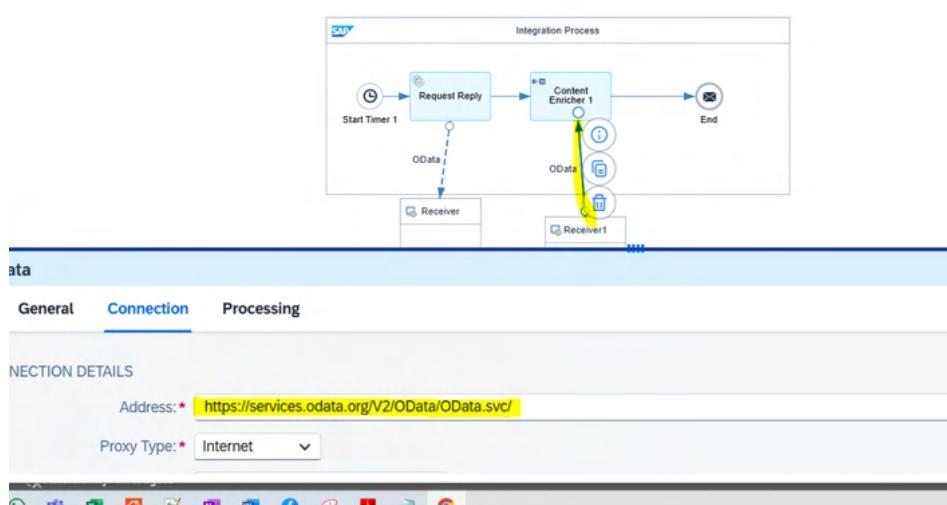
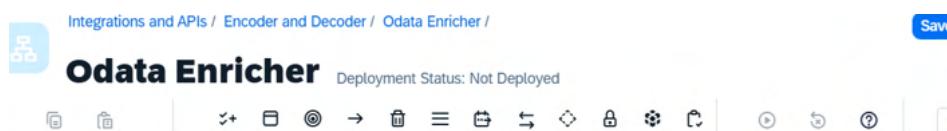
After “finish” iFlow as below:



Add the “External call → select “Content Enricher”



Add “receiver” and link from “enricher” to “receiver” with reverse arrow i.e, Arrow from receiver to “Content Enricher”, choose the Odata Protocol.



Select the “Supplier” option from Processing

Model Operation

1 Connect to System 2 Select Entity & Define Operation

EDMX FILE: services_odata_org_V2_OData_OData_svc.edmx Browse

2. Select Entity & Define Operation

Operation : * Query (GET) Sub Levels : 0

Select Entity : * Suppliers

Generate XML Schema Definition

Fields Filter

Fields

- ID
- Name
- Concurrency
- Address

Select “Content Enricher 1” and configure as below:

Integrations and APIs / Encoder and Decoder / Odata Enricher /

Odata Enricher

Deployment Status: Deployed on Oct 11, 2025, 17:09:25, Runtime Status: Started

Content Enricher

Processing

Aggregation Algorithm: Enrich

ORIGINAL MESSAGE

Path to Node: */Products/Product
Key Element: ID

LOOKUP MESSAGE

Path to Node: */Suppliers/Supplier
Key Element: ID

Select “reverse arrow” which is connected to “content modifier 1” user protocol as “Odata”

Integrations and APIs / Encoder and Decoder / Odata Enricher /

Odata Enricher Deployment Status: Not Deployed

OData

Processing

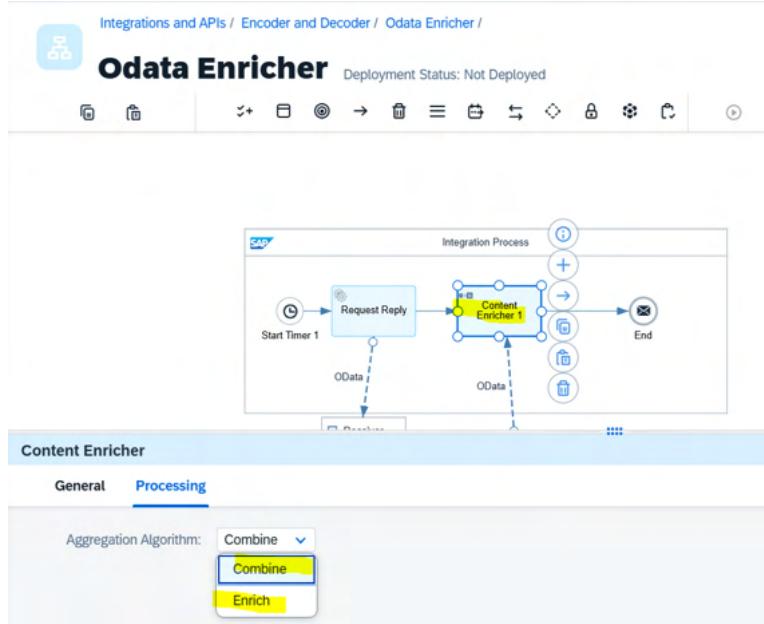
PROCESSING DETAILS

Operation Details: * Query (GET)
Resource Path: * Suppliers
Query Options: \$select=ID,Name,Concurrency,Address

Note: Content Enrich having 2 options

1. **Combined (it will combine “product” and “Supplier” data in the end/output)**
2. **Enrich (it will combine matched data of “product” and “Supplier data” based on the common filed i.e., “ID” will nest within XML and non-matched data at the end of output)**

- **Combined:**



Save → Deploy → trace log → deploy

Go to monitoring → download the log

The screenshot shows the SAP Monitoring interface under 'Overview / Monitor Message Processing / Message Processing Run'. It displays a list of 'Run Steps (6)' with their execution times: End (1 ms), Content Enricher 1 (less than 3 ms), Content Enricher 1 (566 ms), HCIOData (less than 3 ms), HCIOData (572 ms), and Start Timer 1 (1 sec 151 ms). The 'Payload' tab is selected, showing the XML payload of the message. The payload content is as follows:

```
<?xml version='1.0' encoding='UTF-8'?><multimap:Messages xmlns:multimap="http://sap.com/xi/XI/SplitAndMerge"><Products>
<Product>
<Rating>4</Rating>
<DiscontinuedDate/>
<ID>0</ID>
<ReleaseDate>1992-01-01T00:00:00.000</ReleaseDate>
</Product>
<Product>
<Rating>5</Rating>
<DiscontinuedDate/>
<ID>1</ID>
<ReleaseDate>1995-10-01T00:00:00.000</ReleaseDate>
</Product>
<Product>
<Price>20.9</Price>
<Rating>3</Rating>
<DiscontinuedDate/>
<ID>2</ID>
<ReleaseDate>2000-10-01T00:00:00.000</ReleaseDate>
</Product>
<Product>
<Price>19.9</Price>
<Rating>3</Rating>

```

First it will shows “Product” data at the end it will shows “Supplier” data.

Data from Downloaded Log file :

```

<?xml version='1.0' encoding='UTF-8'?>
<messaging:Messages xmlns:messaging="http://sap.com/xi/XI/SplitAndMerge">
<messaging:Message1>
<Products>
    <Product>
        <Price>2.5</Price>
        <Rating>4</Rating>
        <DiscontinuedDate/>
        <ID>0</ID>
        <ReleaseDate>1992-01-01T00:00:00.000</ReleaseDate>
    </Product>
    <Product>
    <Product>
    <Product>
    <Product>
        <Price>22.8</Price>
        <Rating>3</Rating>
        <DiscontinuedDate/>
        <ID>5</ID>
        <ReleaseDate>2006-08-04T00:00:00.000</ReleaseDate>
    </Product>
    <Product>
    <Product>
    <Product>
</Products>
</messaging:Message1>
<messaging:Message2>
<Suppliers>
    <Supplier>
        <Address>
            <State>WA</State>
            <ZipCode>98074</ZipCode>
            <Street>NE 228th</Street>
            <Country>USA</Country>
            <City>Sammamish</City>
        </Address>
        <Concurrency>0</Concurrency>
        <ID>0</ID>
        <Name>Exotic Liquids</Name>
    </Supplier>
</Suppliers>

```

- Output from “Enriched”:

Changes to be made below:

Original Message → means in “request reply” we used “Products”, hence that tag to be given

Lookup message → In enrich configuration we used “Suppliers”, hence that tag to be given

Integrations and APIs / Encoder and Decoder / Odata Enricher / Save

Odata Enricher

Deployment Status: Deployed on Oct 11, 2025, 17:09:25, Runtime Status: Started

Content Enricher

Processing

Aggregation Algorithm: **Enrich**

ORIGINAL MESSAGE

Path to Node: * /Products/Product

Key Element: * ID

LOOKUP MESSAGE

Path to Node: * /Suppliers/Supplier

Key Element: * ID

Note: Product and Supplier information in one block where ID is matching Ex: ID=0

Overview / Monitor Message Processing / Message Processing Run

Run Steps (6)		Integration Flow Model	Log Content	Message Content
End	1 / 1	Message before Step		Down
		Header	Exchange Properties	Payload
				Download Payload
Content Enricher 1	Segment 1	<pre><products> <product> <Price>2.5</Price> <Rating>4</Rating> <DiscontinuedDate/> <ID>0</ID> <Supplier> <Address> <State>WA</State> <ZipCode>98074</ZipCode> <Street>NE 228th</Street> <Country>USA</Country> <City>Sammamish</City> </Address> <Concurrency>0</Concurrency> <ID>0</ID> <Name>Exotic Liquids</Name> </Supplier> <ReleaseDate>1992-01-01T00:00:00.000</ReleaseDate> </product> <product> <Price>3.5</Price> <Rating>3</Rating> <DiscontinuedDate/> <ID>1</ID> <Supplier> <Address> <State>WA</State> <ZipCode>98052</ZipCode> <Street>NE 40th</Street></pre>		
Content Enricher 1	Segment 1			
HCIOData	Segment 1			
HCIOData	Segment 1			
Start Timer 1	Segment 1			

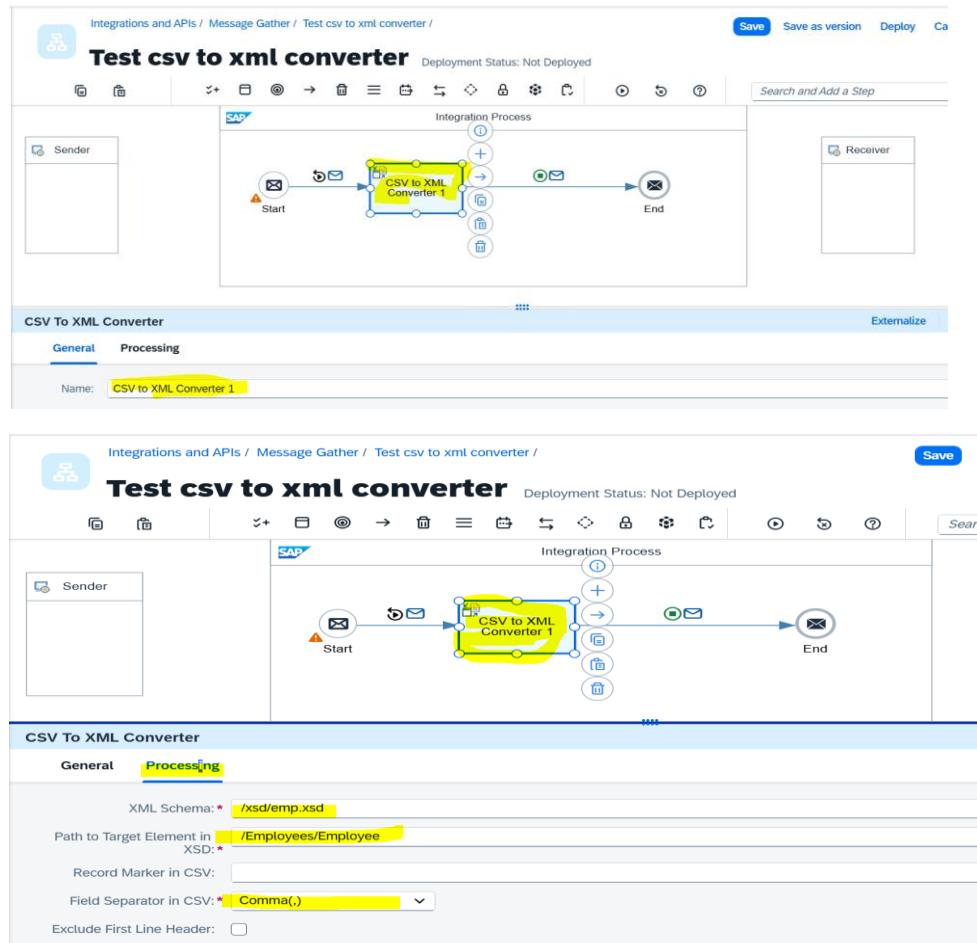
```
<products>
<product>
<Price>2.5</Price>
<Rating>4</Rating>
<DiscontinuedDate/>
<ID>0</ID>
<Supplier>
<Address>
<State>WA</State>
<ZipCode>98074</ZipCode>
<Street>NE 228th</Street>
<Country>USA</Country>
<City>Sammamish</City>
</Address>
<Concurrency>0</Concurrency>
<ID>0</ID>
<Name>Exotic Liquids</Name>
</Supplier>
<ReleaseDate>1992-01-01T00:00:00.000</ReleaseDate>
</product>
<product>
<Price>3.5</Price>
<Rating>3</Rating>
<DiscontinuedDate/>
<ID>1</ID>
<Supplier>
<Address>
<State>WA</State>
<ZipCode>98052</ZipCode>
<Street>NE 40th</Street>
```

Scenario 3: CSV to XML conversion

Real-Time Scenario:

As per the requirement, when an input file is received in **CSV** format, it needs to be **converted into XML** format before being sent to the destination system.

- After logging in to the **SAP BTP Integration Suite**, create a **new package**, and then create a **new artifact (integration flow)** within it
- From transform select the “CSV to XML converter” or search from search bar and place in the iflow as below and configure as below:



Sample xsd file used for this test :

Note: this xsd file based on the csv data information. Use public site to generate the xsd file based on the CSV data.

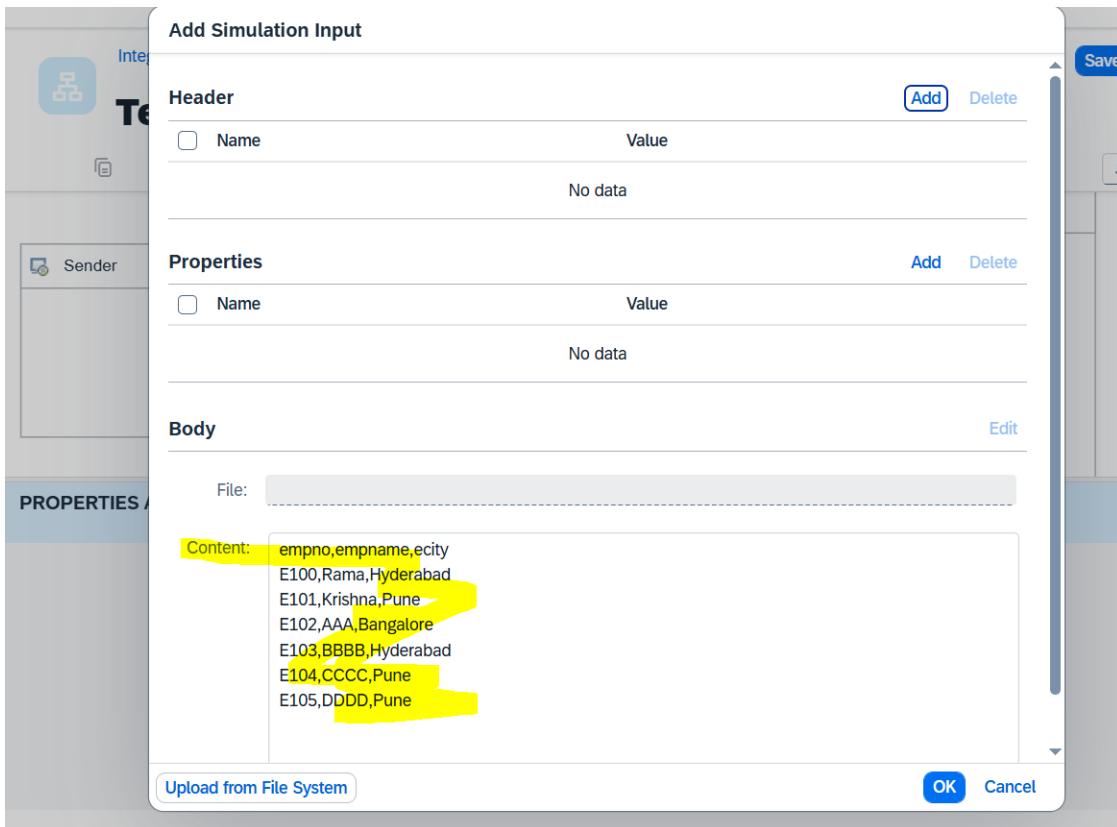


emp.xsd

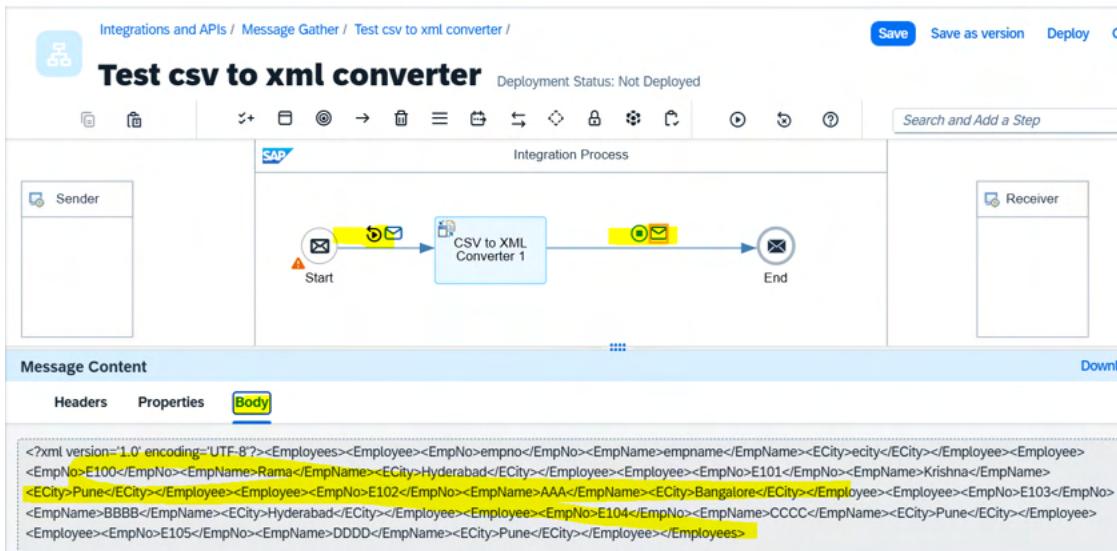
Sample data in CSV file:

```
empno,empname,ecity
E100,Rama,Hyderabad
E101,Krishan,Pun
E102,AAA,Bangalore
E1-3,BBBB,Hyderabad
E104,CCCC,Pune
E105,DDDD,Pune
```

Add “Simulation start” and Simulation End” and add the CSV data in the “simulation start” body as below:



Execute the Simulation and see the output in the “simulation end” tool as below (which is converting csv data to xml format).



Scenario 4: Filtering

This will support and filter based on the filter condition and provide the output.

In POC covered Boolean, Node, Nodelist options

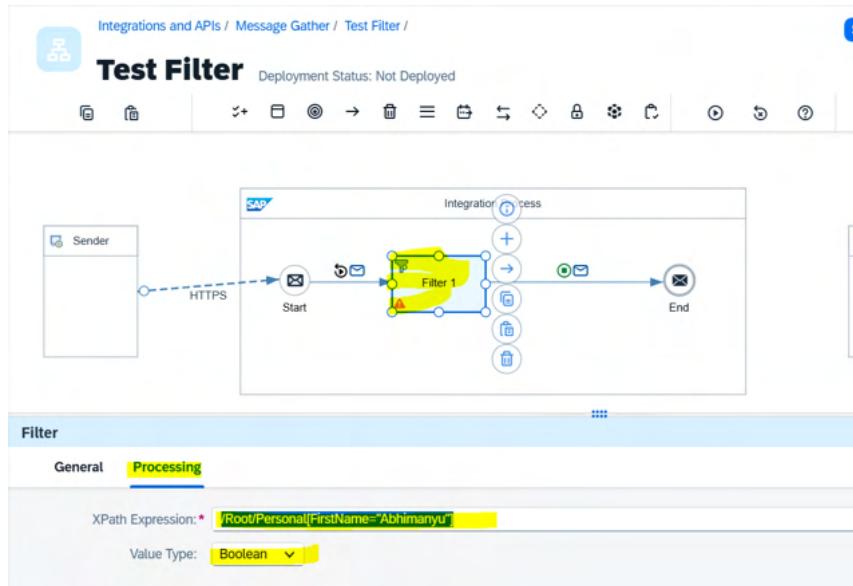
The below sample data used for this POC:



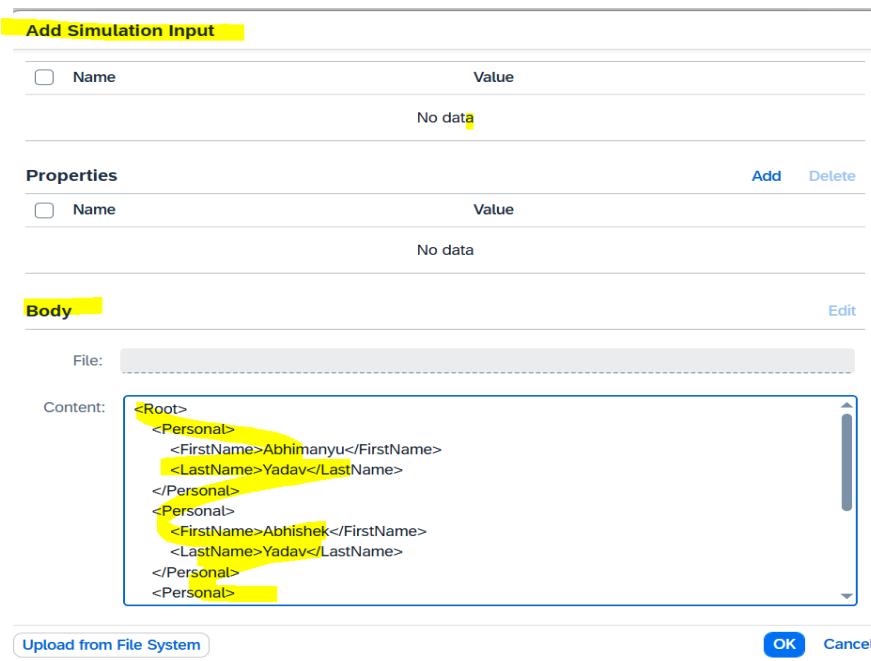
- **Boolean:** based on the condition if match it will give “true” or “false”

In the example we configured to search the “/Root/Personal[FirstName=“Abhimanyu”]”

If this match, “simulation output body” will give the result “true” (if found) or “false” (if not found).



Input provided in the “simulation start body”



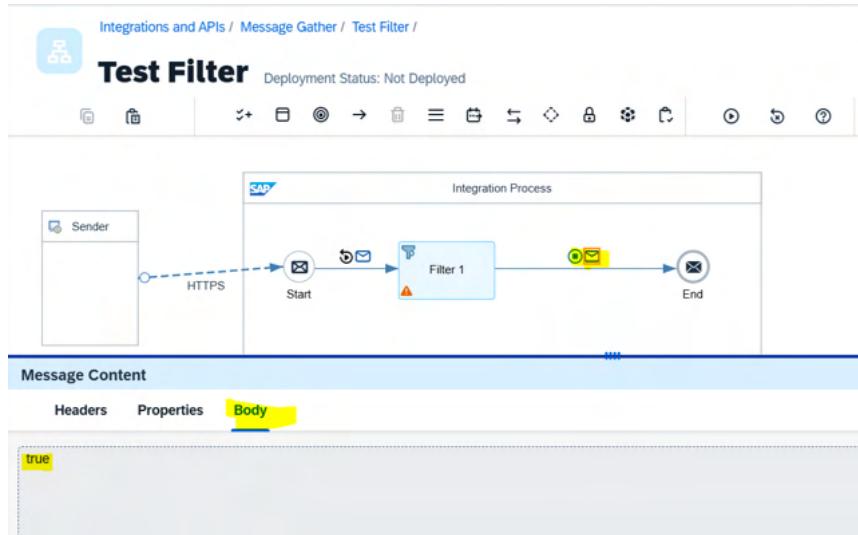
The screenshot shows the 'Add Simulation Input' dialog. It has three tabs: 'Properties' (selected), 'Body' (highlighted in yellow), and 'File:' (disabled). The 'Properties' tab shows a table with one row where 'Name' is empty and 'Value' is 'No data'. The 'Body' tab shows an XML content area with the following code:

```
<Root>
<Personal>
<FirstName>Abhimanyu</FirstName>
<LastName>Yadav</LastName>
</Personal>
<Personal>
<FirstName>Abhishek</FirstName>
<LastName>Yadav</LastName>
</Personal>
<Personal>
```

At the bottom of the dialog are 'Upload from File System' and 'OK' / 'Cancel' buttons.

Note: Since the input condition matches (i.e., **firstname = "Abhimanyu"**), the output is displayed as **"true"**, because the **Value Type** option was set to **Boolean**.

Seen the below output:



The below Filter Output is based on **Value type** “Node” or Nodelist
(Node vs Nodelist)

- **Node** option from *Value Type*:

When the **Value Type** is set to **Node**, the output shows the details of the **first record** that matches the filter condition. This means only the **first matching entry** from the input will be displayed as the **single output**.

This screenshot shows the 'Filter' configuration dialog in SAP Integration Studio. The top bar has a 'Filter' title. Below it, there are two tabs: 'General' and 'Processing' (which is highlighted). Under 'Processing', there is an 'XPath Expression' field containing the value '/Root/Personal/FirstName'. Below that is a 'Value Type:' dropdown menu with 'Node' selected. There are also 'Add' and 'Delete' buttons for managing filters.

Note: in the XML file / input file having the filed “FirstName” and found, then that value will return in output file.

The below is input from “simulation start body” having multile “Firname” values

Input in the simulation body:

This screenshot shows the 'Add Simulation Input' dialog. It has three main sections: 'Header', 'Properties', and 'Body'. In the 'Header' section, there is a table with columns 'Name' and 'Value', showing 'No data'. In the 'Properties' section, there is a similar table with 'Name' and 'Value' columns, also showing 'No data'. In the 'Body' section, there is a 'File:' dropdown and a 'Content:' text area. The 'Content:' area contains the following XML code:

```

<Root>
  <Personal>
    <FirstName>Abhimanyu</FirstName>
    <LastName>Yadav</LastName>
  </Personal>
  <Personal>
    <FirstName>Abhishek</FirstName>
    <LastName>Yadav</LastName>
  </Personal>

```

At the bottom of the dialog, there are 'OK' and 'Cancel' buttons.

Ouput:

Eventhough "Firstanme" having mulitiple times in the input (approx.. 3 times) output will consider First Occurance and display that value:

Integrations and APIs / Message Gather / Test Filter /

Test Filter

Deployment Status: Not Deployed

Integration Process

Message Content

Headers Properties Body

<FirstName>Abhimanyu</FirstName>

- **Nodelist** option from *Value Type* :

Outwill display all matched list / details (Single/multiple output)

Configuration:

Integrations and APIs / Message Gather / Test Filter /

Test Filter

Deployment Status: Not Deployed

Integration Process

Filter

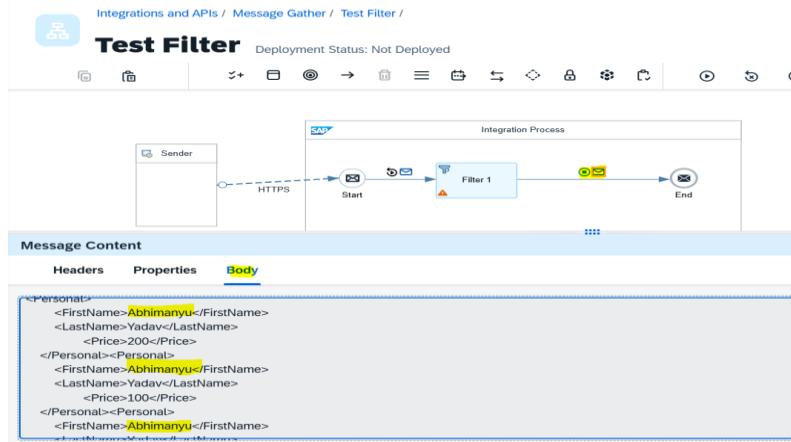
General Processing

XPath Expression: * /Root/Personal[FirstName="Abhimanyu"]

Value Type: Nodelist

Output → will consider and display all data matched with filed (FirstName)—"
/Root/Personal[FirstName="Abhimanyu"]"

Matched 3 members data and showing the same in the output.

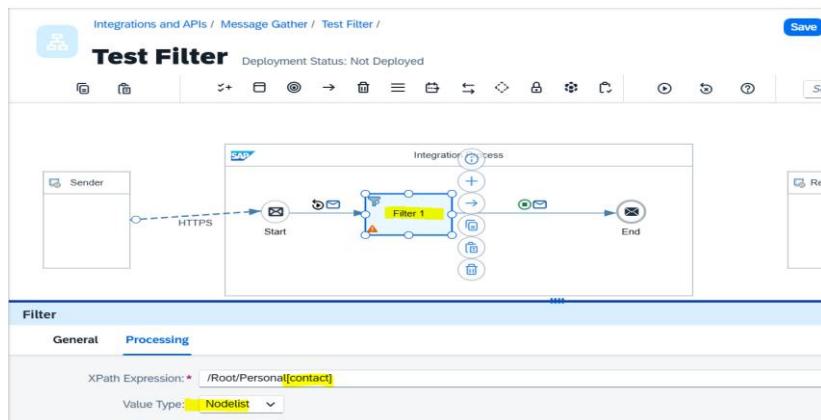


Scenario 5: Filter and display the output for records that contain the “contact” field or a specific tag/field in the input data.

If the input file contains the “contact” tag (in the XML), only the records with this tag will appear in the output, and all other records will be ignored.

Example: Out of 10 records, if only 3 have the “contact” tag, only those 3 records will be displayed in the output, and the remaining 7 will be ignored.

Configuration:



Input data having the below information:

```

<Root>
  <Personal>
    <FirstName>Abhimanyu</FirstName>
    <LastName>Yadav</LastName>
    <Price>200</Price>
  </Personal>
  <Personal>
    <FirstName>Abhishek</FirstName>
    <LastName>Yadav</LastName>
    <contact>
      <phone>2244555 </phone>
    </contact>
  </Personal>
  <Personal>
    <FirstName>Shekhar</FirstName>
    <LastName>Yadav</LastName>
    <Price>300</Price>
    <contact>
      <phone>99933388</phone>
    </contact>
  </Personal>
  <Personal>
    <FirstName>Abhimanyu</FirstName>
    <LastName>Yadav</LastName>
    <Price>100</Price>
  </Personal>
  <Personal>
    <FirstName>Shekhar</FirstName>
    <LastName>Yadav</LastName>
    <Price>400</Price>
    <contact>
      <phone>99933388</phone>
    </contact>
  </Personal>

```

Output:

Integrations and APIs / Message Gather / Test Filter /

Test Filter

Deployment Status: Not Deployed

The screenshot shows a basic integration process. It starts with a 'Sender' icon connected via 'HTTPS' to a 'Start' event. This leads to a 'Filter 1' node, which has a warning icon. From 'Filter 1', the flow goes to an 'End' event.

Message Content

Body

```
<FirstName>Abhishek</FirstName>
<LastName>Yadav</LastName>
<contact>
<phone>2244555 </phone>
</contact>
</Personal><Personal>
<FirstName>Shekhar</FirstName>
<LastName>Yadav</LastName>
<contact>
<phone>99993388</phone>
```

Double filter: Scenario → Filter condition is Contact should be there along with Last Name as "Yadav"

Integrations and APIs / Message Gather / Test Filter /

Test Filter

Deployment Status: Not Deployed

The screenshot shows a more complex filter configuration. After the 'Start' event, the flow splits into two parallel paths. Both paths converge at a 'Filter 1' node, which has a warning icon. From 'Filter 1', the flow continues to an 'End' event.

Filter

Processing

XPath Expression: * /Root/Personal[contact and LastName="Yadav"]

Value Type: Nodelist

Output: Output is only Last name is "**Yadav**" along with that, that person having "**contact**".

Integrations and APIs / Message Gather / Test Filter /

Test Filter

Deployment Status: Not Deployed

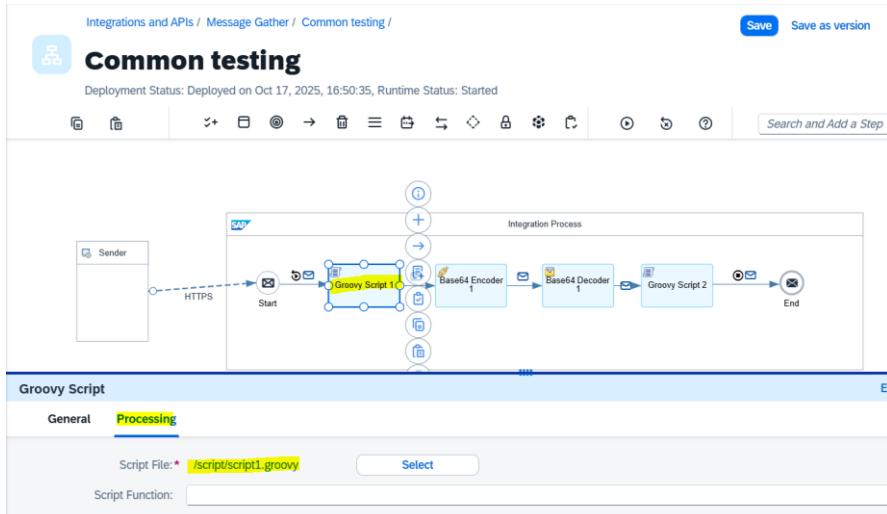
The screenshot shows a single filter configuration. After the 'Start' event, the flow goes directly to a 'Filter 1' node, which has a warning icon. From 'Filter 1', the flow goes to an 'End' event.

Message Content

Body

```
<FirstName>Abhishek</FirstName>
<LastName>Yadav</LastName>
<contact>
<phone>2244555 </phone>
</contact>
</Personal><Personal>
<FirstName>Shekhar</FirstName>
<LastName>Yadav</LastName>
<contact>
<phone>99993388</phone>
```

Scenario 6: How to Capture the Source and Target logs by using Groovy Script.



See the Groovy Script in side “[/script/script1.groovy](#)”

After adding / writing the Groovy script click on “Apply” to validate and fix the script.

Groovy Script at Source:

Integrations and APIs / Message Gather / Common testing / script1.groovy

script1.groovy

Apply

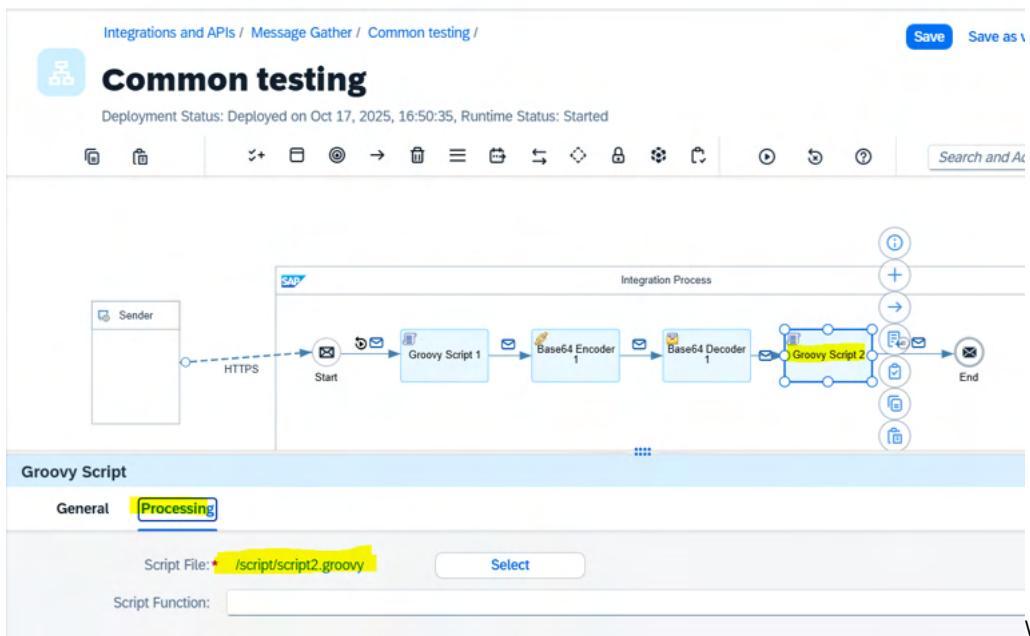
```
1 import com.sap.gateway.ip.core.customdev.util.Message;
2
3 def processData(Message message) {
4     def body = message.getBody(java.lang.String) as String;
5     def messageLog = messageLogFactory.getMessageLog(message);
6
7     if (messageLog != null) {
8         messageLog.setStringProperty("Logging#1", "Printing Payload As Attachment");
9         messageLog.addAttachmentAsString("Source Payload", body, "text/plain");
10    }
11
12    return message;
13 }
14
```

Problems

Upgrade Readiness Checks General Script Checks

Type	Description	Actions

Groovy Script at Target:



Groovy script at Target:

```

1 import com.sap.gateway.ip.core.customdev.util.Message;
2
3 def Message processData(Message message) {
4     def body = message.getBody(java.lang.String) as String;
5     def messageLog = messageLogFactory.getMessageLog(message);
6
7     if (messageLog != null) {
8         messageLog.setStringProperty("Logging#1", "Printing Payload As Attachment");
9         messageLog.addAttachmentAsString("Target Payload:", body, "text/plain");
10    }
11
12    return message;
13}
14

```

Problems

- Upgrade Readiness Checks
- General Script Checks

Type	Description
No data	

Save and Deploy the Iflow and check the log in “monitoring”

Messages (2)	
Artifact Name	Status
Common testing	Completed Oct 17, 2025, 16:53:32
Common testing	Completed Oct 17, 2025, 16:52:44

Common testing
Last Updated at: Oct 17, 2025, 16:53:32

- Status
- Properties
- Logs
- Attachments
- Artifact Details

Status

Message processing completed successfully.

Processing Time: 105 ms

Properties

Message ID: AGjyJzSlgfAGtOjuYm6cJeFdUVd
Correlation ID: AGjyJzRxDmOrTVLNyxp8vfpG5Jij

Messages (2)	
Artifact Name	Status
Common testing	Completed Oct 17, 2025, 16:53:32
Common testing	Completed Oct 17, 2025, 16:52:44

Common testing
Last Updated at: Oct 17, 2025, 16:53:32

- Status
- Properties
- Logs
- Attachments
- Artifact Details

Attachments

Entries (2)				
Name	Type	Modified At	Size	Action
Source Payload	text/plain	Oct 17, 2025, ...	1 KB	Download
Target Payload	text/plain	Oct 17, 2025, ...	1 KB	Download

Source Payload / Log:

Overview / Monitor Message Processing / Message Processing Log Attachments

Common testing

Last Updated at: Oct 17, 2025, 16:53:32 Log Level: Info
Status: Completed Processing Time: 105 ms

Log **Attachments**

Source Payload **Target Payload:**

This is my first HTTPS iflow of Protocol, testing with Groovy script for login !!!

Overview / Monitor Message Processing / Message Processing Log Attachments

Common testing

Last Updated at: Oct 17, 2025, 16:53:32 Log Level: Info
Status: Completed Processing Time: 105 ms

Log **Attachments**

Source Payload **Target Payload:**

This is my first HTTPS iflow of Protocol, testing with Groovy script for login !!!

Logs for each step through simulation mode:

Integrations and APIs / Message Gather / Common testing / **Edit** **Configure**

Common testing

Deployment Status: Deployed on Oct 17, 2025, 16:50:35, Runtime Status: Started

Integration Process

```
graph LR; Sender[Sender] -- "HTTPS" --> Start((Start)); Start --> GS1[Groovy Script 1]; GS1 --> BE1[Base64 Encoder 1]; BE1 --> BD1[Base64 Decoder 1]; BD1 --> GS2[Groovy Script 2]; GS2 --> End((End))
```

Message Content

Headers Properties **Body**

This is for Groovy script testing to capture the log for both source and target (Encrypting and Decrypting)

Integrations and APIs / Message Gather / Common testing /

Common testing Deployment Status: Deployed on Oct 17, 2025, 16:50:35, Runtime Status: Started

Edit Configure

```

graph LR
    Sender[Sender] -- "HTTPS" --> Start((Start))
    Start --> GS1[Groovy Script 1]
    GS1 --> BE1[Base64 Encoder 1]
    BE1 --> BD1[Base64 Decoder 1]
    BD1 --> GS2[Groovy Script 2]
    GS2 --> End((End))
  
```

Message Content

Headers Properties Body

This is for Groovy script testing to capture the log for both source and target (Encrypting and Decrypting)

Integrations and APIs / Message Gather / Common testing /

Common testing Deployment Status: Deployed on Oct 17, 2025, 16:50:35, Runtime Status: Started

Edit Configure

```

graph LR
    Sender[Sender] -- "HTTPS" --> Start((Start))
    Start --> GS1[Groovy Script 1]
    GS1 --> BE1[Base64 Encoder 1]
    BE1 --> BD1[Base64 Decoder 1]
    BD1 --> GS2[Groovy Script 2]
    GS2 --> End((End))
  
```

Message Content

Headers Properties Body

VGhpccBpcyBmb3lgR3Jvb3Z5iHnjcmIwdCB0ZXN0aW5nIHViGnhcHR1cmUgdGhlGxvzbMbg3ig
Ym90aCBzb3Vyy2UgYW5kIHRhcmdldCAoRW5jcnIwdGluZyBhbmoRGVjcnIwdGluZyk=

Integrations and APIs / Message Gather / Common testing /

Common testing Deployment Status: Deployed on Oct 17, 2025, 16:50:35, Runtime Status: Started

Edit Configure Deploy

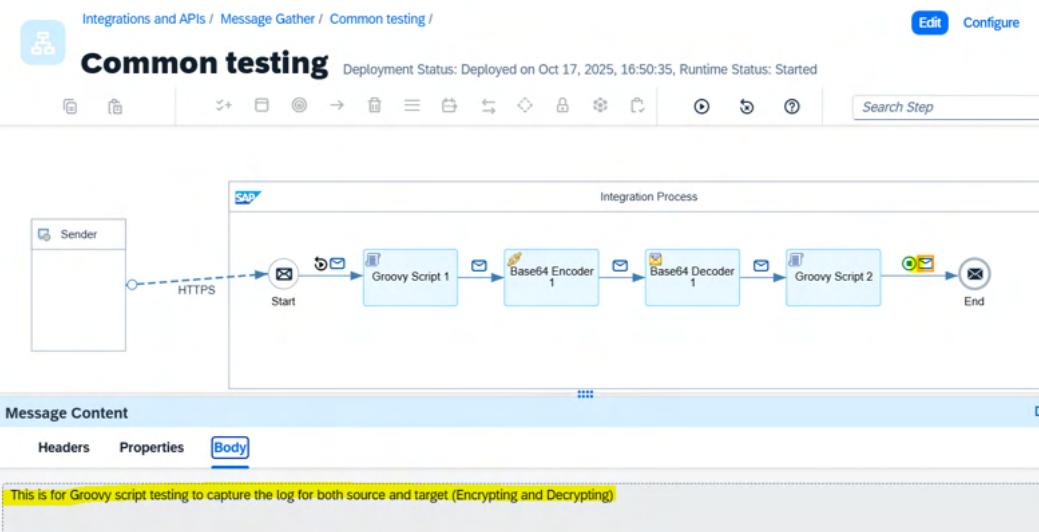
```

graph LR
    Sender[Sender] -- "HTTPS" --> Start((Start))
    Start --> GS1[Groovy Script 1]
    GS1 --> BE1[Base64 Encoder 1]
    BE1 --> BD1[Base64 Decoder 1]
    BD1 --> GS2[Groovy Script 2]
    GS2 --> End((End))
  
```

Message Content

Headers Properties Body

This is for Groovy script testing to capture the log for both source and target (Encrypting and Decrypting)



Logs from the iflow:

Overview / Monitor Message Processing

Time: Past Hour Status: All Type: All Package: All Artifacts: All ID: Message, Correlation or Application

Nov 07, 2025, 13:06:39 - Nov 07, 2025, 14:06:39

Messages (1)	
Artifact Name	Status
Common testing	Completed Nov 07, 2025, 14:04:17
	352 ms

Common testing
Last Updated at: Nov 07, 2025, 14:04:17

Status Properties Logs **Attachments** Artifact Details

Attachments

Entries (2)					
Name	Type	Modified At	Size	Action	
Source Payload	text/plain	Nov 07, 2025, ...	1 KB	Download	
Target Payload	text/plain	Nov 07, 2025, ...	1 KB	Download	

Artifact Details

Source payload log:

Integration Suite

Home Discover Design Test Configure Monitor Analyze Engage Inspect Monetize Settings

Overview / Monitor Message Processing / Message Processing Log Attachments

Common testing

Last Updated at: Nov 07, 2025, 14:04:17 Log Level: Trace
Status: Completed Processing Time: 352 ms

Log **Attachments**

Source Payload Target Payload:

This is for Groovy script testing to capture the log for both source and target (Encrypting and Decrypting)

Target payload log:

Last Updated at: Nov 07, 2025, 14:04:17 Log Level: Trace
 Status: Completed Processing Time: 352 ms

Common testing

Log Attachments

Source Payload Target Payload

This is for Groovy script testing to capture the log for both source and target (Encrypting and Decrypting)

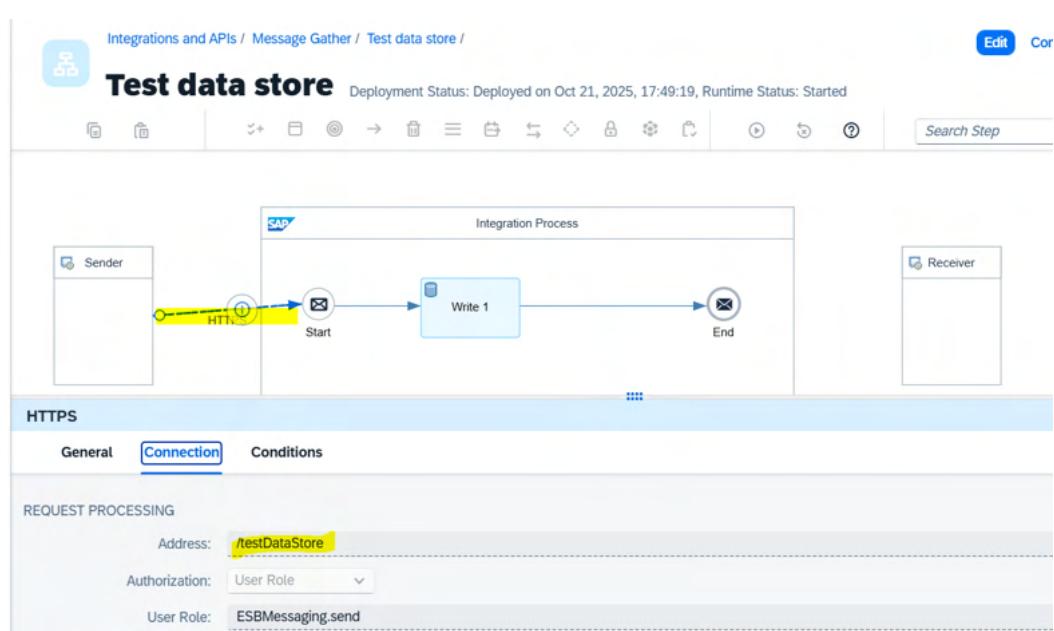
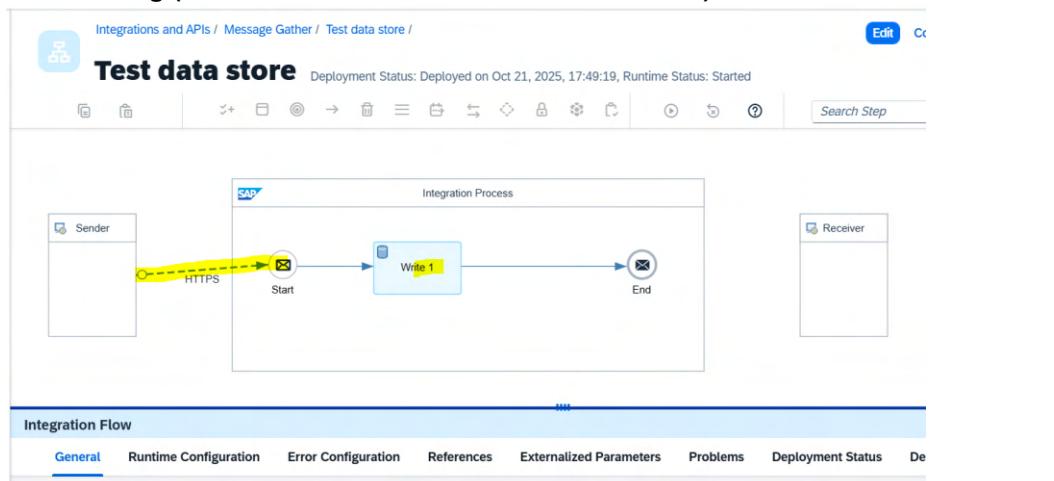
=====

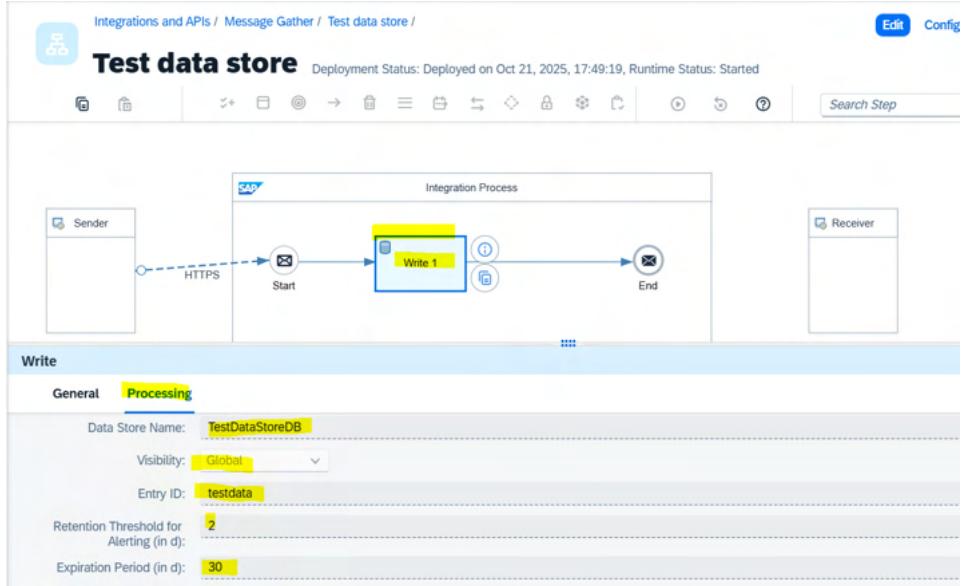
Scenario 7: Data Store and Fetch

Realtime scenario:

We can use an iFlow to **store data into a database** (with the DB and table created during the iFlow) and then **retrieve that data from the same database and table** using another iFlow.

- **Data Storing (1st iFlow to store the data in DB and table):**





Execute the input data from POSTMAN (use POST method option to store in the DB and table)

Method: **POST** URL: <https://1284bac5trial.it-cptrial03-rt.cfapps.ap21.hana.ondemand.com/http/testDataStore>

Body **XML**

```

1 <Root>
2   <Personal>
3     <FirstName>Abhimanyu</FirstName>
4     <LastName>Yadav</LastName>
5     <Price>200</Price>
6   </Personal>
7   <Personal>
8     <FirstName>Abhishek</FirstName>
9     <LastName>Yadav</LastName>
10    <contact>
11      <phone>2244555 </phone>
12    </contact>
13  </Personal>
14  <Personal>
15    <FirstName>Shekhar</FirstName>
16    <LastName>Yadav</LastName>
17    <Price>300</Price>
18    <contact>
19      <phone>99933388</phone>

```

Body Cookies Headers Test Results

200 OK 526 ms 1.84 KB

Data stored in the DB (check from the iFlow messaging) :

Integrations and APIs

Discover | >

Design | ▾

Integrations and APIs

B2B Scenarios

Custom Type Systems

MIGs

MAGs

Test | >

Configure | >

Monitor | ▾

Access Logs

System Log Files

Overview

Connectivity Tests

Manage Stores

Data Stores 1

Variables 0

Access Logs

System Log Files

Overview / Manage Data Stores

Data Stores (1)	
TestDataSourceDB	1 Global

TestDataSourceDB

Entries (1)

ID	Status	Due At	Created At
testdata	Waiting	Oct 23, 2025, 17:49:28	Oct 21, 2025, 17:49:28

Message ID: AGj3elBMLaOyaa2wPN62VCC7XNK9

Home | Discover | Design | Integrations and APIs | Test | Configure | Monitor | Analyze | Engage | Inspect | Monetize | Semantics

Overview / Manage Integration Content

Integration Content (8)	
Name	Status
Test data fetch	Started
Integration Flow	
Test data store	Started
Integration Flow	
Test Aggregator	Started
Integration Flow	
Common testing	Started
Integration Flow	
Test Content Modifier	Started
Integration Flow	
Odata Enricher	Started
Integration Flow	
FirstFlowTest	Started
Integration Flow	
Test Encoder	Started

Test data store

Deployed On: Oct 21, 2025, 17:49:19 ID: Test_data_store Package: Message Gather Deployed By: koppusrao@gmail.com Version: 1.0.0

[Endpoints](#) [Status Details](#) [Artifact Details](#) [Log Configuration](#)

Endpoints

<https://128.4ba1trial.1c+cptrial03-rt.cfapps.ap21.hana.ondemand.com/http/testDataSource>

Status Details

The Integration Flow is deployed successfully.

Artifact Details

Monitor Message Processing View deployed Artifact

Overview / Monitor Message Processing

Time: Past Hour Status: All Type: All Package: All Artifacts: Test data ... or

Oct 21, 2025, 17:12:27 - Oct 21, 2025, 18:12:27

Messages (4)

Artifact Name	Status
Test data store	Completed
Oct 21, 2025, 17:49:28	34 ms
Test data store	Failed
Oct 21, 2025, 17:47:26	32 ms
Test data store	Failed
Oct 21, 2025, 17:47:12	52 ms
Test data store	Completed
Oct 21, 2025, 17:44:42	129 ms

Test data store

Last Updated at: Oct 21, 2025, 17:49:28

[Status](#) [Properties](#) [Logs](#) [Artifact Details](#)

Logs

Trace data is removed after the configured retention time

Log Level: Trace Instance ID: 0

Artifact Details

Manage Integration Content View deployed Artifact Navigate to Artifact Editor

Name: Test data store

The below data stored in the “TestDataSourceDB” → testdata (table)

Overview / Monitor Message Processing / Message Processing Run

Run Steps (4)

End	Segment 1	3 ms
Write 1	Segment 1	17 ms
HTTPS	Segment 1	3 ms
HTTPS	Segment 1	< 3 ms

[Integration Flow Model](#) [Log Content](#) [Message Content](#)

Message before Step

[Header](#) [Exchange Properties](#) [Payload](#)

```

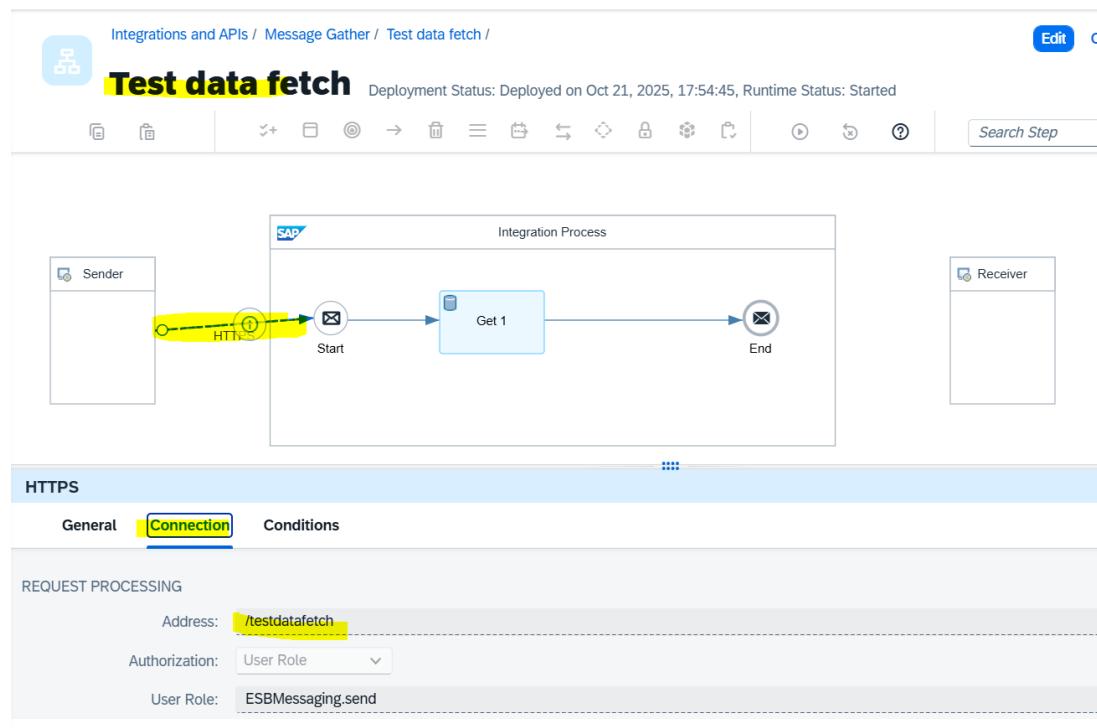
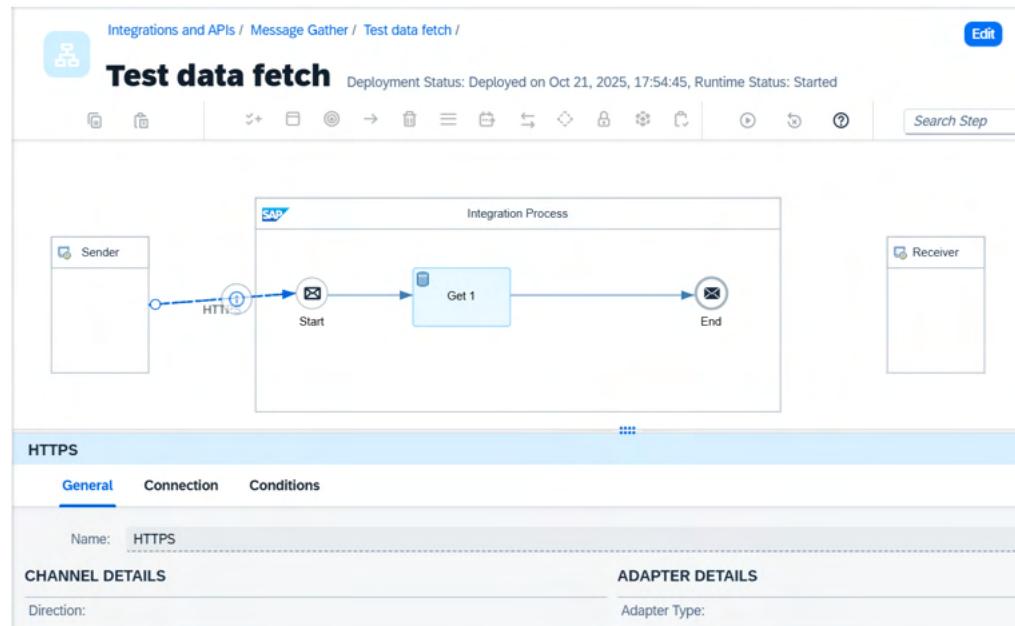
<Root>
<Personal>
<FirstName>Abhimanyu</FirstName>
<LastName>Yadav</LastName>
<Price>200</Price>
</Personal>
<Personal>
<FirstName>Abhishek</FirstName>
<LastName>Yadav</LastName>
<contact>
<phone>224555 </phone>
</contact>
</Personal>
<Personal>
<FirstName>Shekhar</FirstName>
<LastName>Yadav</LastName>
<Price>300</Price>
<contact>
<phone>99933388</phone>
</contact>
</Personal>
<Personal>
<FirstName>Abhimanyu</FirstName>
<LastName>Yadav</LastName>
<Price>100</Price>
</Personal>

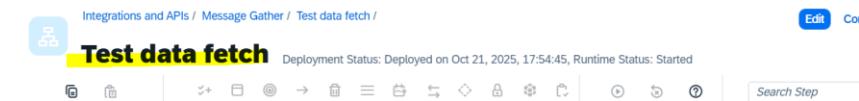
```

Dow

- **Retrieve / Fetch / Get the data from the “TestDataStoreDB” → testdata (table)**

Note: This DB is configured as “Global”





Get

- General
- Processing**

Data Store Name: **TestDataStoreDB**

Visibility: Global

Entry ID: **testdata**

Delete On Completion:

Throw Exception on Missing Entry:

Note: Provide the same name in “data Store Name” as DB name (TestDataStoreDB) and table name (testdata) because while creation / writing time we used the same names.

- Save and Deploy
- Execute from POSTPAN by calling GET URL to fetch the data from DB/Table

```

<Root>
  <Personal>
    <FirstName>Abhimanyu</FirstName>
    <LastName>Yadav</LastName>
    <Price>200</Price>
  </Personal>
  <Personal>
    <FirstName>Abhishek</FirstName>
    <LastName>Yadav</LastName>
    <contact>
      <phone>2244555 </phone>
    </contact>
  </Personal>
  <Personal>
    <FirstName>Shekhar</FirstName>
    <LastName>Yadav</LastName>
    <Price>300</Price>
  </Personal>

```

The same information can be observed from iFlow Message /Monitoring section:

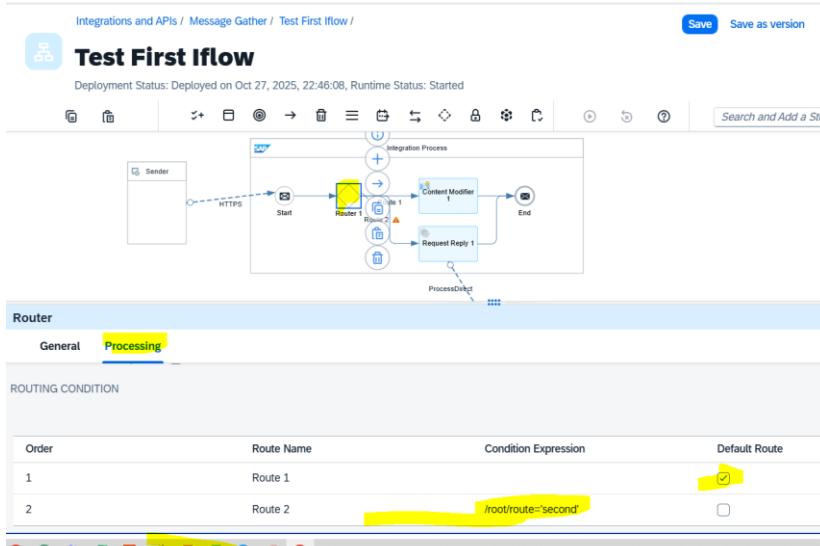
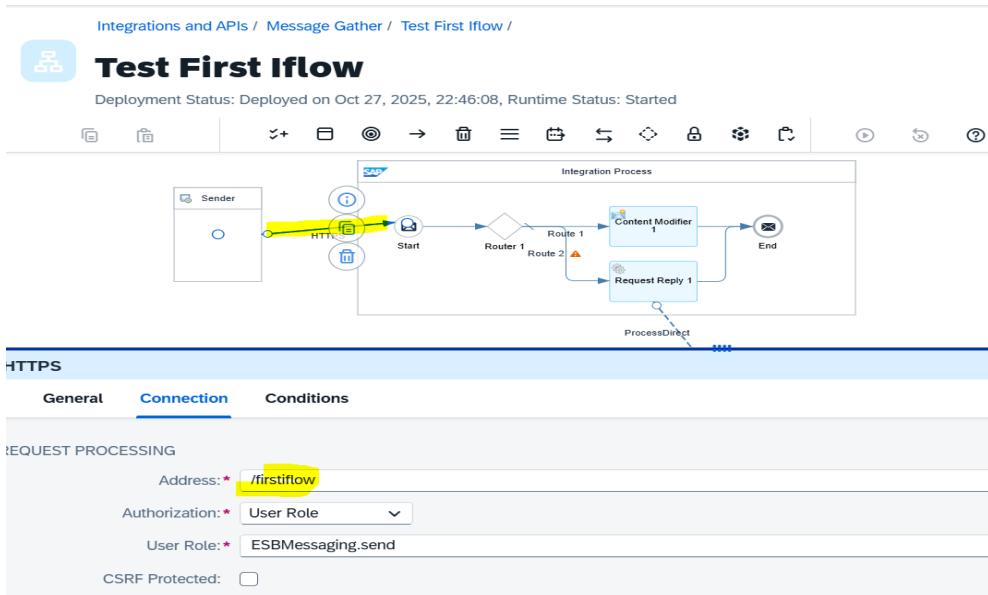
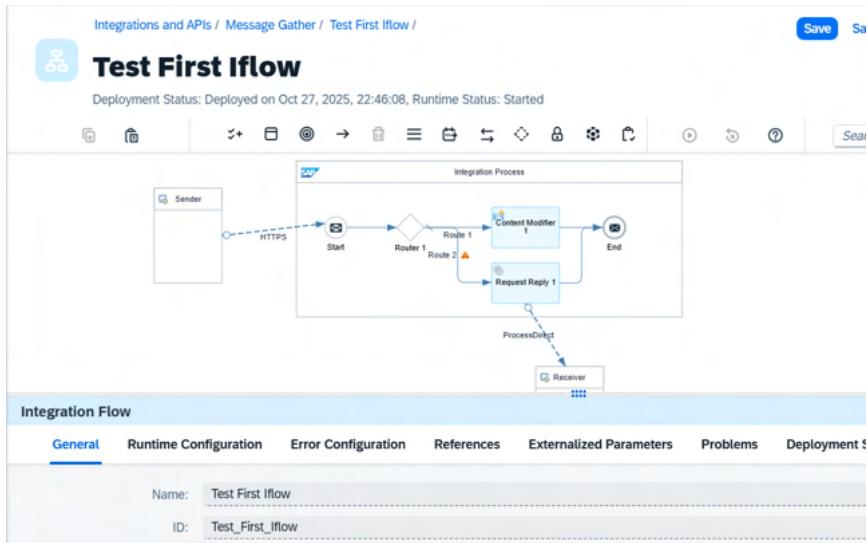
```

<Root>
  <Personal>
    <FirstName>Abhimanyu</FirstName>
    <LastName>Yadav</LastName>
    <Price>200</Price>
  </Personal>
  <Personal>
    <FirstName>Abhishek</FirstName>
    <LastName>Yadav</LastName>
    <contact>
      <phone>2244555 </phone>
    </contact>
  </Personal>
  <Personal>
    <FirstName>Shekhar</FirstName>
    <LastName>Yadav</LastName>
    <Price>300</Price>
  </Personal>

```

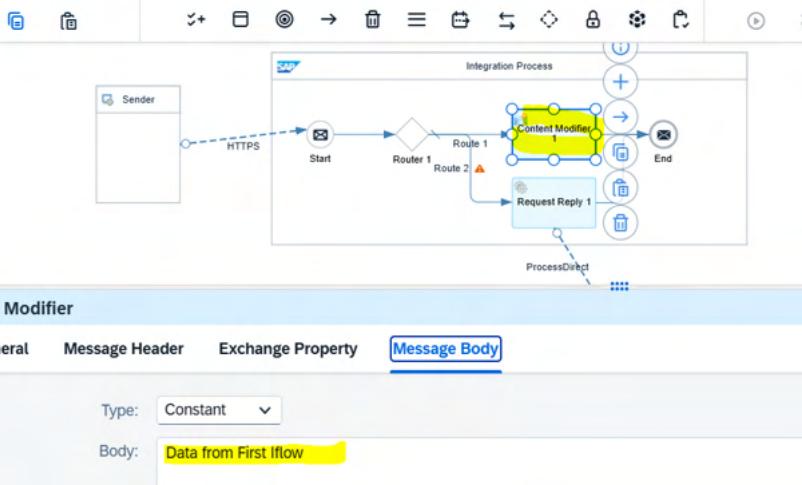
Scenerio 8: Create two iFlows and configure one iFlow to call the other.

- Create first IFlow:



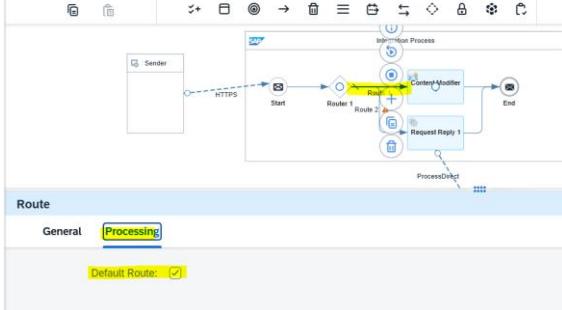
Test First Iflow

Deployment Status: Deployed on Oct 27, 2025, 22:46:08, Runtime Status: Started



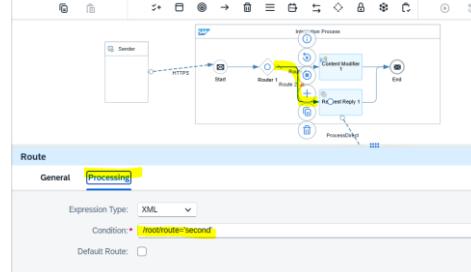
Test First Iflow

Deployment Status: Deployed on Oct 27, 2025, 22:46:08, Runtime Status: Started



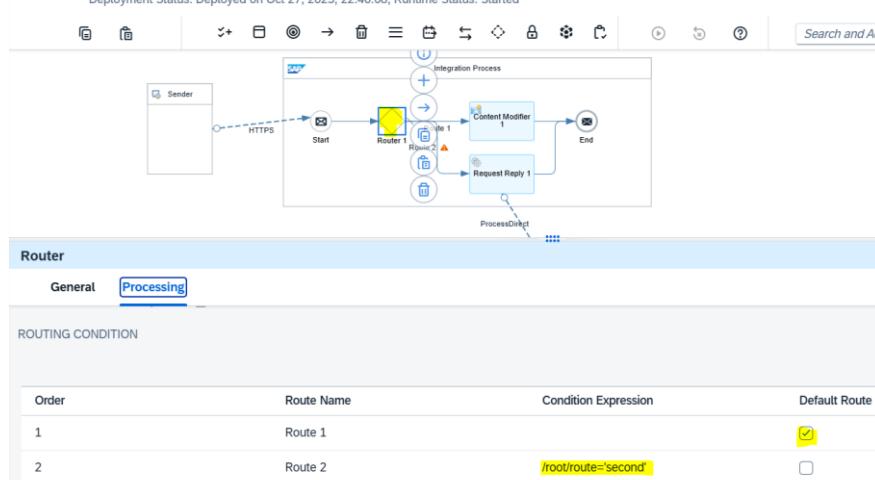
Test First Iflow

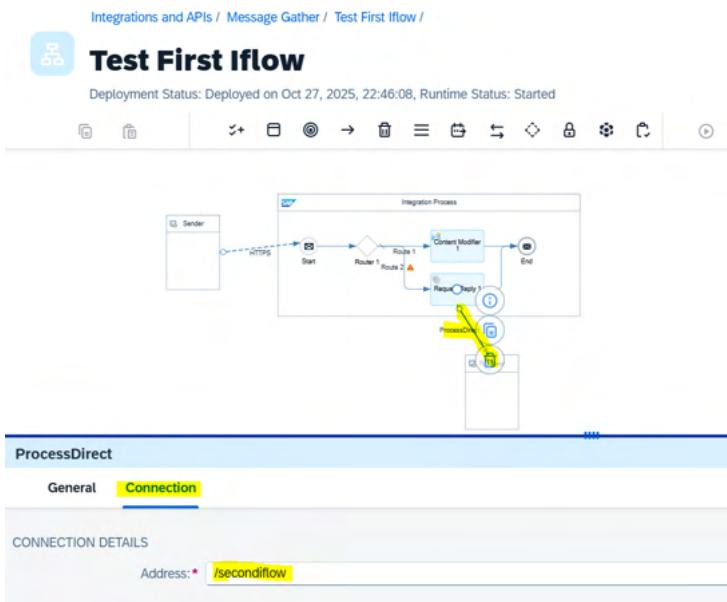
Deployment Status: Deployed on Oct 27, 2025, 22:46:08, Runtime Status: Started

**Save** Save as versi

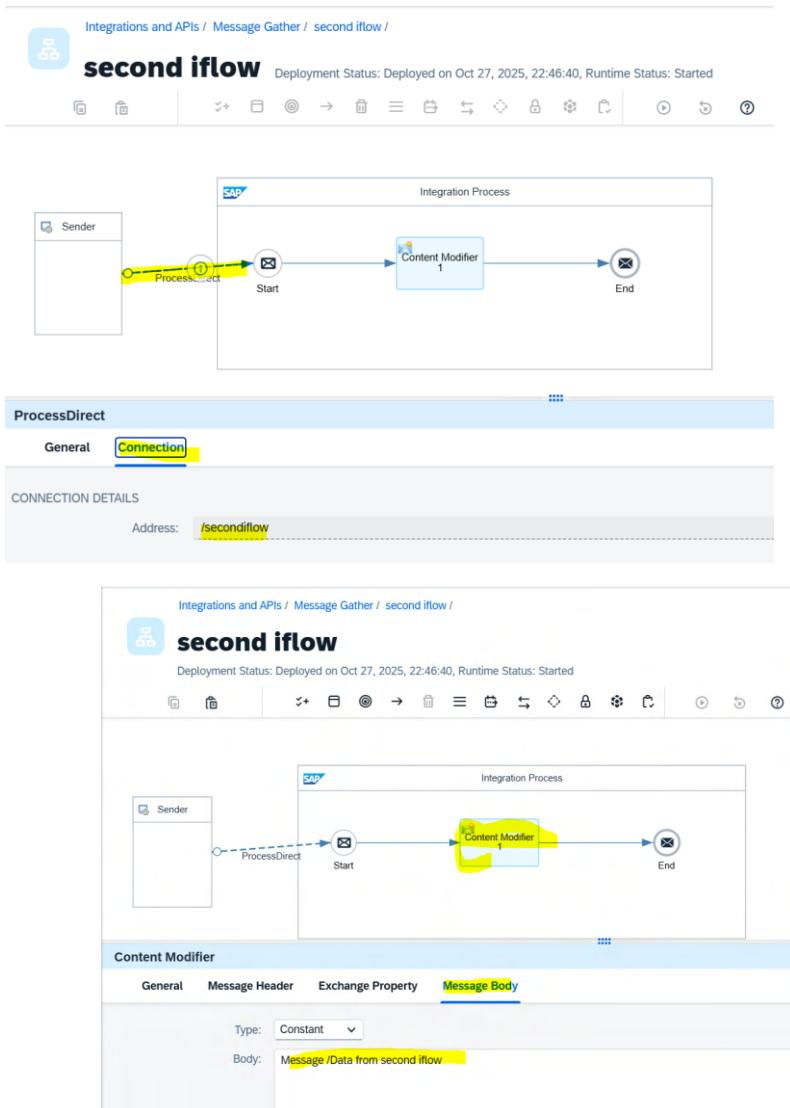
Test First Iflow

Deployment Status: Deployed on Oct 27, 2025, 22:46:08, Runtime Status: Started





- Create 2nd Iflow:



Validation using POSTMAN:

If the input in the <route> tag is **anything other than “second”**, the response will come from the **first iFlow** based on its configured body.

If the <route> tag value is **“second”**, the response will come from the **second iFlow** as per its configured body.

The image displays two separate instances of the POSTMAN application interface, each showing a request to a different iFlow endpoint.

Top Screenshot (http://firstiflow):

- Request URL:** https://1284bac5trial.it-cptrial03-rt.cfapps.ap21.hana.ondemand.com/http/firstiflow
- Method:** POST
- Body Content:**

```
1 <root>
2   <route>first</route>
3   <!--
4   <route>first</route>
5   <route>second</route>
6   -->
7 </root>
```
- Response Status:** 200 OK
- Response Body:** Data from First Iflow

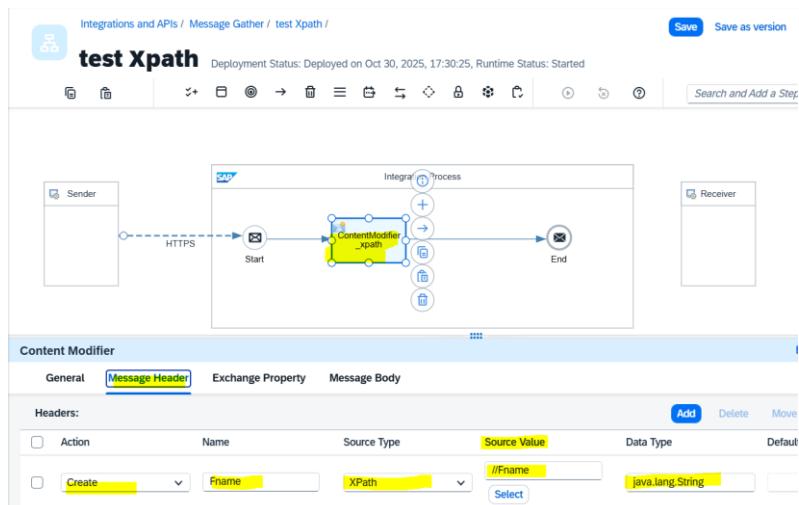
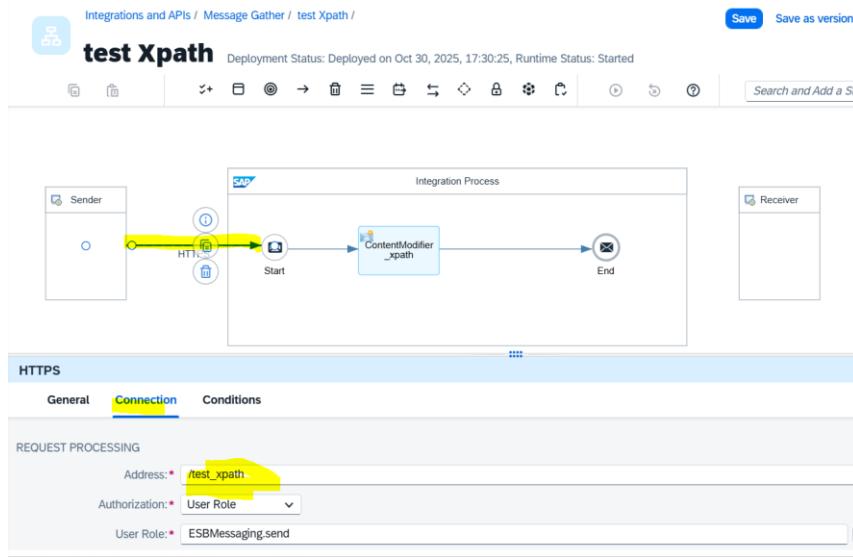
Bottom Screenshot (http://secondiflow):

- Request URL:** https://1284bac5trial.it-cptrial03-rt.cfapps.ap21.hana.ondemand.com/http/secondiflow
- Method:** POST
- Body Content:**

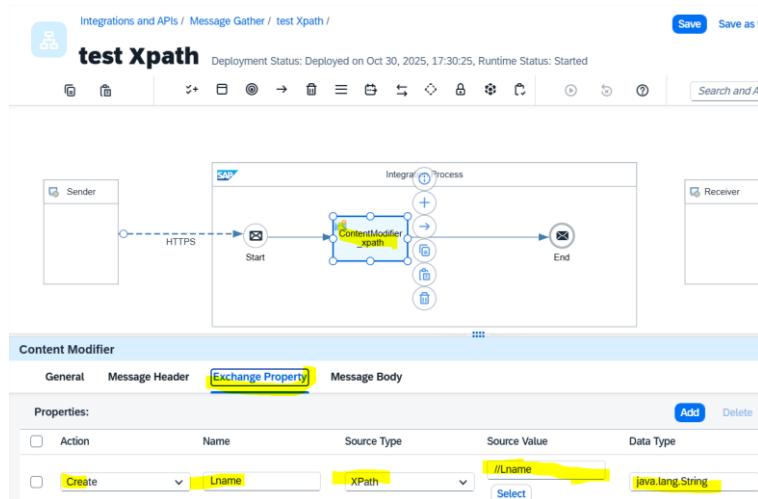
```
1 <root>
2   <route>second</route>
3   <!--
4   <route>first</route>
5   <route>second</route>
6   -->
7 </root>
```
- Response Status:** 200 OK
- Response Body:** Message /Data from second iflow

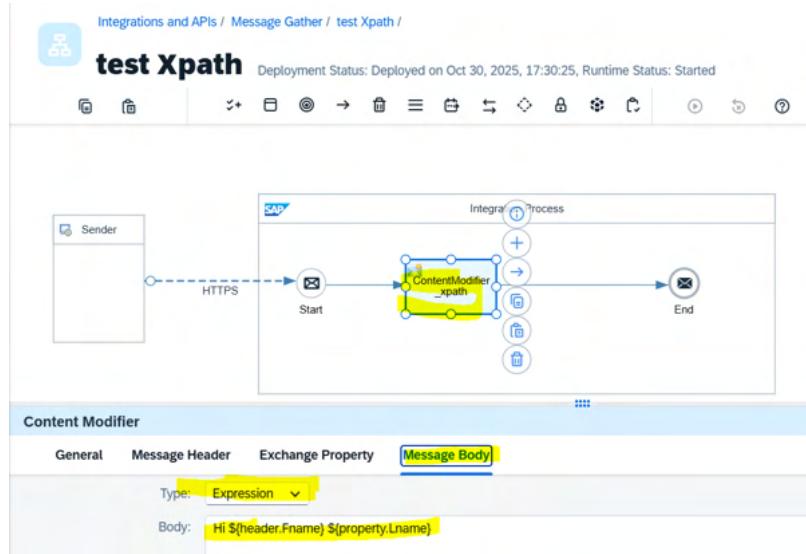
Scenario 9: XPath in Cloud Integration

XPath is used to identify the exact **tag path or hierarchy** in the input/source XML file. For example, to access <Fname> or <Lname>, the precise path from the input XML must be configured in the iFlow.



Note: "Source Value" (`//Fname`) to be given exact HTML tag in XML file. Or path can be given line "`/root/Fname`".





- Call the URL from POSTMAN to see the output.

HTTP https://1284bac5trial.it-cptrial03-rt.cfapps.ap21.hana.ondemand.com/http/test_xpath

POST https://1284bac5trial.it-cptrial03-rt.cfapps.ap21.hana.ondemand.com/http/test_xpath

Params Authorization Headers (10) Body Scripts Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL XML

```

1 <root>
2   <Fname>Sreenivasa Rao</Fname>
3   <Lname>K</Lname>
4 </root>

```

Body Cookies Headers (14) Test Results 200 OK 180 ms 565 B

Raw Preview Visualize 1 Hi Sreenivasa Rao

HTTP https://1284bac5trial.it-cptrial03-rt.cfapps.ap21.hana.ondemand.com/http/test_xpath

POST https://1284bac5trial.it-cptrial03-rt.cfapps.ap21.hana.ondemand.com/http/test_xpath

Params Authorization Headers (10) Body Scripts Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL XML

```

1 <root>
2   <Fname>Sreenivasa Rao</Fname>
3   <Lname>Koppu</Lname>
4 </root>

```

Body Cookies Headers (14) Test Results 200 OK 2.22 s 569 B

Raw Preview Visualize 1 Hi Sreenivasa Rao Koppu

HTTP https://1284bac5trial.it-cptrial03-rt.cfapps.ap21.hana.ondemand.com/http/test_xpath

GET https://1284bac5trial.it-cptrial03-rt.cfapps.ap21.hana.ondemand.com/http/test_xpath

Params Authorization Headers (10) Body Scripts Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL XML

```

1 <root>
2   <Fname>SRao</Fname>
3   <Lname>Koppu</Lname>
4 </root>

```

Body Cookies Headers (14) Test Results 200 OK 830 ms 549 B

Raw Preview Visualize 1 Hi SRao Koppu

Scenario 10: Converter (JSON to XML)

Integrations and APIs / Message Gather / Json_to_xml_converter /

Json_to_xml_converter

Deployment Status: Deployed on Oct 31, 2025, 13:01:40, Runtime Status: Started

Integration Process

HTTPS

- General
- Connection**
- Conditions

REQUEST PROCESSING

- Address: * **/json_to_xml**
- Authorization: * User Role
- User Role: * ESBMessaging.send
- CSRF Protected:

Integrations and APIs / Message Gather / Json_to_xml_converter /

Json_to_xml_converter

Deployment Status: Deployed on Oct 31, 2025, 13:01:40, Runtime Status: Started

Integration Process

JSON To XML Converter

- General
- Processing**

Name: **JSON to XML Converter**

Integrations and APIs / Message Gather / Json_to_xml_converter /

Json_to_xml_converter

Deployment Status: Deployed on Oct 31, 2025, 13:01:40, Runtime Status: Started

Integration Process

JSON To XML Converter

- General
- Processing**

Use Namespace Mapping:

JSON Prefix Separator: Colon(:)

Add XML Root Element:

Name: * **MT_Custom**

Namespace Mapping: **xmNs:ns0=http://cpi.sap.com/demo**

Note: Here “MT_Custom” is the name space in the XML file (i.e default is “root”) Namespace Mapping: copy / configure the Namespace path:

Namespace configuration:

- Click on outside iFlow

- Select “Runtime Configuraiton”
- Add the namespace Mapping (the below snamp shot namespace taken from the tutorial where it is working) → xmlns:ns0=http://cpi.sap.com/demo;xmlns:ns1=http://sap.com/xi/XI/SplitAndMerge

Integrations and APIs / Message Gather / Json_to_xml_converter /

Json_to_xml_converter

Deployment Status: Deployed on Oct 31, 2025, 13:01:40, Runtime Status: Started

Save Save as v Logs Search and

Integration Flow

General Runtime Configuration Error Configuration References Externalized Parameters Problems Deployment Status

Runtime Profile: * Cloud Integration

Namespace Mapping: xmlns:ns0=http://cpi.sap.com/demo;xmlns:ns1=http://sap.com/xi/XI/SplitAndMerge

Allowed Header(s):

HTTP Session Reuse: None

- Save and deploy and execute from POSTMAN

HTTP https://1284bac5trial.it-cpitrial03-rt.cfapps.ap21.hana.ondemand.com/http/test_xpath

GET https://1284bac5trial.it-cpitrial03-rt.cfapps.ap21.hana.ondemand.com/http/json_to_xml

Params Authorization Headers (10) Body Scripts Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```

1 {
2   "row": [
3     {
4       "CustomerID": "NEWCID1",
5       "CompanyName": "NEWCOMPANY1",
6       "Country": "France"
7     },
8     {
9       "CustomerID": "NEWCID2",
10      "CompanyName": "NEWCOMPANY2",
11      "Country": "Italy"
12    }
13  ]
14}
  
```

Body Cookies Headers (15) Test Results 200 OK 791 ms 944 B

% XML Preview Visualize

```

1 <?xml version='1.0' encoding='UTF-8'?>
2 <ns0:MT_Custom xmlns:ns0="http://cpi.sap.com/demo">
3   <ROW>
4     <CustomerID>NEWCID1</CustomerID>
5     <CompanyName>NEWCOMPANY1</CompanyName>
6     <Country>France</Country>
7   </ROW>
8   <ROW>
9     <CustomerID>NEWCID2</CustomerID>
10    <CompanyName>NEWCOMPANY2</CompanyName>
11    <Country>Italy</Country>
12  </ROW>
13 </ns0:MT_Custom>
  
```