



TGH

Making Integrations Simpler

boomi
Partner



Validation of XML file against XSD Schema



Validation of XML file against XSD Schema

In this blog, we will validate the XML file against the XSD Schema file using Groovy Scripting.

What is XSD?

XSD stands for XML schema definition. XSD files are made up of tags just like an XML file. Like an XML file, an XSD file has elements and each element must have a name and a type. It describes the relationship between elements and attributes in an XML document and also how the XML files should be structured.

In this Use Case, we will read the XSD schema file from disk and store it in a Dynamic Process Property. We will hard code the XML files in message shape and validate it against dynamic process property in Groovy script.

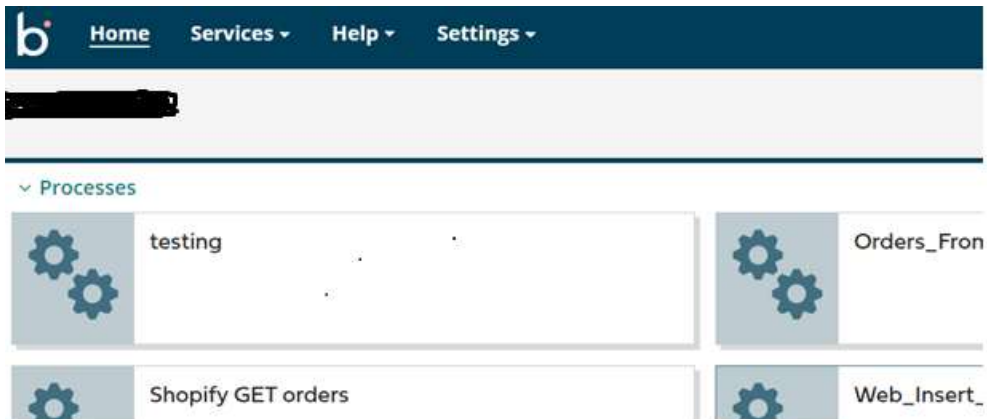
There are 2 Dynamic Document Properties created out of Groovy script using which we check if the XML files are routed to the true or false branch. True indicates that the XML file is in the correct format and sent to disk and the false branch contains the XML file with a bad format.

Let us implement the use case by creating the process in Boomi.

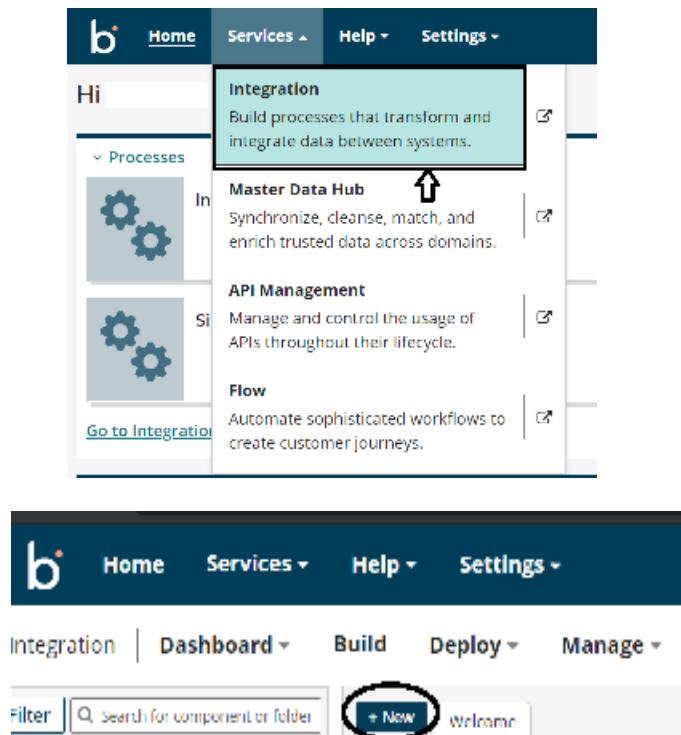
Step 1: Log on to the Boomi platform (<https://platform.boomi.com/>) with the required credentials i.e. Email Address and Password.



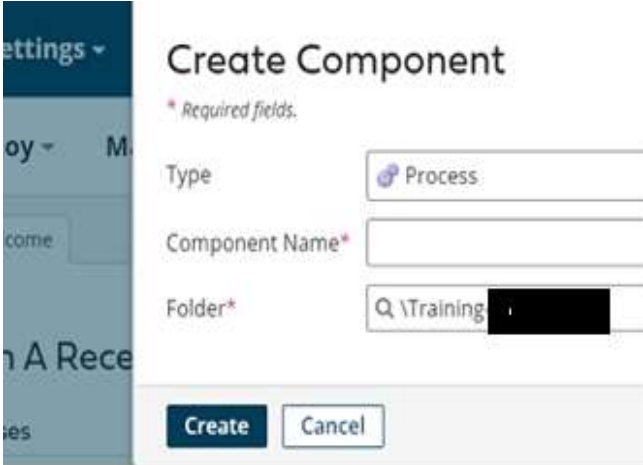
Step 2: Once logged into the Boomi platform, we will be able to view the Home page.



Step 3: Now, click on Services followed by Integration. We will see the Build page. Click on New.

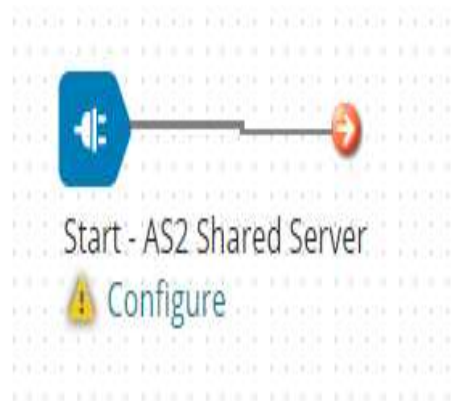


Step 4: Once, click on New, we will be able to see three fields i.e. Type, Component Name and Folder.

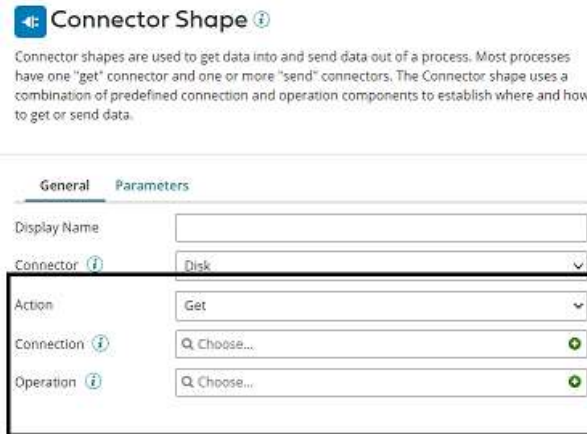


- Select Type as process as we are building a process. Component Name and Folder can be given based on your choice (i.e. which name to be given and where do we want to create the process). Click on Create.

Step 5: We see that the process gets created with a start shape which is configured with AS2 Shared Server by default.



Step 6: Select the start shape and choose Type as Connector as we are getting the XSD schema file from Disk Connector. Here, we have to configure 3 fields in connector i.e. Action, Connector and Operation. Choose connector as Disk.



Connector Shape

Connector shapes are used to get data into and send data out of a process. Most processes have one "get" connector and one or more "send" connectors. The Connector shape uses a combination of predefined connection and operation components to establish where and how to get or send data.

General Parameters

Display Name

Connector

Action

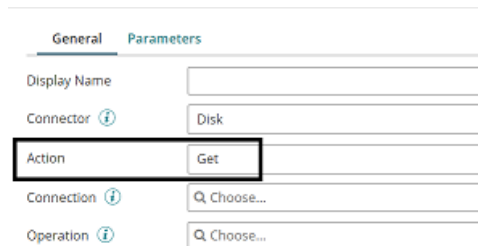
Connection

Operation

Get – To get the data from a disk location.

Send – To send to the disk location.

- Here, we will choose action as **GET** as we are reading the file.



General Parameters

Display Name

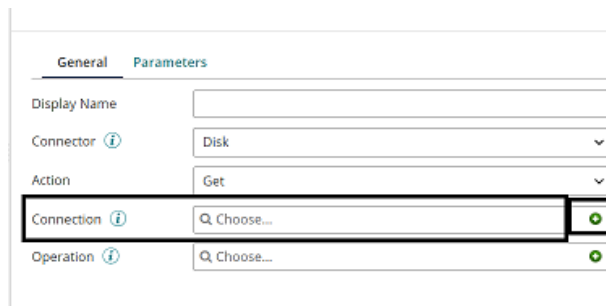
Connector

Action

Connection

Operation

- Click + on connection to create a new one.



General Parameters

Display Name

Connector

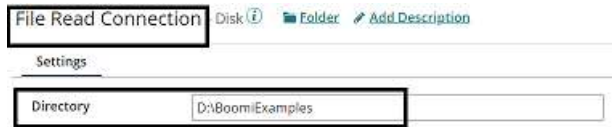
Action

Connection

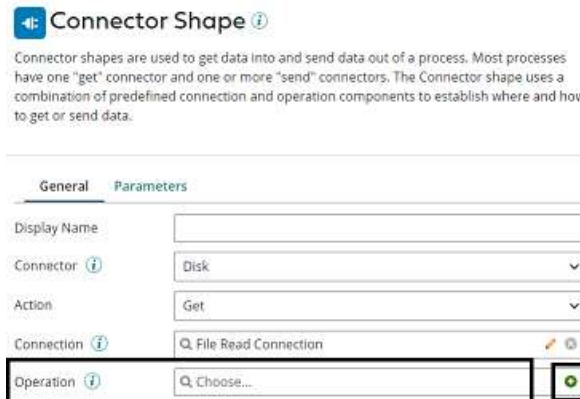
Operation

- Name the file and give the directory from where we want to read the file. Here, we are reading the XSD file from the D drive and BoomiExamples folder as shown in the screenshot.

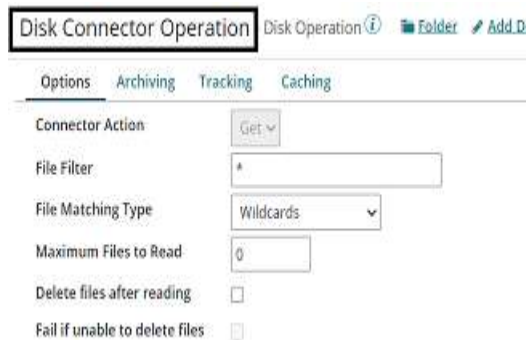
This PC > Data (D:) > BoomiExamples



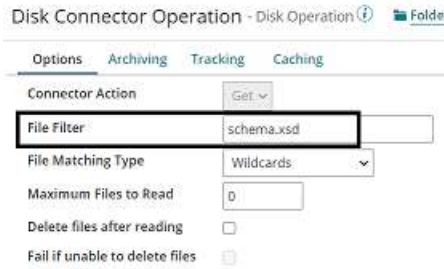
- Click save and close
- Now, we will configure the operation. Click + on operation to create a new one.



- Name the operation and configure the following.



- **File Filter:** Read-only files with a file name that matches the file filter. Here, it will be **schema.XSD**



Disk Connector Operation - Disk Operation ⓘ Folder

Options Archiving Tracking Caching

Connector Action Get ▾

File Filter schema.xsd

File Matching Type Wildcards ▾

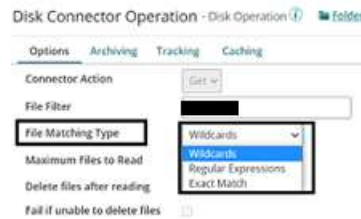
Maximum Files to Read 0

Delete files after reading

Fail if unable to delete files

File Matching Type:

- **Wildcards** uses simple file filters like * and. * represent multiple characters and ? represents a single character.
- **Regular Expressions** can include complex regular expressions.
- **Exact Match** includes the filename that we are reading.



Disk Connector Operation - Disk Operation ⓘ Folder

Options Archiving Tracking Caching

Connector Action Get ▾

File Filter [Redacted]

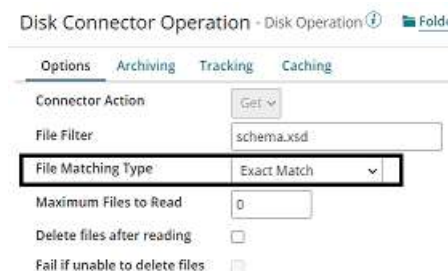
File Matching Type Wildcards ▾

Maximum Files to Read 0

Delete files after reading

Fail if unable to delete files

- Here, the file matching type would be **Exact Match** as we are giving the file name.



Disk Connector Operation - Disk Operation ⓘ Folder

Options Archiving Tracking Caching

Connector Action Get ▾

File Filter schema.xsd

File Matching Type Exact Match ▾

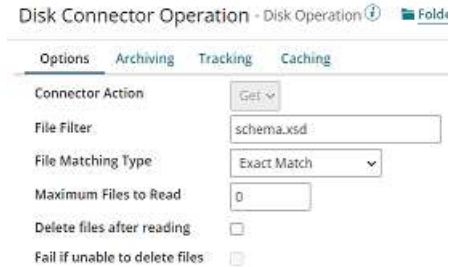
Maximum Files to Read 0

Delete files after reading

Fail if unable to delete files

- **Maximum files to read:** It sets the maximum number of files to be read at one time. Let it be default i.e..

- **Delete files after reading:** If we want the file to be read and deleted, we can check this option. Here, we are leaving it to default. Click save and close
- The complete disk operation looks like this,



Disk Connector Operation - Disk Operation ⓘ Fold

Options Archiving Tracking Caching

Connector Action: Get

File Filter: schema.xsd

File Matching Type: Exact Match

Maximum Files to Read: 0

Delete files after reading:

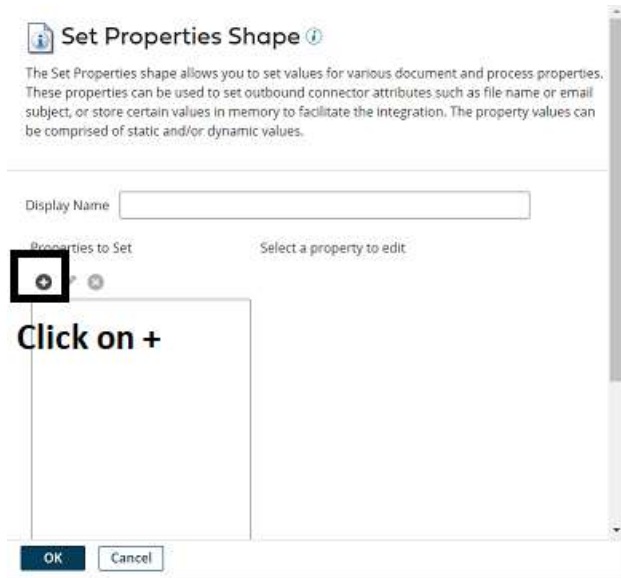
Fail if unable to delete files:

- This is how the sample XSD file looks like,

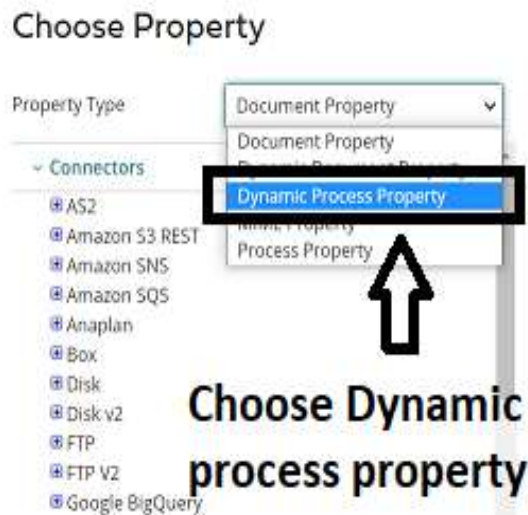
```
<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
  <xsd:element name = "contact">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name = "name" type = "xsd:string" />
        <xsd:element name = "company" type = "xsd:string" />
        <xsd:element name = "phone" type = "xsd:int" />
        <xsd:element name = "role" type = "xsd:string"/>
        <xsd:element name = "place" type = "xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```


Step 7: Drag and drop set properties shape onto the process canvas to create Dynamic Process Property.

- Click + on Properties as shown.



- Choose Dynamic Process Property from the drop-down.



- Give the property name as “xsd schema” and click ok.

Choose Property

Property Type:

Property Name*:

Options: Persist this property

- Now, we have to set parameters for the Dynamic Process Property. Select the Dynamic Process property and choose + on parameters.



Step 8: Here, we are selecting Type as current data which holds the XSD file coming from disk. Click ok.

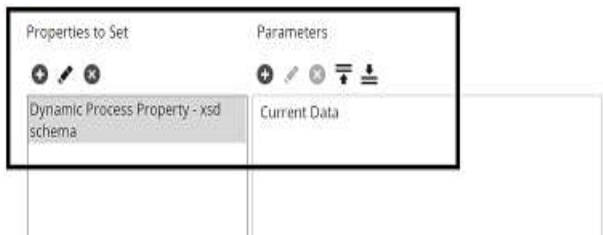
Parameter Value

Type:

Static Value:

- Static
- Connector Call
- Cross Reference Lookup
- Current Data**
- Date/Time
- Document Cache Lookup
- Document Property
- Execution Property
- Dynamic Process Property
- Process Property
- Profile Element
- Sequential Value
- SQL Statement
- Static
- Stored Procedure
- Unique Value

- After setting the value for Dynamic Process Property, it looks like as follows.



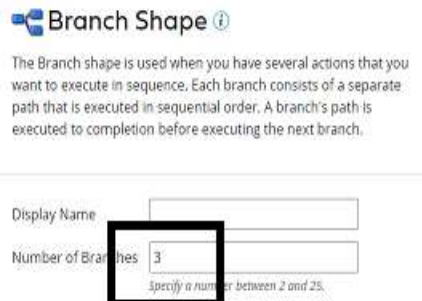
Step 9: Drag and drop the message shape onto the process canvas and place the correct format of hard-coded XML data in the message body as shown. Click ok.



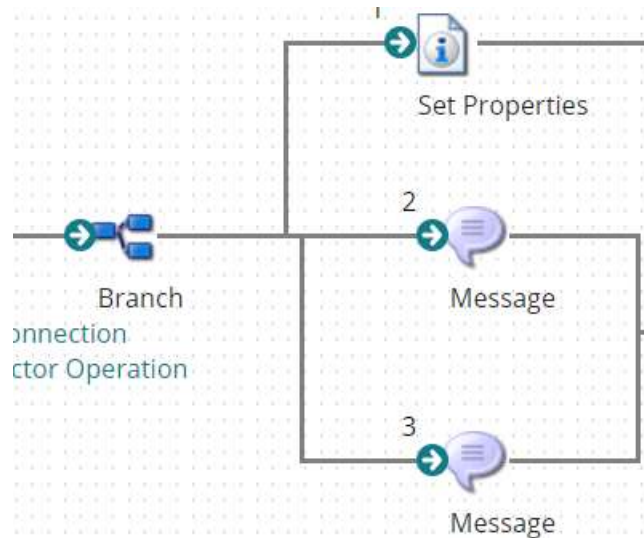
Step 10: Also, add another message shape and place the incorrect format of the XML file in the message body. Click ok.



Step 11: Drag and drop branch shape and set the number of branches to 3.



Step 12: Set the 1st branch to Set Properties shape which holds Dynamic Process property and the rest of the 2 branches are set to the message shapes where we hard code the values.



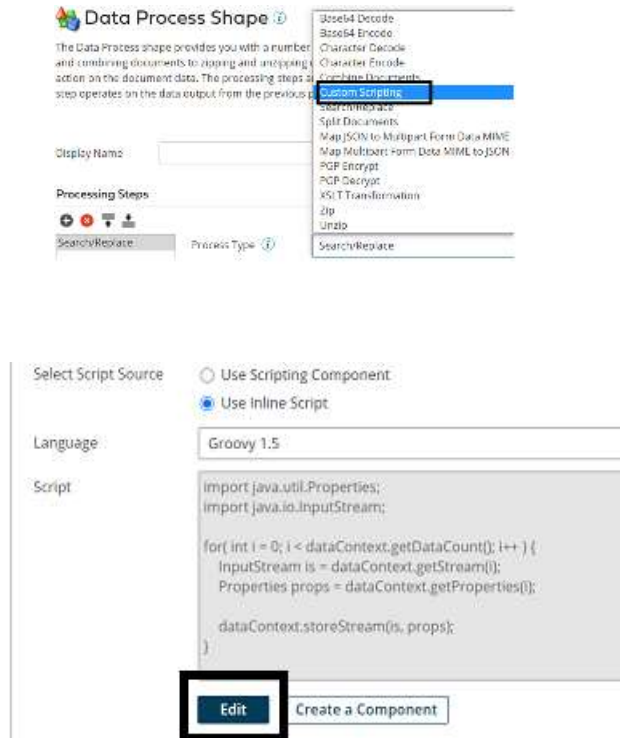
Step 13: Drag and drop the data process shape onto the process canvas to write a Groovy script which validates XML files against xsd files.



Step 14: Select Data Process Shape and click +.



- Choose Process Type as custom scripting from the drop-down. Click on edit.



Step 15: Now, we will be able to see the command prompt as shown in the screenshot.



- Now we write a groovy script which validates XML files against Dynamic Process Property and creates Dynamic Document Property as “Format” in the script with 2 values as True and False.

```

import java.util.Properties;
import java.io.InputStream;
import com.boomi.execution.ExecutionUtil;
import org.xml.sax.SAXException;
import javax.xml.XMLConstants;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.Source;
import javax.xml.transform.stream.StreamSource;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;
import javax.xml.validation.Validator;

for (int i = 0; i < dataContext.getDataCount(); i++)
{
    InputStream is = dataContext.getStream(i);
    Properties props = dataContext.getProperties(i);

    Source xmldata = new StreamSource(is);
    String schemaDPP = ExecutionUtil.getDynamicProcessProperty("xsd schema");
    Source schemaFile = new StreamSource(new StringReader(schemaDPP));
    SchemaFactory factory = SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
    Schema schema = factory.newSchema(schemaFile);
    Validator validator = schema.newValidator();

    try {
        validator.validate(xmldata);
        props.setProperty("document.dynamic.userdefined.Format", "true");
    } catch (SAXException e) {
        props.setProperty("document.dynamic.userdefined.Format", "false");
    }

    is.reset();
    dataContext.storeStream(is, props);
}
    
```

Have to import these classes to validate xml files against XSD file

Here, we are getting xml files and storing it in a variable called as xmldata. Also, getting DPP i.e "xsd schema" and storing it in schemaDPP and creating a SchemaFactory capable of understanding WXS schema and Validator instance which is used to validate the instance documents

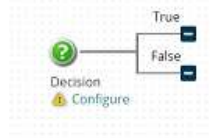
We are creating DPP as Format and validating the xml documents. If Format value is set to true will go to true branch and false will go to false branch

- Add script in the command prompt and click ok.

```

import java.util.Properties;
import java.io.InputStream;
import com.boomi.execution.ExecutionUtil;
import org.xml.sax.SAXException;
import javax.xml.XMLConstants;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.Source;
import javax.xml.transform.stream.StreamSource;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;
import javax.xml.validation.Validator;
for (int i = 0; i < dataContext.getDataCount(); i++)
{
    InputStream is = dataContext.getStream(i);
    Properties props = dataContext.getProperties(i);
    Source xmldata = new StreamSource(is);
    String schemaDPP = ExecutionUtil.getDynamicProcessProperty("xsd schema");
    Source schemaFile = new StreamSource(new StringReader(schemaDPP));
    SchemaFactory factory = SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
    Schema schema = factory.newSchema(schemaFile);
    Validator validator = schema.newValidator();
    try {
        validator.validate(xmldata);
        props.setProperty("document.dynamic.userdefined.Format", "true");
    }
    catch (SAXException e)
    {
        props.setProperty("document.dynamic.userdefined.Format", "false");
    }
    is.reset();
    dataContext.storeStream(is, props);
}
    
```

Step 16: Drag and drop the Decision shape onto the process canvas to validate with Dynamic Document Property which was created in custom script. If the value of the Dynamic Document property is True then it goes to the True Branch and if the value is false it goes to the False branch.



Step 17: Click on the shape and configure the first value.

Display Name	<input type="text"/>
First Value	<input type="text" value="Choose..."/>
Comparison	<input type="text" value="Equal To"/>
Second Value	<input type="text" value="Choose..."/>

Step 18: Here, the first value will be Dynamic Document Property which was created in a custom script. The type will be document property.

Parameter Value

Type:

Static Value:

Static Value

- Static
- Connector Call
- Cross Reference Lookup
- Current Data
- Date/Time
- Document Property
- Dynamic Property
- Dynamic Process Property
- Process Property
- Profile Element
- SQL Statement
- Static
- Stored Procedure

Step 19: Choose Dynamic Property as Dynamic Document Property as shown.

Choose Property

Property Type:

Connectors:

- AS2 Server
- Amazon S3
- ...

Document Property

Dynamic Document Property

Static Property

Step 20: Give Dynamic Document Property name which got generated in the script. Here, the property name is Format. Click ok.

Choose Property

Property Type	Dynamic Document Property
Property Name*	Format
Default Value	

Step 21: Comparison will be Equal To.

Display Name	
First Value	Document Property - Dynamic Document Property - Format
Comparison	Equal To
Second Value	Choose...

Step 22: Click on the Second value and choose Type as static and the Static value will be true.

Display Name	
First Value	Document Property - Dynamic Document Property - Format
Comparison	Equal To
Second Value	Choose...

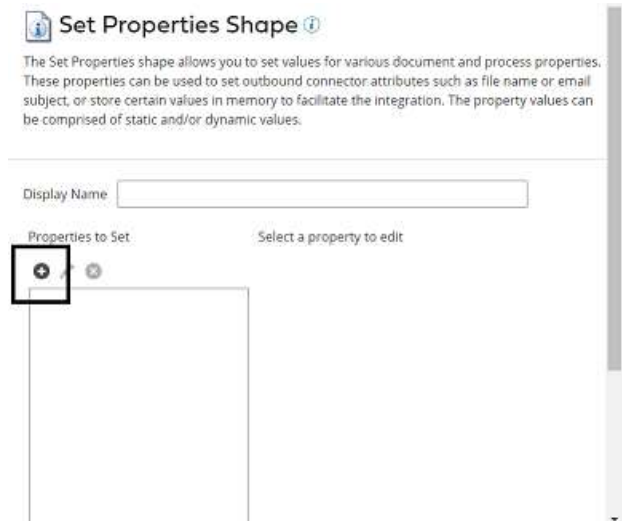
Parameter Value

Type	Static
Static Value	true

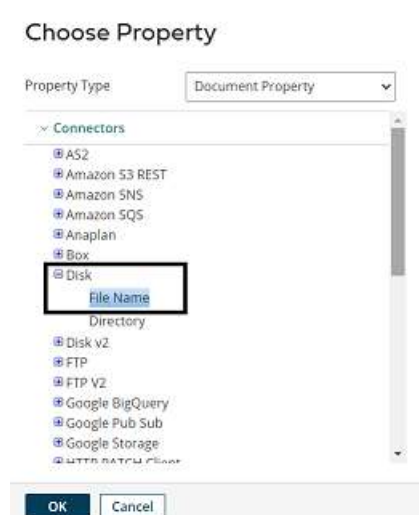
Step 23: After configuring the decision shape, it looks like

Display Name	
First Value	Document Property - Dynamic Document Property - Format
Comparison	Equal To
Second Value	Static value of 'true'

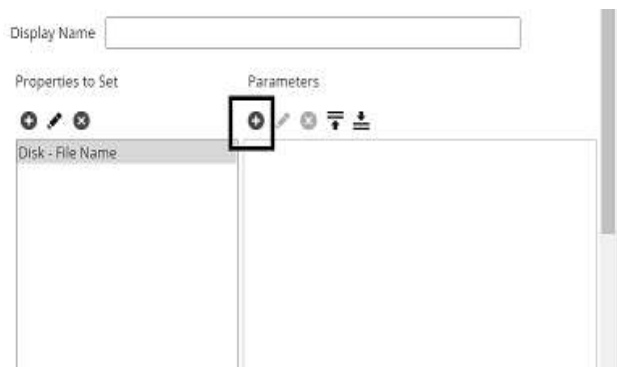
Step 24: In the true branch, we will place set properties shape to set the file name which comes to the disk. Select + on properties.



Step 25: Select the disk connector from Connectors and choose the file name. Click ok.



Step 26: Select + on the parameters side and assign a value to the disk.



Step 27: Choose type as static and static value as contact.xml.

Parameter Value

Type	Static
Static Value	contact.xml

- After configuring the properties in a set properties shape, it looks like

Properties to Set	Parameters
<div style="display: flex; justify-content: space-between;"> + ✎ ✕ </div> <div style="border: 1px solid gray; padding: 2px;">Disk - File Name</div>	<div style="display: flex; justify-content: space-between;"> + ✎ ✕ ⇄ ⇅ </div> <div style="border: 1px solid gray; padding: 2px;">Static value of 'contact.xml'</div>

Step 28: Drag and drop the disk connector shape onto the process canvas to send a response to the disk.



Step 29: We have to configure 3 fields in the connector i.e. Action, Connector and Operation.

Connector Shape ⓘ

Connector shapes are used to get data into and send data out of a process. Most processes have one "get" connector and one or more "send" connectors. The Connector shape uses a combination of predefined connection and operation components to establish where and how to get or send data.

General	Parameters
Display Name	<input type="text"/>
Connector ⓘ	Disk
Action	Get
Connection ⓘ	🔍 Choose...
Operation ⓘ	🔍 Choose...

Get – To get the data from a disk location.

Send – To send to the disk location.

©TGH Software Solutions Pvt. Ltd.

- Here, we will choose action as **SEND** as we are sending a response to the file.

- Click + on connection to create a new one.

- Name the file and give the directory to save the response file. Here, it is the D drive and BoomiExamples folder as shown in the screenshot.

- Click save and close.

Step 30: Click + on operation and name it. Leave everything to default. Click save and close.

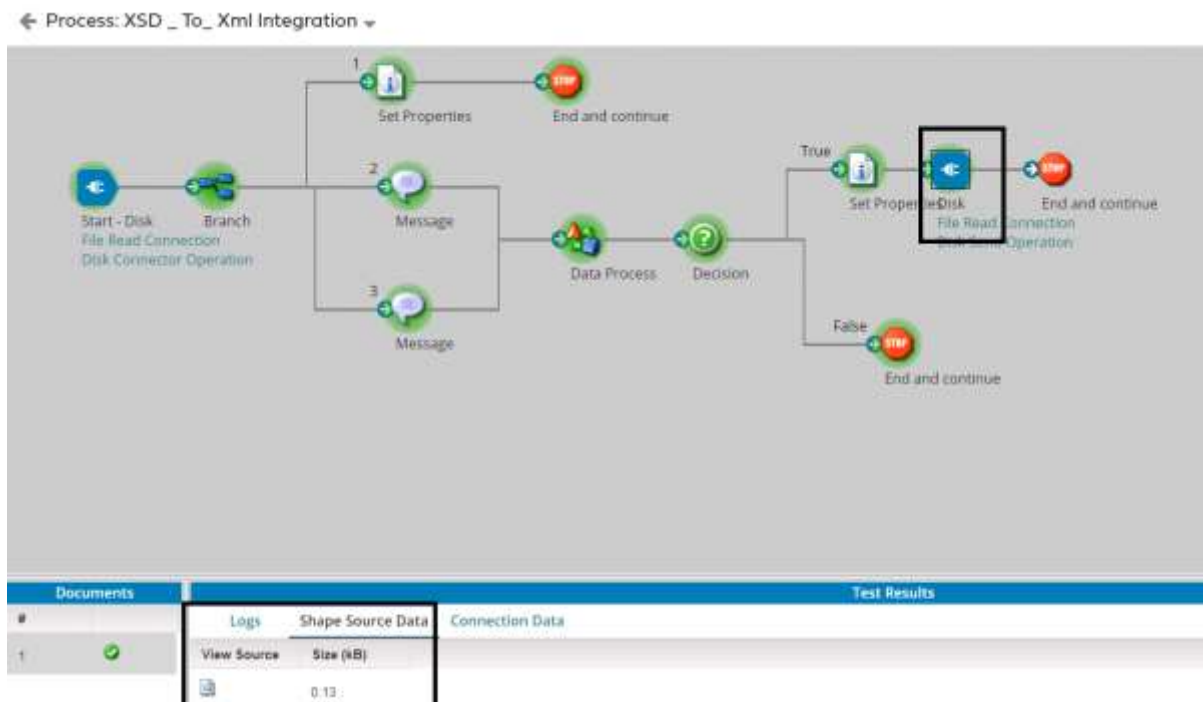
Step 31: In false branch, we will just place a stop shape to indicate the end of the flow.



Step 32: Arrange all the shapes in order and run the test by clicking on Run Test and configuring the local atom.



Step 33: We see that the process is executed and one file has gone to True shape and the other one to False. Click on disk connector and view the source to see the file which went to disk in the True branch.



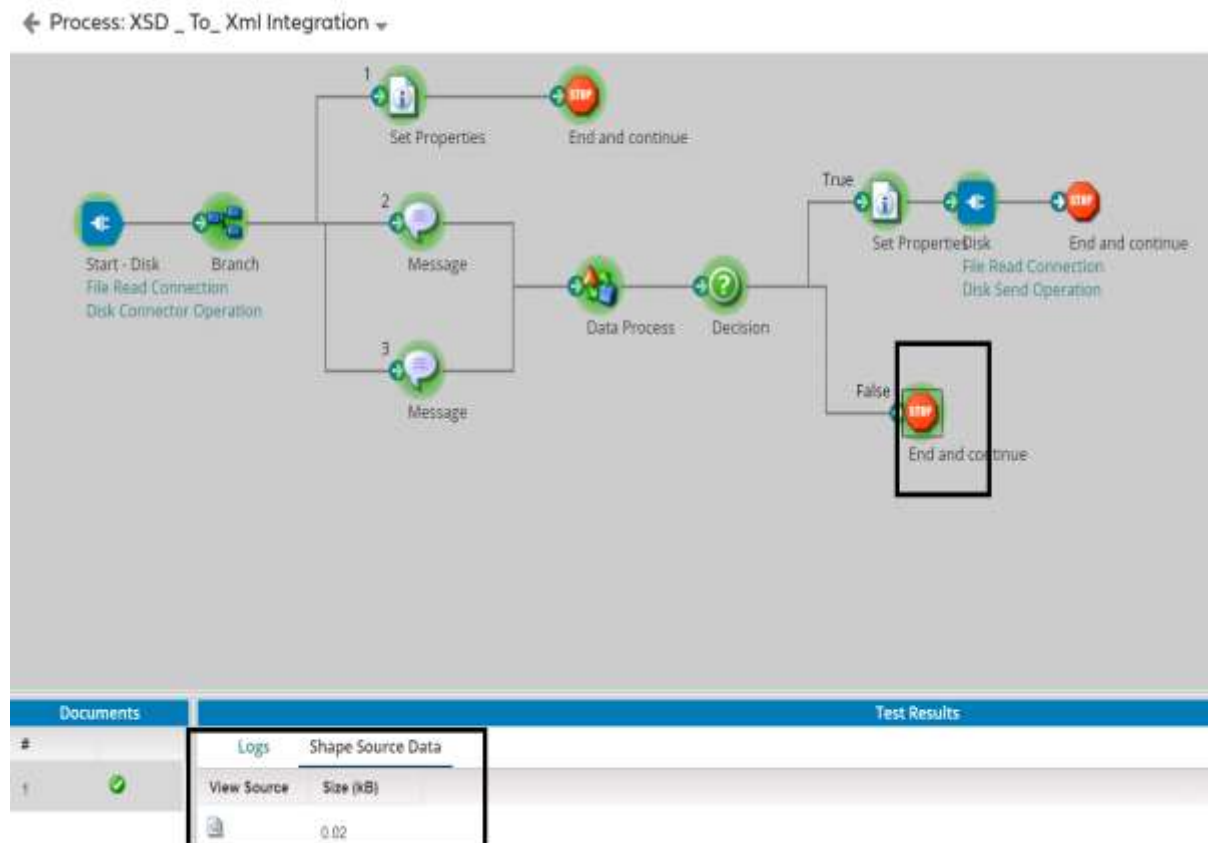
Step 34: Click on view source and we see the XML file with the correct format.

Document Viewer

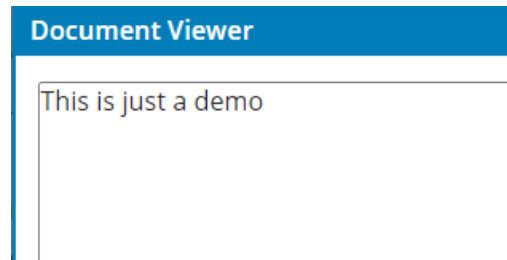
```

1 <contact>
2   <name>Sak</name>
3   <company>Dell</company>
4   <phone>1000123457</phone>
5   <role>Dev Eng</role>
6   <place>Mumbai</place>
7 </contact>
  
```

Step 35: Now, select the False branch and we see that a file is sent.



Step 36: Click on view source and we see the file with incorrect format.



Step 37: Navigate to the folder which we have configured in the disk location and we see that the file named contact.xml is sent to the disk location.



```
<contact>  
<name>Sak</name>  
<company>Dell</company>  
<phone>1000123457</phone>  
<role>Dev Eng</role>  
<place>Mumbai</place>  
</contact>
```



TGH

Making Integrations Simpler



TGH Software Solutions Pvt. Ltd.

www.techygeekhub.com

At TGH, we specialize in driving digital transformation through seamless Integration Technologies.

Operating as an INTEGRATION FACTORY, we serve as a one-stop shop for all your integration needs. Our expert team is well-versed in enterprise software and legacy system integration, along with leading iPaaS technologies like Boomi, MuleSoft, Workato, OIC, and more.

We're committed to enhancing business processes and solving problems through our integration expertise.



Email address

connect@techygeekhub.com



Phone number

+ 011-40071137
+ 91-8810610395



Our offices

Noida Office

iThum
Plot No -40, Tower A,
Office No: 712,
Sector-62, Noida,
Uttar Pradesh, 201301

Hyderabad Office

Plot no: 6/3, 5th Floor,
Techno Pearl Building,
HUDA Techno Enclave,
HITEC City, Hyderabad,
Telangana 500081

