

# XSLT Mapping in SAP PI 7.1



## Applies to:

SAP NetWeaver Process Integration 7.1 (SAP PI 7.1)

## Summary

This document explains about using XSLT mapping in SAP Process Integration for converting a simple input to a relatively complex output. It explains the complete process of preparing a .xsl file and then importing it to PI.

**Authors:** Amit Srivastava, Anshul Chowdhary

**Company:** Infosys Technologies Limited

**Created on:** 13 August 2010

## Author Bio



Amit Srivastava is working as a Senior Software Engineer on SAP XI/PI. He began his career on Nov-2007 and since then he has been working on SAP XI/PI. His area of expertise is SAP XI/PI.



Anshul Chowdhary is working as a Technology Analyst. He began his career on July-2006 and has an experience of around 1 year on DOT NET. He started working on SAP XI/PI from December-2007 and is still hooked to the technology.

## Table of Contents

A Basic Overview on XSLT.....	3
Basic XSLT Tags .....	5
XPath Functions in XSLT Mapping:.....	9
How to Use an XSLT Mapping in PI .....	10
Example 1 .....	11
Example 2 .....	15
References.....	20
Disclaimer and Liability Notice.....	21

## A Basic Overview on XSLT

XSLT stands for EXtensible Stylesheet Language Transformation. When it is not possible to use message mapping, usually when we need to create a complex structure from a flat message or where aggregation of nodes etc is required, we prefer using XSLT mapping. XSLT describes how an XML structure is transformed into another XML structure. It is very simple to use an XSLT mapping in PI. The XSLT is developed and then imported as a zip file into ESR. The structured description of a simple XSL style sheet is as shown below:

### Example

Let's take a simple example which will give a complete idea about XSLT:

- 1.) Let the Source be as shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<ns0:MT_XSLT_Source xmlns:ns0="http://XYZ.com/gen">
  <Person>
    <FirstName>Anshul</FirstName>
    <LastName>Chowdhary</LastName>
    <Gender>Male</Gender>
    <Address>
      <Street>2nd Main</Street>
      <Houseno>83/b</Houseno>
      <City>Mysore</City>
    </Address>
  </Person>
</ns0:MT_XSLT_Source>
```

- 2.) Let the desired target be as shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:MT_XSLT_Target xmlns:ns1="http://XYZ.com/Test">
  <Title>Male</Title>
  <Name>Anshul Chowdhary</Name>
  <Street>83/b 2nd Main</Street>
  <City>Mysore</City>
</ns1:MT_XSLT_Target>
```

Now as we have the source and the target with us we can develop an XSLT mapping between them using any of the XML editors or even a note pad.

The XSL style sheet of the above transformation is as given below:

```
<?xml version='1.0' encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:ns0="http://XYZ.com/Gen"
xmlns:ns1="http://XYZ.com/Test">
  <xsl:template match="/">
    <ns1:MT_XSLT_Target>
      <Title>
        <xsl:value-of
select="ns0:MT_XSLT_Source/Person/Gender" />
      </Title>
      <Name>
        <xsl:value-of
select="concat(concat(ns0:MT_XSLT_Source/Person/FirstName,' '),
ns0:MT_XSLT_Source/Person/LastName)" />
      </Name>
      <Street>
        <xsl:value-of
select="concat(concat(ns0:MT_XSLT_Source/Person/Address/Houseno
,' '),
ns0:MT_XSLT_Source/Pers
on/Address/Street)" />
      </Street>
      <City>
        <xsl:value-of
select="ns0:MT_XSLT_Source/Person/Address/City" />
      </City>
    </ns1:MT_XSLT_Target>
  </xsl:template>
</xsl:stylesheet>
```

## Basic XSLT Tags

Now let's explain the above XSLT elaborately. Since an XSL style sheet is an XML document itself, it always begins with the XML declaration: `<?xml version="1.0" encoding="UTF-8"?>`. The next element, `<xsl:stylesheet>`, defines that this document is an XSLT style sheet document (along with the version number and XSLT namespace attributes). The `<xsl:template>` element defines a template. The `match="/"` attribute associates it with the root of the XML source document. The content inside `<xsl:template>` element defines some HTML content to be written as an output. The last two lines define the end of the template and of the style sheet respectively. Let's understand each tag used in an XSLT elaborately:

### 1) `<xsl:stylesheet>` or `<xsl:transform>` :

`<xsl:stylesheet>` or `<xsl:transform>` are the root elements that declare the document to be an XSL style sheet. Either of the two elements can be used as root elements as they are synonymous.

EG: `<xsl:stylesheet version="1.0" xmlns:xsl=http://www.w3.org/1999/XSL/Transform>`

The `xmlns:xsl="http://www.w3.org/1999/XSL/Transform"` points to the official W3C XSLT namespace. If you use this namespace, you must also include the attribute `version="1.0"`.

### 2) `<xsl:template>` :

An XSL style sheet contains one or more set of rules that are called templates. A template contains rules to apply when a specified node is matched. The "match" attribute is used to associate a template with an XML element or it can also be used to define a template for the entire XML document. The value of the match attribute is an XPath expression (i.e. `match="/"` defines the whole document).

### 3) `<xsl:value-of>` :

The `<xsl:value-of>` element is used to extract the value of a selected node. The value of the select attribute is an XPath expression. An XPath is used for defining parts of an XML document. An XPath expression works like navigating a file system where a forward slash (/) selects subdirectories.

### 4) `<xsl:for-each>` :

The `<xsl:for-each>` element is used to loop in XSLT. The value of the select attribute is an XPath expression.

For example in our above example if we had multiple person data at the source then we could have used `<xsl:for-each element>` as shown below:

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

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:ns0="http://XYZ.com/Gen"
xmlns:ns1="http://XYZ.com/Test">
  <xsl:template match="/">
    <ns1:MT_XSLT_Target>
      <xsl:for-each select="ns0:MT_XSLT_Source/Person">
        <Title>
          <xsl:value-of select="Gender"/>
        </Title>
        <Name>
          <xsl:value-of select="concat(concat(FirstName,' '),

```

```

                LastName)"/>
        </Name>
        <Street>
            <xsl:value-of select="concat(concat(Address/Houseno, '
        '),
                Address/Street)"/>
        </Street>
        <City>
            <xsl:value-of select="Address/City"/>
        </City>
    </xsl:for-each>
</ns1:MT_XSLT_Target>
</xsl:template>
</xsl:stylesheet>

```

We can also filter the output from the XML file by adding a criterion to the select attribute of `<xsl:for-each>` element.

EG: `<xsl:for-each select="ns0:MT_XSLT_Source/Person[FirstName='Anshul']">`

Valid filter operators are:

`=` (equal)

`!=` (not equal)

`&lt;` (less than)

`&gt;` (greater than)

5) **<xsl:sort>** :

The `<xsl:sort>` element is used to sort the output.

Example:

```

<?xml version='1.0' encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:ns0="http://XYZ.com/Gen"
xmlns:ns1="http://XYZ.com/Test">
    <xsl:template match="/">
        <ns1:MT_XSLT_Target>
            <xsl:for-each select="ns0:MT_XSLT_Source/Person">
                <xsl:sort select="Name">
                    <Title>
                        <xsl:value-of select="Gender"/>
                    </Title>
                    <Name>
                        <xsl:value-of
select="concat(concat(FirstName, ' '),

```

```

                LastName)"/>
            </Name>
            <Street>
                <xsl:value-of
                select="concat(concat(Address/Houseno,' '),
                Address/Street)"/>
            </Street>
            <City>
                <xsl:value-of select="Address/City"/>
            </City>
        </xsl:sort>
    </xsl:for-each>
</ns1:MT_XSLT_Target>
</xsl:template>
</xsl:stylesheet>

```

The select attribute indicates what XML element to sort on. In the above example it will display the output based upon sorting the "Names".

#### 6) <xsl:if> :

The <xsl:if> element is used to put a conditional test against the content of the XML file. The value of the required test attribute contains the expression to be evaluated.

**Syntax:** <xsl:if test="expression">

</xsl:if>

Example:

```

<?xml version='1.0' encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:ns0="http://XYZ.com/Gen"
xmlns:ns1="http://XYZ.com/Test">
  <xsl:template match="/">
    <ns1:MT_XSLT_Target>
      <xsl:for-each select="ns0:MT_XSLT_Source/Person">
        <xsl:if test="Gender =Male">
          <Title>
            <xsl:value-of select="Gender"/>
          </Title>
          <Name>
            <xsl:value-of
            select="concat(concat(FirstName,' '),
            LastName)"/>
          </Name>
          <Street>
            <xsl:value-of

```

```

        select="concat(concat(Address/Houseno, ' '),
                        Address/Street)"/>
        </Street>
        <City>
            <xsl:value-of select="Address/City"/>
        </City>
    </xsl:if>
</xsl:for-each>
</ns1:MT_XSLT_Target>
</xsl:template>
</xsl:stylesheet>

```

The above code will only output those person details which have "male" as gender .

### 7) <xsl:choose> :

The <xsl:choose> element is used to handle condition based tests. Multiple conditions are expressed with the help of <xsl:when> and <xsl:otherwise> elements.

#### Syntax/EG:

```

<xsl:choose>
  <xsl:when test=" Gender =Male ">
    ... some processing logic inside ...
  </xsl:when>
  <xsl:otherwise>
    ... some processing logic inside....
  </xsl:otherwise>
</xsl:choose>

```

Choose condition will come just above the element in the XSL where the condition needs to be implied.

### 8) <xsl:apply-templates> :

The <xsl:apply-templates> element applies a template to the current element or to the current element's child nodes. If we add a select attribute to the <xsl:apply-templates> element it will process only the child element that matches the value of the attribute. We can use the select attribute to specify the order in which the child nodes are processed.



## XPATH Functions in XSLT Mapping:

This explains the use of various XPATH functions with their syntaxes:

### 1) substring()

This Function is used to extract some specified portion from the original string. It extracts the specified number of characters from a string.

**Syntax:** substring("ANSHUL CHOWDHARY",1,6)

Output: "ANSHUL"

### 2) translate()

The translate function takes the input string in the value argument of the syntax as shown below and substitutes all occurrences of a string specified in the string1 argument with that mentioned in string2 argument.

**Syntax:** translate("Anshul chowdhary","abcdefghijklmnopqrstuvwxyz",  
 ABCDEFGHIJKLMNOPQRSTUVWXYZ)

Output:"ANSHUL CHOWDHARY"

### 3) string()

The string function converts the input to a string.

**Syntax:** string("Anshul chowdhary")

### 4) concat()

The concat function takes all input arguments individually and concatenates them together in the specified order.

**Syntax:** concat("anshul","chowdhary")

Output:"anshulchowdhary"

### 5) sum()

The sum function converts PCDATA text to a numeric value

**Syntax:** sum(p2:marks/score)

### 6) count()

This function is used to count the nodes

**Syntax:** count(p2:marks/subjects) .

The use of the above XPATH Functions is explained in an example below, but before going into that we should know how to use XSLT Mapping in PI.

## How to Use an XSLT Mapping in PI

**Basic Steps-** There are some basics steps required for using XSLT mapping in PI. Those steps are -

**STEP 1:** Create the source and target data type.

**STEP 2:** Create the Source and the Target Message types.

**STEP 3:** Create Inbound and Outbound Service interfaces.

**STEP 4:** XSLT Mapping does not require creation of Message mapping as the .XSL file is directly imported to the Operations Mapping.

**STEP 5:** Create a .XSL file which contains the logic for converting source data type to target data type.

**STEP 6:** Zip the developed .xsl file and import it into Enterprise Services Builder under Imported Archives.

**STEP 7:** In Operation Mapping choose mapping program as XSL and specify this zip program. (When one chooses the Type as XSL, in search help all XSL Mapping programs that are imported under Imported Archives of the particular namespace gets listed for selection)

**STEP 8:** Test the mapping program imported by moving to the Test tab.

Based upon the above mentioned steps, a few scenarios have been configured in PI as shown below. While explaining the examples it has been assumed that the user has basic knowledge of interface creation in PI 7.1.

## Example 1

Creating flat structure from a complex message.

The source structure is as shown in the outbound datatype below.

The screenshot shows the SAP Data Type Designer interface for the data type **DT\_XSLT\_Outbound**. The status is **Active** and the display language is **English**. The classification is **Free-Style Data Type**. The XSD definition is as follows:

Name	Category	Type	Occurrence	Default	Details	Business C...	Description
DT_XSLT_Outbound	Complex Type						
Person	Element		1..unbounded				
FirstName	Element	xsd:string	1				
LastName	Element	xsd:string	1				
Gender	Element	xsd:string	1				
Address	Element		1				
Street	Element	xsd:string	1				
Houseno	Element	xsd:string	1				
City	Element	xsd:string	1				

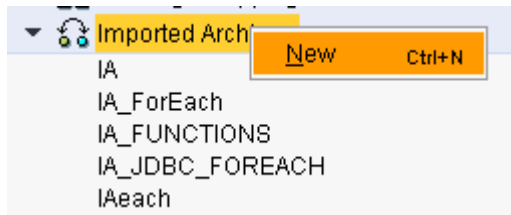
The inbound datatype is as mentioned below. This is the desired target structure.

The screenshot shows the SAP Data Type Designer interface for the data type **DT\_XSLT\_Inbound**. The status is **Active** and the display language is **English**. The classification is **Free-Style Data Type**. The XSD definition is as follows:

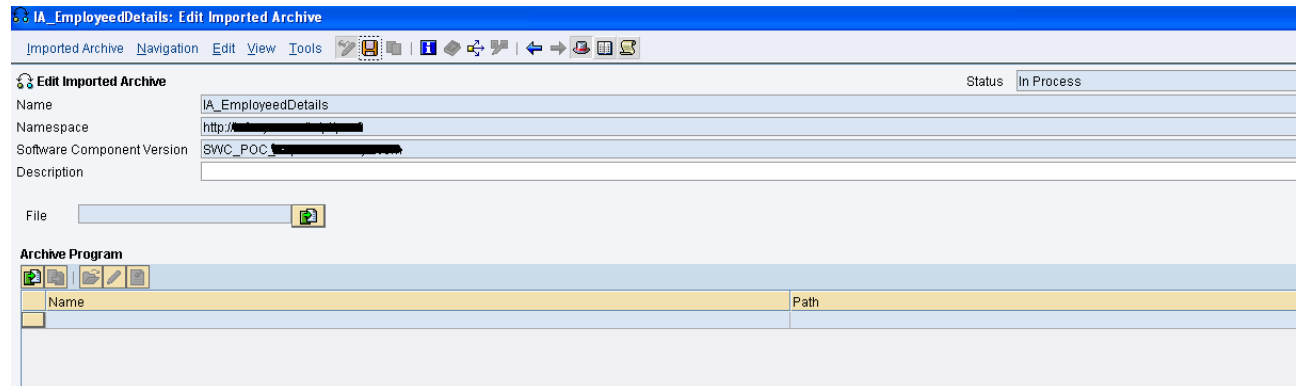
Name	Category	Type	Occurrence	Default	Details	Business C...	Description
DT_XSLT_Inbound	Complex Type						
Details	Element		1..unbounded				
Title	Element	xsd:string	1				
Name	Element	xsd:string	1				
Street	Element	xsd:string	1				

For the above two data types, prepare the message types, service interfaces etc. Message Mapping will not exist for interfaces which use XSLT mapping. The XSLT mapping is required to be specified in Operation Mapping. To achieve this, .XSL file is transported to Imported archives in the form of a ZIP file as shown below.

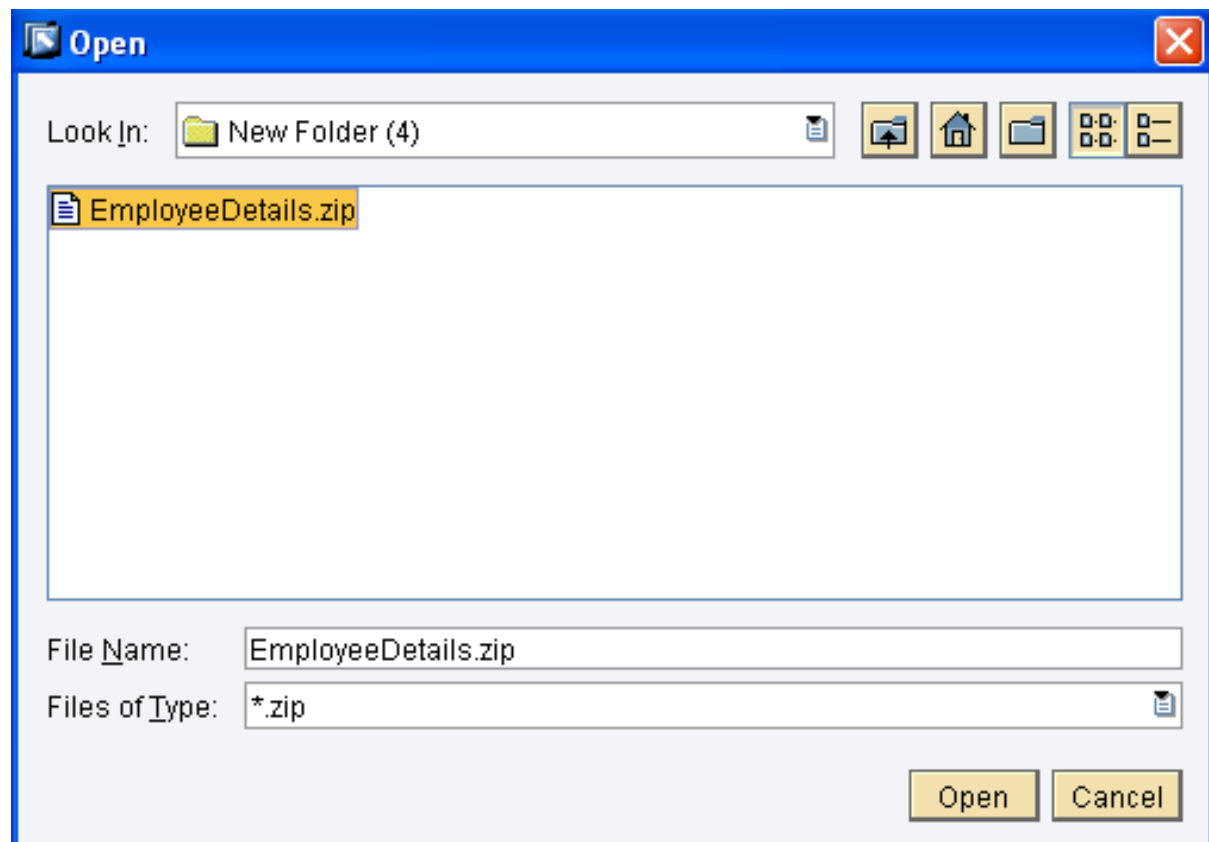
In the ESR go to the desired namespace and right click on Imported Archive. Select New as shown below.



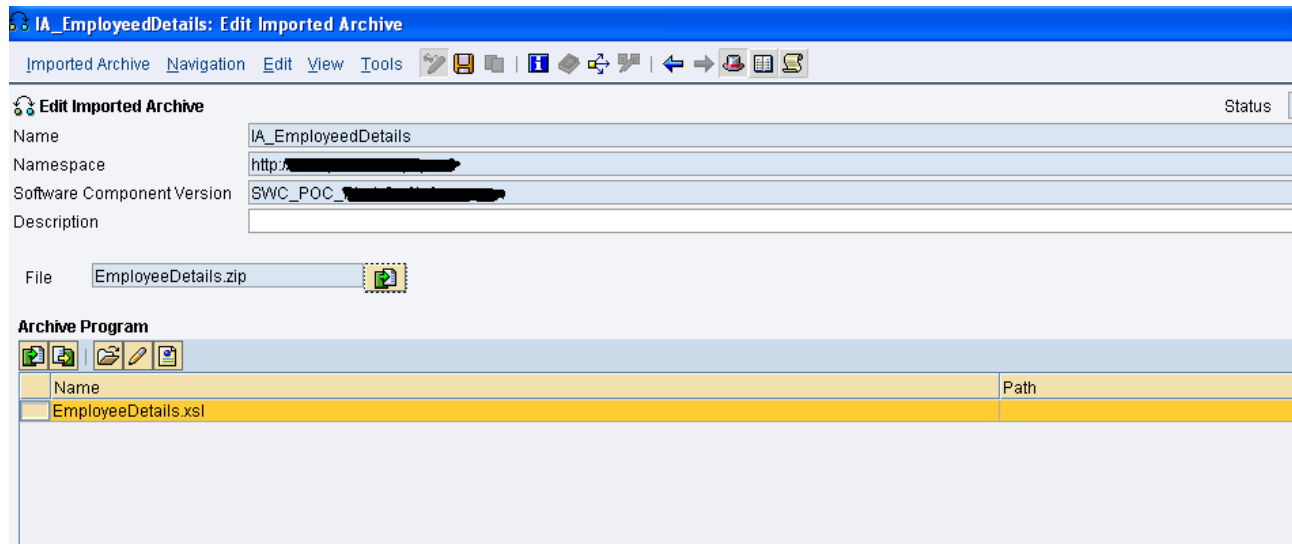
The below screen pops up when the name of the xsl file is entered after clicking on New. The file is imported by clicking in the Import Archive button as shown below.



When Import Archive button is pressed, a browser window open up from where the desired xsl mapping is chosen.



Once the mapping is selected, it comes in ESR as shown below.



The xsl mapping is as given below.

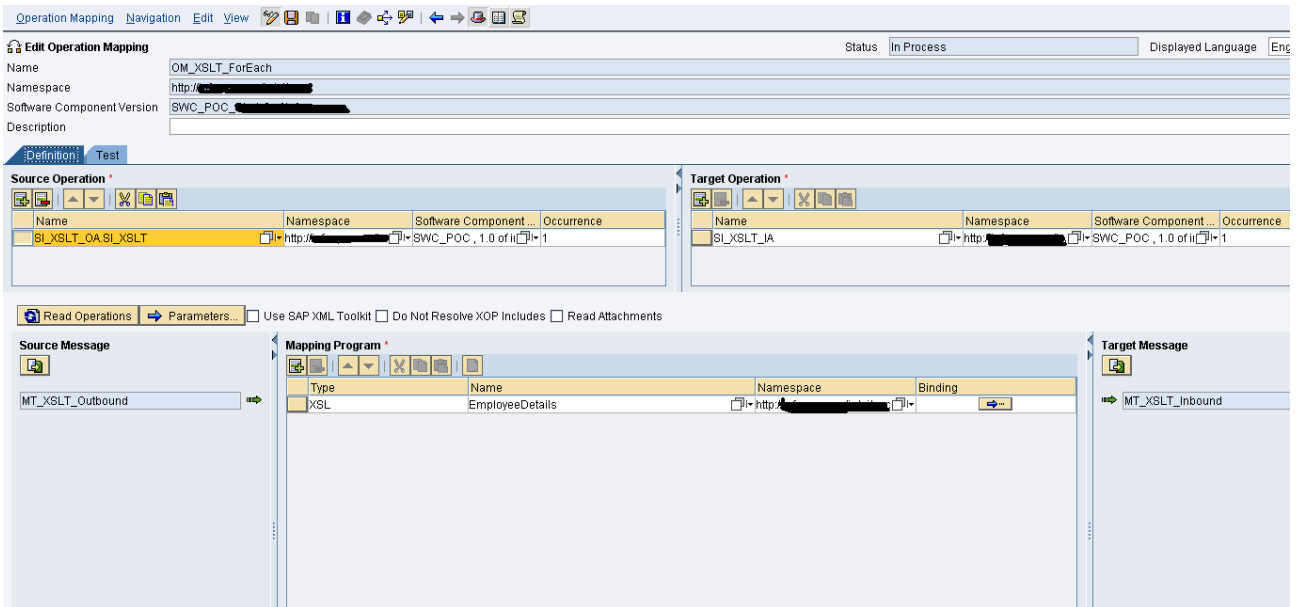
```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:ns0="
http://XYZ.com/gen1" xmlns:ns1=" http://XYZ.com/gen 2">
  <xsl:template match="/">
    <ns1:MT_XSLT_Inbound>
      <xsl:for-each select="ns0:MT_XSLT_Outbound/Person">
        <Details>
          <Gender>
            <xsl:value-of select="Gender" />
          </Gender>
          <Name>
            <xsl:value-of select="concat(concat(FirstName, '
'), LastName)" />
          </Name>
          <Street>
            <xsl:value-of select="Address/Street" />
          </Street>
          <HouseAddress>
            <xsl:value-of
select="concat(concat(Address/Houseno, ' '), Address/City)" />
          </HouseAddress>
        </Details>
      </for-each>
    </ns1:MT_XSLT_Inbound>
  </template>
</xsl:stylesheet>
```

```

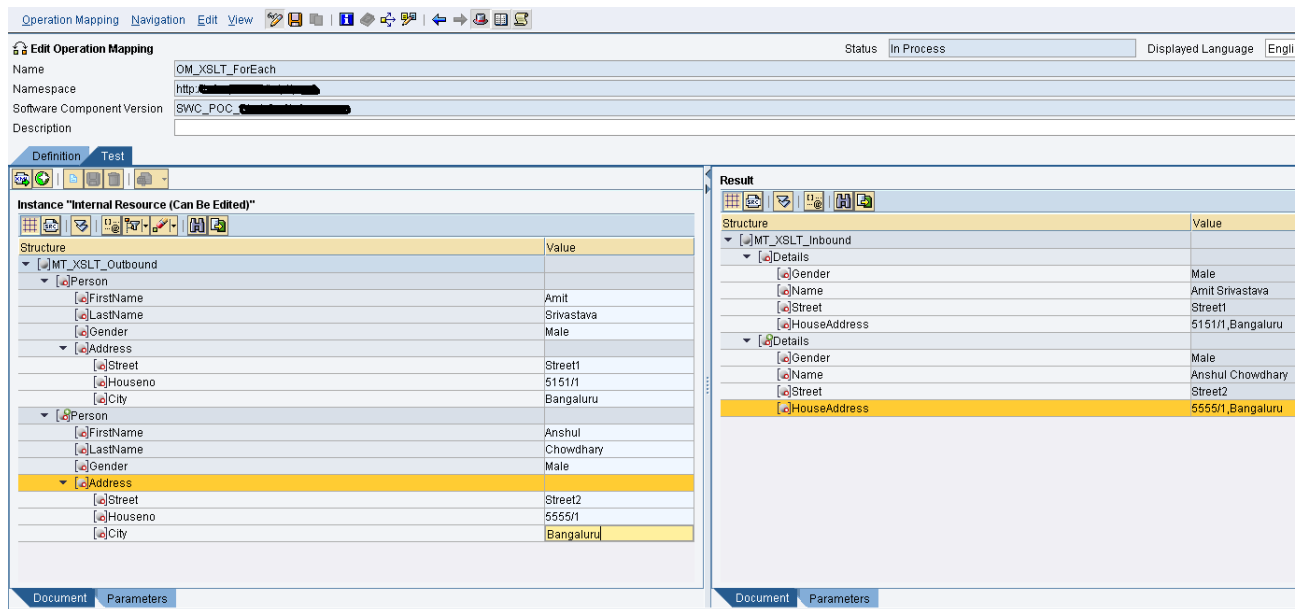
</xsl:for-each>
</ns1:MT_XSLT_Inbound>
</xsl:template>
</xsl:stylesheet>

```

In Operation Mapping select XSL as type under Mapping Program and press the search button in the Name section. The entire XSL mapping imported gets listed from where the desired mapping is chosen.



The desired output of the interface is as shown in the below screen.



## Example 2

There is one common problem in JDBC scenarios that whenever SP name changes at target side , we need to redo our graphical message mapping . So, in order to get rid of that we can use XSLT mapping. You just need to change the SP name in the XSL sheet . The scenario is as mentioned below:

The below given are the outbound and the inbound datatypes.

The screenshot shows the 'Display Data Type' window for 'DT\_XSLT\_JDBC\_Outbound'. The Name is 'DT\_XSLT\_JDBC\_Outbound', Namespace is 'http://...', and Software Component Version is 'SWC\_POC , 1.0...'. The Classification is 'Free-Style Data Type'. The 'Type Definition' tab is active, showing the XSD structure in a table:

Name	Category	Type	Occurrence	Defa...	Det...	Busi...	De...	UI...
DT_XSLT_JDBC_Outbound	Complex Type							
Revenue	Element		1..unbounded					
CorpDb_Oppt_Id	Element	xsd:string	1					
Emp_Created	Element	xsd:string	1					
Emp_Modified	Element	xsd:string	0..1					


The screenshot shows the 'Display Data Type' window for 'DT\_XSLT\_JDBC\_Inbound'. The Name is 'DT\_XSLT\_JDBC\_Inbound', Namespace is 'http://...', and Software Component Version is 'SWC\_SAP\_POC , 1.0...'. The Classification is 'Free-Style Data Type'. The 'Type Definition' tab is active, showing the XSD structure in a table:

Name	Category	Type	Occurrence	Default	Details	Business...	Descrip
DT_XSLT_JDBC_Inbound	Complex Type						
StatementName	Element		1..unbounde				
spUploadXSLT	Element		1				
action	Attribute	xsd:string	required				
intOpportunityId	Element	xsd:integer	1				
type	Attribute	xsd:integer	required				
btEmpNoCreated	Element	xsd:string	1				
type	Attribute	xsd:string	required				
btEmpNoModified	Element	xsd:string	1				
type	Attribute	xsd:string	required				






The XSLT mapping code is imported as mentioned in the above example.

**Edit Imported Archive** Status **Active** Display Language

Name: IA\_JDBC\_FOREACH  
Namespace: http://[REDACTED]  
Software Component Version: SWC\_POC\_PI, 1.0 [REDACTED]  
Description:






File: JDBC1.zip 

**Archive Program**

Name	Path
JDBC1.xsl	

**Open** X

Look in: Desktop     

- EP
- pics
- EP.zip
- FOREACH.zip
- JDBC.zip
- JDBC1.zip**
- new.zip
- qq.zip
- qqeach.zip
- rr.zip
- self.zip

File name: JDBC1.zip  
Files of type: \*.zip

**Open** **Cancel**



The xsl mapping used is as given below.

```
<?xml version="1.0" encoding="UTF-8" ?>
  <xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:ns0="
http://XYZ.com/gen 1" xmlns:ns1=" http://XYZ.com/gen ">
  <xsl:template match="/">
    <ns1:MT_XSLT_JDBC_Inbound>
      <xsl:for-each select="ns0:MT_XSLT_JDBC_Outbound/Revenue">
        <StatementName>
          <spUploadXSLT>
            <xsl:attribute name="action">EXECUTE</xsl:attribute>
            <intOpportunityId>
              <xsl:attribute name="type">string</xsl:attribute>
              <xsl:value-of select="CorpDb_Oppt_Id" />
            </intOpportunityId>
            <txtEmpNoCreated>
              <xsl:attribute name="type">string</xsl:attribute>
              <xsl:value-of select="Emp_Created" />
            </txtEmpNoCreated>
            <txtEmpNoModified>
              <xsl:attribute name="type">string</xsl:attribute>
              <xsl:value-of select="Emp_Modified" />
            </txtEmpNoModified>
          </spUploadXSLT>
        </StatementName>
      </xsl:for-each>
    </ns1:MT_XSLT_JDBC_Inbound>
  </xsl:template>
</xsl:stylesheet>
```

Then the mapping is used in Operation Mapping to get the desired result.

Operation Mapping Edit View

Display Operation Mapping Status Active Display Language E

Name OM\_XSLT\_JDBC

Namespace http://infosSWC\_PO1

Software Component Version SWC\_POC\_PI, 1.0

Description

Definition Test

**Source Operation \***

Name	Namesp...	Softwar...	Occurr...
SI_XSLT_JDBC_OA	http://infosSWC_PO1		

**Target Operation \***

Name	Namesp...	Softw...
SI_XSLT_JDBC_IA	http://infosSWC...	

Read Operations Parameters...  Use SAP XML Toolkit  Do Not Resolve XOP Includes  Read Attachments

**Source Message**

MT\_XSLT\_JDBC\_O

**Mapping Program \***

Type	Name	Namesp...	Binding
XSL	JDBC1	http://infosys	

**Target Message**

MT\_XSLT...

The desired result is achieved as shown below.

**Display Operation Mapping** Status: Active Display Language: E

Name: OM\_XSLT\_JDBC  
 Namespace: http://...  
 Software Component Version: SWC\_POC\_PI, 1.0 of ...  
 Description: ...

Definition Test

Instance "Internal Resource (Can Be Edited)"

Structure	Value
MT_XSLT_JDBC_Outbound	
Revenue	
CorpDb_Oppt_Id	12
Emp_Created	97244
Emp_Modified	Amit
Revenue	
CorpDb_Oppt_Id	13
Emp_Created	972445
Emp_Modified	Amit Sri

Result

Structure	Value
MT_XSLT_JDBC_Inbound	
StatementName	
spUploadXSLT	
action	EXECUTE
intOpportunityId	12
btEmpNoCreated	97244
btEmpNoModified	Amit
StatementName	
spUploadXSLT	
action	EXECUTE
intOpportunityId	13
type	string
btEmpNoCreated	972445
type	string
btEmpNoModified	Amit Sri
type	string

Document Parameters

## References

[XSLT Tutorial - w3schools.com](http://w3schools.com)

[Blog: xpath functions in xslt mapping](#)

[XSL Transformations \(XSLT\) Version 2.0 - w3.org](http://w3.org)

[XSLT Mapping for SAP PI 7.1 on help.sap.com](http://help.sap.com)

## Disclaimer and Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.