ADEPTIA

Adeptia Suite 6.1 Data Mapper

Recommendation Guide For Saxon XSL Transformer

Version 1.0
Release Date March 4, 2014

Adeptia Inc.
343 West Erie, Suite 440
Chicago, IL 60654, USA
Phone: (312) 229-1727 x111
Fax: (312) 229-1736

DOCUMENT INFORMATION

Adeptia Suite

Adeptia Suite 6.1 Data Mapper Recommendation Guide for Saxon XSL Transformer

Adeptia Suite Version 6.1

Printed January 2013

Printed in USA

Adeptia Support Information

For support queries, please contact us at support@adeptia.com.
Access the Adeptia Web site at the following URL:

www.adeptia.com

Copyright

Trademarks

Confidentiality

Disclaimer

TABLE OF CONTENTS

CONVENTIONS

The following tables list the various conventions used in this documentation. We follow these conventions to help you quickly and easily identify particular elements, processes, and names that occur frequently in documents.

Typographical conventions

This guide uses the following typographical conventions:

Convention	Description
Bold Text	Indicates one of the following: ▪ Screen element ▪ A file or folder name ▪ A control in an application's user interface ▪ Important information
Bold Italics Text	File or folder path that you need to enter as per your system configuration
`Monospaced Text`	Indicates the code that you need to enter as it is
`Monospaced Italics Text`	Indicates the code that you need to enter as per your system configuration/specification
Hyperlink	Indicates a link to a website or web material

Graphical conventions

This guide uses the following graphical conventions:

Convention	Description
ℹ	Indicates additional information that may be of interest to the reader

Contacts/Reporting problems

These sections present contact information for a variety of situations.

Sales

In case of any sales queries, please contact us at sales@adeptia.com.

Support

For support queries, please contact us at support@adeptia.com.

Latest updates and information

For the latest updates and information, please visit us at www.adeptia.com.

Adeptia Web site

Access the Adeptia Web site at the following URL:

www.adeptia.com

INTRODUCTION

This document explains the usage of DB query function in the data mapper that you can use to fetch data from multiple columns. This document also lists out some of the guidelines that you need to follow while using Saxon parser in the Adeptia Suite 6.1.

Target Audience

- User who want to use the DB query function in Adeptia Suite 6.1
- Users who want to use Saxon parser bundled with Adeptia Suite 6.1

Pre-requisites

- You need to install Adeptia Suite 6.1

Topic Covered

This guide covers the following topics:

- Fetch data from multiple columns using DB query function
- Guidelines with Saxon XSL Transformer
- Using Tokenized Function Using Saxon XSL Transformer

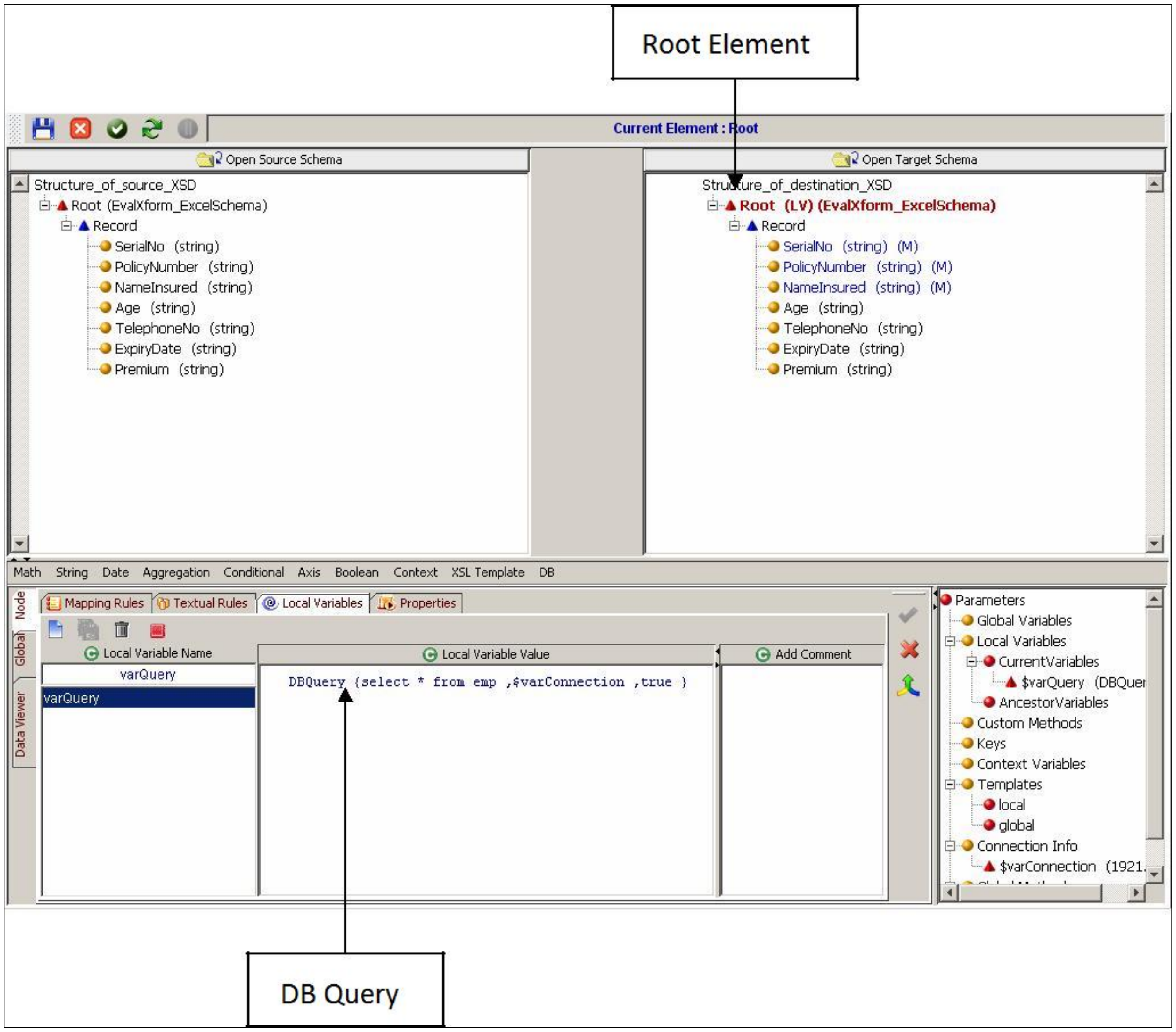
FETCH DATA FROM MULTIPLE COLUMNS USING DB QUERY FUNCTION

If you want to fetch data from multiple columns then, you can use the DB query function in the data mapper to do that. Using DB query you can fetch data from multiple columns of a single database table or data from different columns of different database tables. Follow the steps mentioned below to fetch data from multiple columns using the DB query function.

Steps to fetch data from multiple columns by using the DB query function

Below are the steps that you need to follow to use the DB query function:

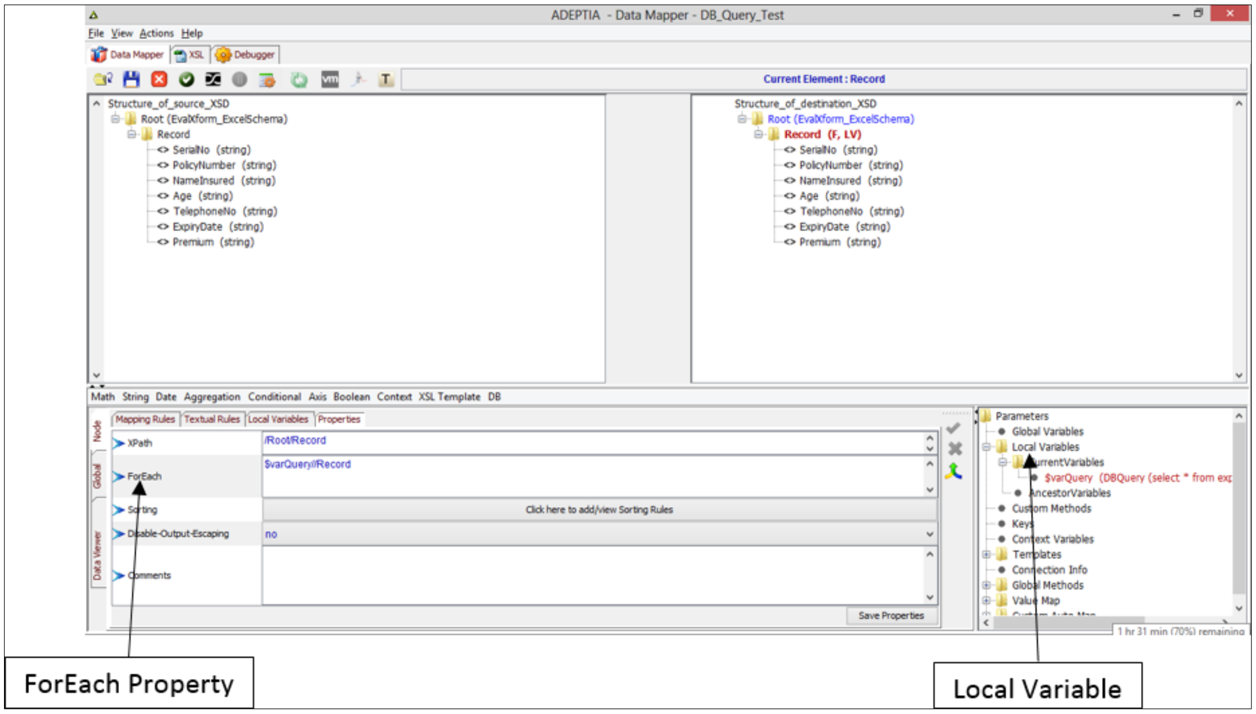
1. Load source and target schema in the Data Mapper.
2. Create a Connection Info variable for the database from where you want to fetch your data.
3. Select the Root element of the target schema and create a Local Variable which will contain the value of the DB Query (see Figure 1).



```
DBQuery (select * from emp ,$varConnection ,true )
```

Figure 1: Root Element and DB Query in Data Mapper

4. Click the **Save** button to save the local variable.
5. Click the Parent element (Record) from the target panel and then click the **Properties** tab.
6. Click on the **For Each** field and double click the local variable that you have created to store the value of the DB query (see Figure 2).
   - You need to append it with `//Record` when you are using Saxon XSL Transformer
   - You need to append it with `/Record` when you are using the old Xalan XSL Transformer



XPath: /Root/Record
ForEach: $varQuery//Record
Sorting
Disable-Output-Escaping: no
Comments

Figure 2: ForEach Property and Local Variable in the Data Mapper

7. Click the **Save Properties** button to save the **ForEach** property that you have just created.
8. To map the output of the DB query, select the target element to which you want to map the output and then click the **Textual Rules** tab.
9. In the **Textual Rules** tab, enter the name of the column of the database table whose value you want to map to the selected target element (see Figure 3).

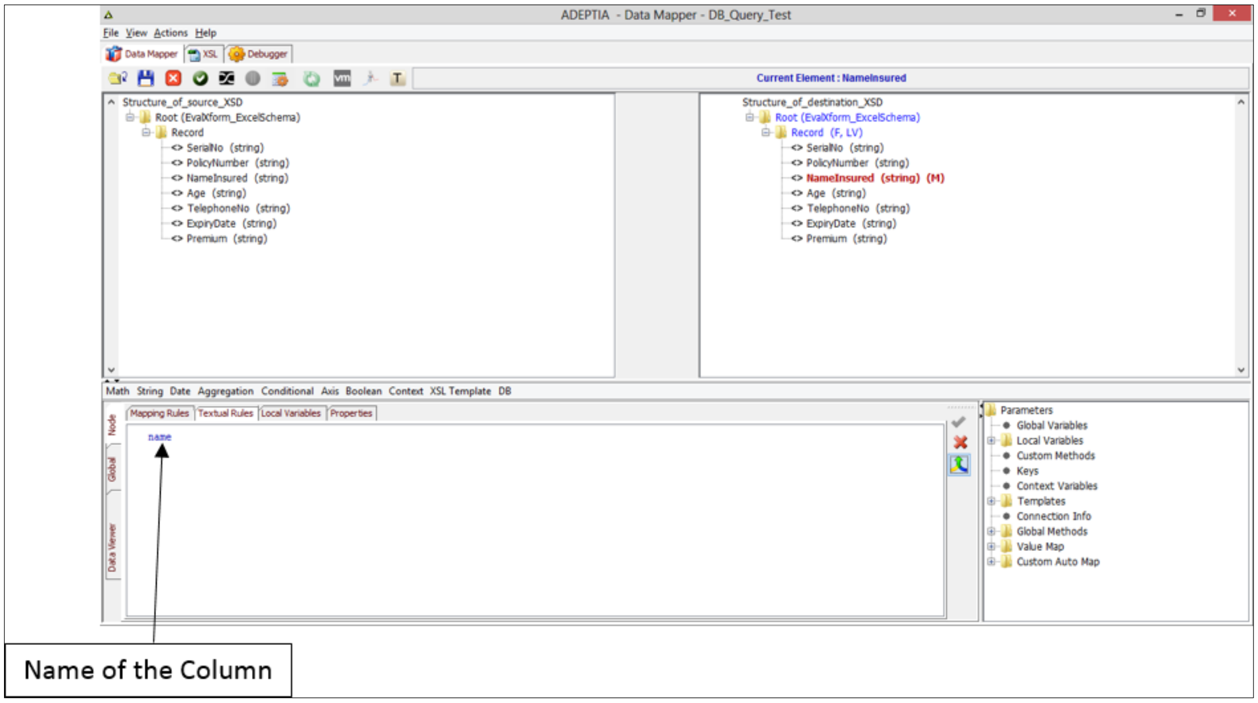


Figure 3: Name of the Column in Textual Rules Tab

10. Click the **Apply Mapping** button to apply the mapping that you have just created.

> You can map other target elements with other columns.

GUIDELINES FOR CHANGING THE XSL TRANSFORMER FROM XALAN TO SAXON

The Adeptia Suite 6.1 comes with a new XSL transformer, Saxon. We recommend you to use Saxon transformer to transform all you data mapping activities. However, you need to take care of the following points when you are using Saxon transformer:

- You need to perform the **edit** and **save** operation on the **global, group** and **local** XSL templates of the **Templates** (see Figure 4) group just once in the following scenarios:
    - When you are using an existing template (with no java function), that you have created using a previous version of Adeptia Suite, to create a new mapping activity.
    - When you are changing the XSL transformer type from Xalan to Saxon for an existing mapping activity that you have created using a previous version of Adeptia Suite.

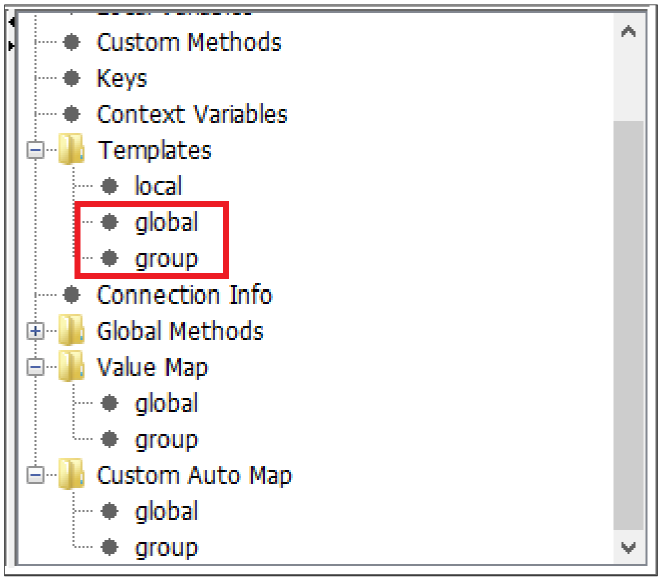


Figure 4: global and group Templates

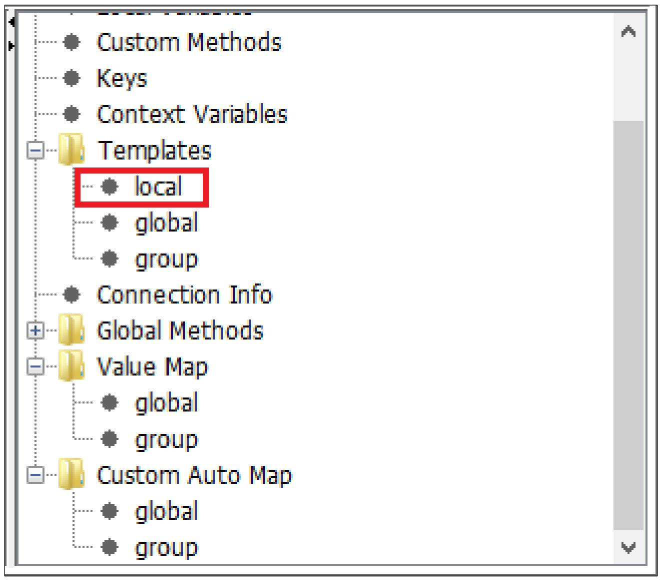
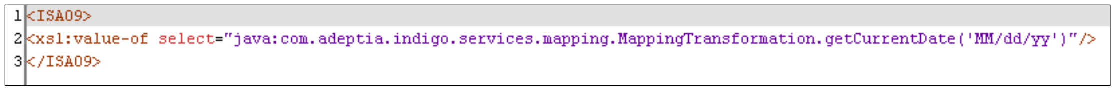


Figure 5: local Templates

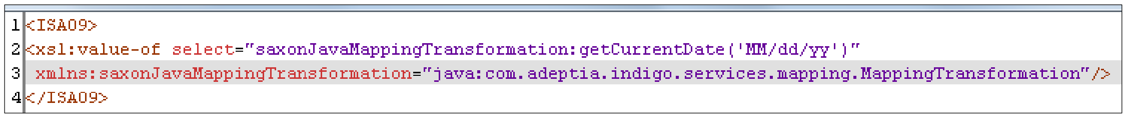
- If you use the **Custom XSL Before** and **Custom XSL After** functions in Xalan then, while moving over to the new XSL transformer (Saxon) you need to edit the code of these functions as per the new XSL transformer.
  For Example: If you use a Java method call in **Custom XSL Before** or **XSL Template** function then your Xalan code would be as in the Figure 6. When you change your XSL Transformer type to Saxon then, you need to change the code of the function as per Saxon (see Figure 7).



```
<ISA09>
<xsl:value-of select="java:com.adeptia.indigo.services.mapping.MappingTransformation.getCurrentDate('MM/dd/yy')"/>
</ISA09>
```

Figure 6: Custom XSL Before in Xalan

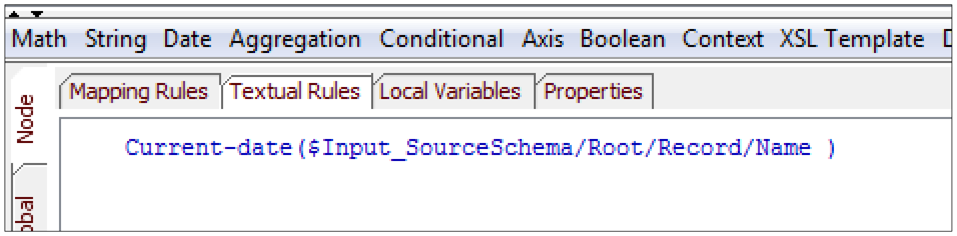
After changing the XSL transformer you need to edit and modify the **Custom XSL Before** function as per Saxon:



```
<ISA09>
<xsl:value-of select="saxonJavaMappingTransformation:getCurrentDate('MM/dd/yy')"
 xmlns:saxonJavaMappingTransformation="java:com.adeptia.indigo.services.mapping.MappingTransformation"/>
</ISA09>
```

Figure 7: Custom XSL Before in Saxon

- In case of using Java method functions in Data Mapper like date format and date difference etc. and using XPath as a parameter. If you do not apply the **ForEach** condition on the target node and the source input XML contains multiple records then, in such case you need to specify the record number in the Xpath being used for such functions. For example:

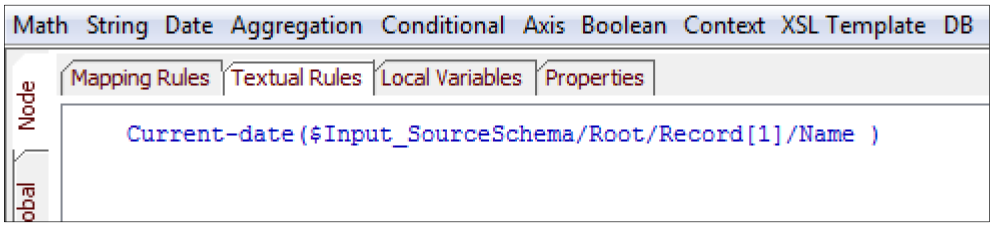


```
Current-date($Input_SourceSchema/Root/Record/Name )
```

Figure 8: Target Record Not Specified (Xalan)

Figure 8 shows a case where **ForEach** condition is not applied on the target node in Xalan XSL transformer. In this case, if you do not specify a record then, Xalan will automatically pick up a value from the first record.

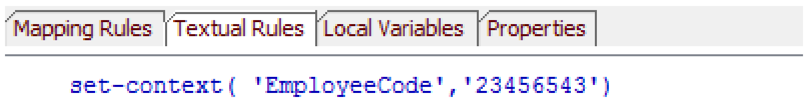
But in case of Saxon XSL transformer, it would not work if you have not applied the **ForEach** condition on the target node. Instead you need to specify the record from where the XSL transformer need to pick up the data from in Saxon (see Figure 9).



```
Current-date($Input_SourceSchema/Root/Record[1]/Name )
```

Figure 9: Target Node Specified (Saxon)

- If you are using **set-context** or **get-context** function then second parameter always has to be either string value or an xPath. If you need to pass the numeric values as second argument then enclose it inside single quote (see Figure 10).



```
set-context( 'EmployeeCode','23456543')
```

Figure 10: Set Context (Saxon)

- If you need to pass **negative constant** to the custom method whose parameter data type is **int** then, the following changes needs to be done:

1. In the Data Mapper, go to **Actions →Namespace → Add Namespace** (see Figure 11).

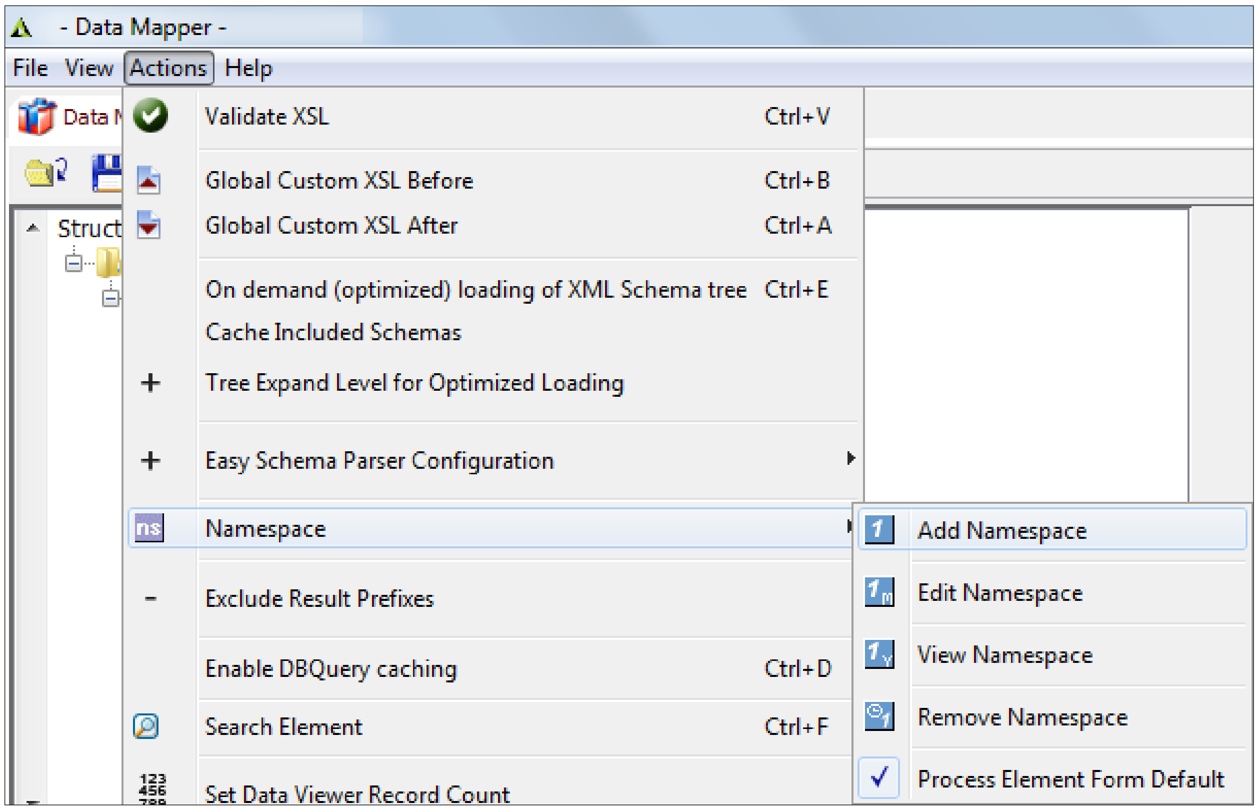
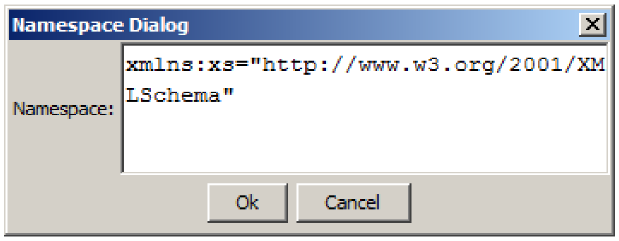

- Data Mapper -
File View Actions Help
Data M
Struct
Validate XSL Ctrl+V
Global Custom XSL Before Ctrl+B
Global Custom XSL After Ctrl+A
On demand (optimized) loading of XML Schema tree Ctrl+E
Cache Included Schemas
Tree Expand Level for Optimized Loading
Easy Schema Parser Configuration
Namespace
Exclude Result Prefixes
Enable DBQuery caching Ctrl+D
Search Element Ctrl+F
Set Data Viewer Record Count
Add Namespace
Edit Namespace
View Namespace
Remove Namespace
Process Element Form Default

Figure 11: Adding a Namespace

2. Here add namespace http://www.w3.org/2001/XMLSchema and bind it with a namespace prefix **"xs"** (see Figure 12).



```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

Figure 12: Binding a Namespace Prefix

3. Now to pass a negative constant to a custom method, you need to pass it like this xs:integer(negative constant). For example: java:DTNUtils.getPreviousDate($Input_wsSchema_Test/request/Map/key ,'yyyyMMdd' , xs:integer(-3)).

4. Click on the ✓ button to save the rule.

USING TOKENIZED FUNCTION USING SAXON XSL TRANSFORMER

Below are the steps that you need to follow to use Saxon XSL transformer in Tokenized functions.

Steps to use Saxon XSL transformer in Tokenized Functions

In case you are using the XSL function tokenize in Xalan then, while moving to Saxon, you will have to make the following changes:

1. Load source and target schema in the Data Mapper.
2. In the Data Mapper, go to **Actions →Namespace → Add Namespace**. This action will add a namespace in the Data Mapper (see Figure 13).

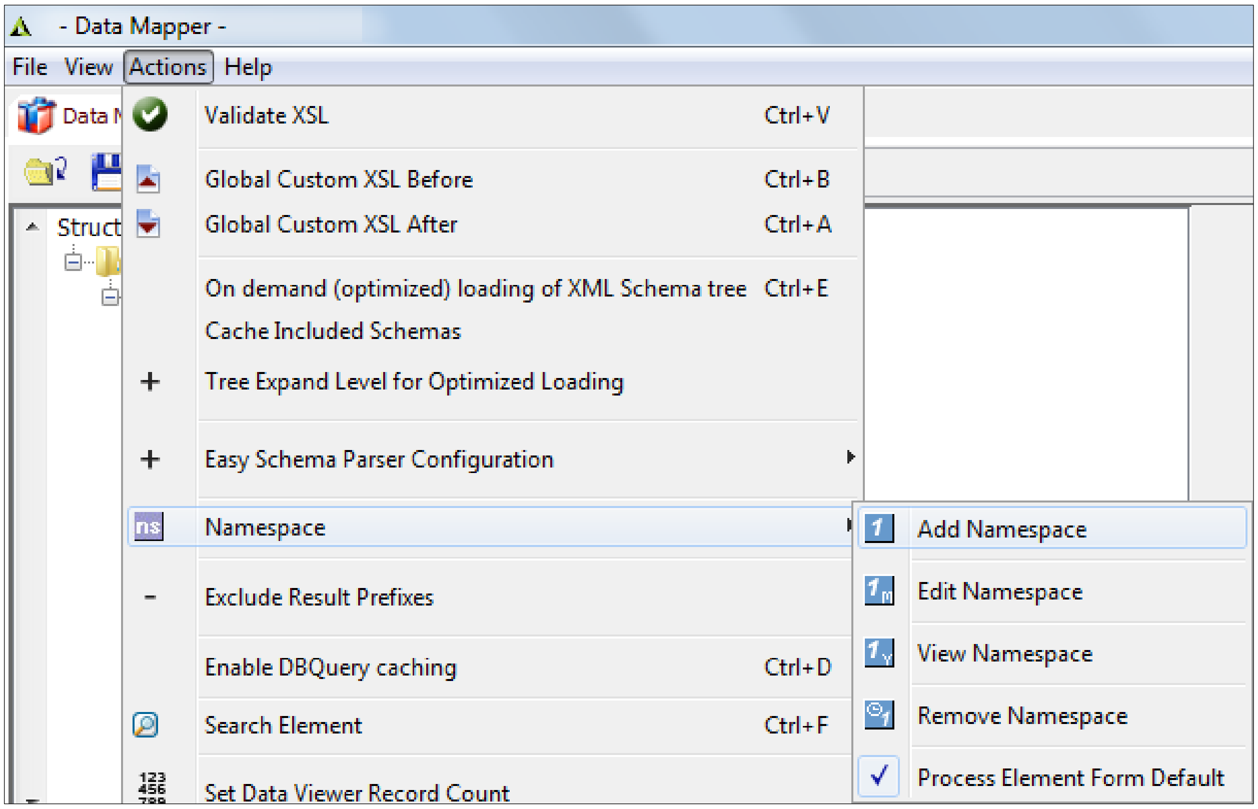
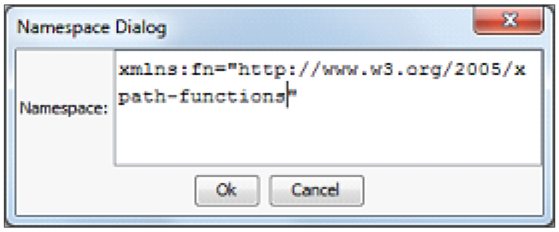

- Data Mapper -
File
View
Actions
Help
Data M
Struct
Validate XSL Ctrl+V
Global Custom XSL Before Ctrl+B
Global Custom XSL After Ctrl+A
On demand (optimized) loading of XML Schema tree Ctrl+E
Cache Included Schemas
Tree Expand Level for Optimized Loading
Easy Schema Parser Configuration
Namespace
Exclude Result Prefixes
Enable DBQuery caching Ctrl+D
Search Element Ctrl+F
Set Data Viewer Record Count
Add Namespace
Edit Namespace
View Namespace
Remove Namespace
Process Element Form Default

Figure 13: Adding a Namespace

3. After adding a namespace, bind it with a namespace prefix (see Figure 14).

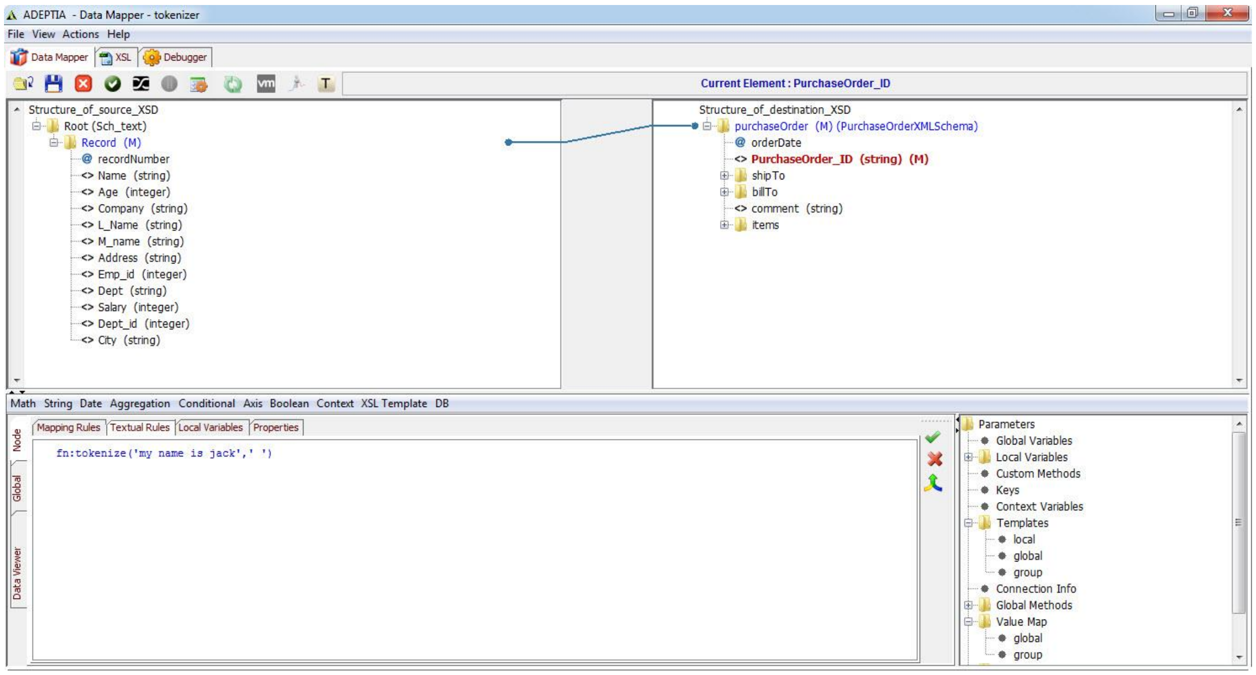


```
xmlns:fn="http://www.w3.org/2005/xpath-functions"
```

Figure 14: Binding a Namespace Prefix

4. In case you are using this function in Saxon then, append the namespace prefix, of the URL defined (`xmlns:fn="http://www.w3.org/2005/xpath-functions"`) in the above step, before the tokenize function. For example - `fn:tokenize()`.
5. In case you are using this function with Xalan then, you need to replace the mapping rules with the above syntax.
   For example - In Xalan: `str:tokenize()`.

6. Select a target node and apply this rule to it (see Figure 15).



```
fn:tokenize('my name is jack',' ')
```

Figure 15: Applying the Tokenized Rule at Target Node

7. Click on the ✓ button to save the rule.

During migration, there can be issues when you are using certain EXSLT functions in Xalan. In such cases, please refer to the http://saxon.sourceforge.net/saxon6.5.3/extensions.html#after link to know about the implementation of those functions in Saxon and make the necessary changes in the mapping.