

Java XML Converter (JXC)

Manual

Contents

Overview	3
Example XML Schemas	4
The Orders schema	4
The Order Summary schema	4
Using the JXC API in your Java project	5
Filtering XML	5
Mapping XML	6
Using the GUI tools	8
Mapping XML	8
Pre Filtering XML	11
Using the Expression Builder	13
XML namespace support	14
Schema recursion	15
Known limitations	15
XPath 2.0 functions	15

Overview

JXC is an open source Java project released under the LGPLv3 license for converting XML from one form to another.

Please note that JXC is distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. Please refer to the licence for details.

It can be used as a tool to:

- Filter and map XML documents with XPath expressions using a GUI
- Export mappings created using the GUI as XSL files

Or as a library JAR to:

- Filter XML using XPath expressions in your Java application
- Map XML from one schema to another in your Java application

JXC therefore offers the capability for XML filtering and mapping to be dynamically driven at run time in your Java application.

To run the GUI tools a Java Runtime Environment (JRE) version 1.6 or above must be installed.

Example XML Schemas

This section describes the example XML schemas used to demonstrate and describe JXC functionality.

The Orders schema

Figure 1 illustrates an XML document constrained by the example Orders schema (Orders.xsd).

```
<?xml version="1.0" encoding="UTF-8"?>
<orders>
  <order orderId="1">
    <customerId>1</customerId>
    <orderRef>ORD00001</orderRef>
    <customerName>Henry</customerName>
    <orderDate>2010-01-10</orderDate>
    <status>Dispatched</status>
    <billingAddress>
      <houseNumber>12</houseNumber>
      <street>Wagtail Avenue</street>
      <town>Peterborough</town>
      <area>Cambridgeshire</area>
      <country>United Kingdom</country>
      <postCode>NG99 9YY</postCode>
    </billingAddress>
    <item orderItemId="1">
      <productId>1</productId>
      <productName>Deluxe Dog Basket</productName>
      <quantity>1</quantity>
      <price>49.99</price>
    </item>
  </order>
  <order ...
```

Figure 1 Orders XML. *The example Orders XML document is used to explain JXC functionality*

The Order Summary schema

Figure 2 illustrates an XML document constrained by the example Order Summary schema (OrderSummary.xsd). It contains some of the same data as the example Orders document shown in Figure 1.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<customer_orders>
  <customer_order orderDate="2010-01-10" orderRef="ORD00001" postCode="PE99 9YY"
status="Dispatched" statusCode="DSP" totalValue="49.99"/>
  <customer_order orderDate="2010-02-21" orderRef="ORD00002" postCode="PE99 9YY"
status="Pending" statusCode="PND" totalValue="18.97"/>
  <customer_order orderDate="2010-02-22" orderRef="ORD00003" postCode="HD55 8UY"
status="Pending" statusCode="PND" totalValue="1.78"/>
</customer_orders>
```

Figure 2 Order Summary XML. *The example Order Summary XML is used to explain how the XML Mapper works*

Using the JXC API in your Java project

This section describes how to use the JXC API in your Java project. Begin by adding jxc.jar to your project class path.

Filtering XML

The key class for filtering XML is called XMLFilter as illustrated by Figure 3 where it has been used to extract only the order header information (by omitting deliveryAddress, billingAddress and item elements) for customer Henrys dispatched orders (by conditionally retaining order elements).

```
XMLFilter filter = new XMLFilter();
filter.addCriteria(new XMLFilterCriteria("deliveryAddress", XMLFilterCriteria.Action.OMIT));
filter.addCriteria(new XMLFilterCriteria("billingAddress", XMLFilterCriteria.Action.OMIT));
filter.addCriteria(new XMLFilterCriteria("item", XMLFilterCriteria.Action.OMIT));
filter.addCriteria(new XMLFilterCriteria(
    "order", XMLFilterCriteria.Action.RETAIN, "customerName = 'Henry' and status = 'Dispatched'"));
String filteredXML = XMLUtil.toXMLString(filter.apply(new File("Orders.xml")));
System.out.println("Filtered XML="+filteredXML);
```

Figure 3 Example XMLFilter Usage. The XMLFilter API can be used to filter XML documents

Filter criteria behave according to Table 1.

Action	Condition	Behaviour
RETAIN	Empty	No effect on the XML document
RETAIN	Entered	The element will only be retained if the condition evaluates to true
OMIT	Empty	The element will always be removed from the XML document
OMIT	Entered	The element will only be omitted if the condition evaluates to true

Table 1 Filter Criteria Behaviour

Therefore, based upon the example XML illustrated in Figure 1, the output of the code example listed in Figure 3 is shown in Figure 4 below.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><orders xmlns:ns1="null"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="null Orders.xsd">
  <order orderId="1">
    <customerId>1</customerId>
    <orderRef>ORD00001</orderRef>
    <customerName>Henry</customerName>
    <orderDate>2010-01-10</orderDate>
    <status>Dispatched</status>
  </order>
</orders>
```

Figure 4 Example XMLFilter Output. XMLFilter has been used to create a new document containing only customer Henry's dispatched orders

Mapping XML

This section describes how to use the XML Mapper API in your Java project to map the example Orders XML document to produce a new document that conforms to the Order Summary schema containing only customer Henry's orders. Since the target schema (Order Summary) requires a statusCode attribute that is not contained within the Orders XML, it must be defaulted.

Figure 5 shows how the XMLMapper class would be used to map only the orderRef, orderDate, postcode and status, default a value for statusCode as well as aggregating order item prices into totalValue from the verbose Orders schema to the much simplified Order Summary schema.

```
XMLMapper mapper = new XMLMapper();
mapper.addMapping(new XMLMapping( "orders", "customer_orders");
mapper.addMapping(new XMLMapping(
    "orders/order", "customer_orders/customer_order", "customerName = 'Henry'");
mapper.addMapping(new XMLMapping(
    "orders/order/orderDate", "customer_orders/customer_order/@orderDate");
mapper.addMapping(new XMLMapping(
    "orders/order/orderRef", "customer_orders/customer_order/@orderRef");
mapper.addMapping(new XMLMapping(
    "orders/order/deliveryAddress/postCode",
    "customer_orders/customer_order/@postCode");
mapper.addMapping(new XMLMapping(
    "orders/order/status", "customer_orders/customer_order/@status");
mapper.addMapping(new XMLDefault(
    "customer_orders/customer_order/@statusCode", "status = 'Dispatched'", "DSP");
mapper.addMapping(new XMLDefault(
    "customer_orders/customer_order/@statusCode", "status = 'Pending'", "PND");
mapper.addMapping(new XMLMapping(
    "orders/order/item/price", "customer_orders/customer_order/@totalValue", null,
    XSLTFunction.SUM);
String mappedXML = XMLUtil.toXMLString(mapper.apply(new File("Orders.xml")));
System.out.println("Mapped XML="+mappedXML);
```

Figure 5 Example XMLMapper Usage. *The XMLMapper API can be used to map XML from one schema to another*

Figure 6 shows the output obtained from running the example code illustrated in Figure 5.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><customer_orders>
  <customer_order orderDate="2010-01-10" orderRef="ORD00001" postCode="PE99 9YY"
status="Dispatched" statusCode="DSP" totalValue="49.99"/>
  <customer_order orderDate="2010-02-21" orderRef="ORD00002" postCode="PE99 9YY"
status="Pending" statusCode="PND" totalValue="18.97"/>
</customer_orders>
```

Figure 6 Example XMLMapper Output. *XMLMapper has been used to create a new Order Summary XML document from the information held in an Orders XML document*

XML mappings can be categorised into three different types: XMLControl, XMLMapping and XMLDefault, according to their attributes.

Table 2 describes how XMLMapper processes each of these three mapping types.

Mapping Inputs			Mapping Type	
Source (sourcePath)	Target (targetPath)	Default (defaultValue)	Class	Description
Y	N	N	XMLControl	Causes XMLMapper to iterate over the source document using the XPath selection specified by the sourcePath parameter. This kind of mapping does not result in any elements or attributes being output into the target document. Therefore, this kind of mapping is purely for controlling how XMLMapper iterates over the source document.
Y	Y	N	XMLMapping	When both the sourcePath and targetPath parameters are specified, XMLMapper processes the mapping by outputting content into the target document according to: <ul style="list-style-type: none"> A. If the sourcePath and targetPath refer to a branch element then a branch element will be added to the target document. B. If the sourcePath and targetPath refer to leaf element or attribute then a data element or attribute will be added to the target document containing the data found.
N	Y	Y	XMLDefault	Causes the XMLMapper to output the value of the defaultValue parameter into the leaf element or attribute specified by the targetPath parameter.

Table 2 XML mapping behaviour. XMLMapper processes mappings differently according to the values specified for the source, target and default parameters

Please note: it is possible for all three mapping types (control, mapping and default) to be conditionally processed by specifying a Boolean XPath condition in the condition constructor argument of an AbstractXMLMapping sub-class.

Using the GUI tools

This section describes how to use the Java XML Converter tools. The tools may be launched by double clicking on jxc.jar.

Mapping XML

The first example uses JXC's XML mapping functionality to transform the example Orders XML document into a new document that conforms to the Order Summary schema. Since the target schema (Order Summary) requires a statusCode attribute that is not contained within the Orders XML, it must be defaulted.

Step 1: Launch the GUI

Once opened, the GUI will display the window shown in Figure 7.

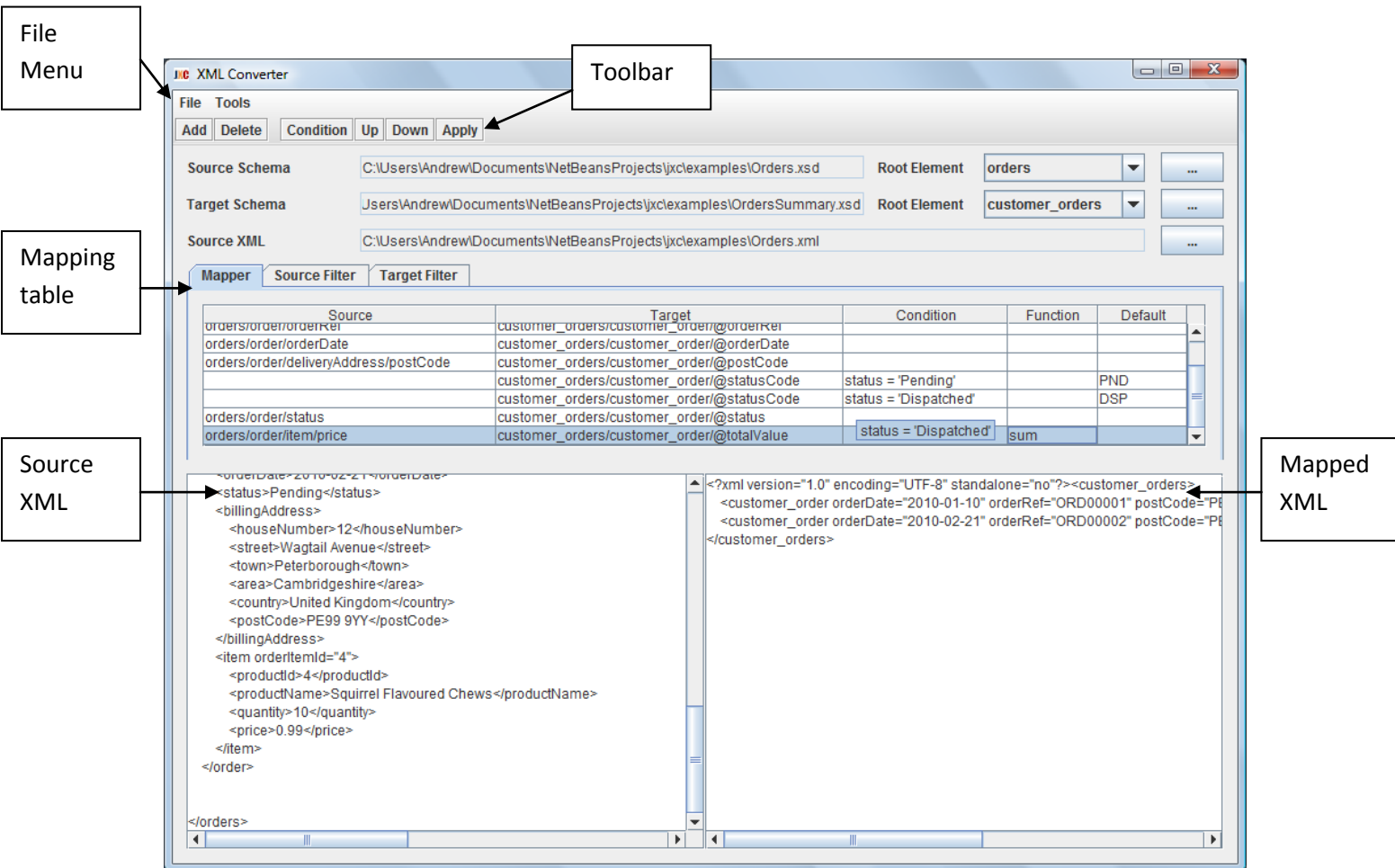


Figure 7 JXC GUI. The GUI can be used to create and test xml mappings working with static XML files

Step 2: Open a source XML schema

Begin by selecting the XML schema for the type of source XML document you wish to map. Do this by clicking the button labelled "..." placed to the right of the Source Schema field. Simply select the schema file from the file chooser dialog that is displayed. Selection will result in the root element list being populated with all top level elements defined in the schema. You must now qualify the Source Schema with the name of the root element you wish to use by selecting from the list box to the right of the Source Schema field.

Step 3: Open a target XML schema

Now select the XML schema for the type of target XML document you wish to map the source to. Do this by clicking the button labelled “...” placed to the right of the Target Schema field. Simply select the schema file from the file chooser dialog that is displayed. Selection will result in the list box field of Mapping column in the mapping table being populated with all valid elements and attributes defined in the target schema. You must now qualify the target schema with the name of the root element you wish to use by selecting from the list box to the right of the Target Schema field.

Step 4: Open a source XML document

Next, open an example XML document that matches the Source Schema previously selected. Once both XML schemas and an XML document have been opened, you can begin to build the mappings.

In the Figure 7 example, Orders.xml has been selected.

Step 5: Enter some mappings and mapping conditions

Begin by clicking the Add button located on the toolbar at the top of the window and then locating the element you wish to map in the Source schema from the popup tree window displayed in Figure 8. Click OK and the selected item will be copied into the source column of the mapping table.

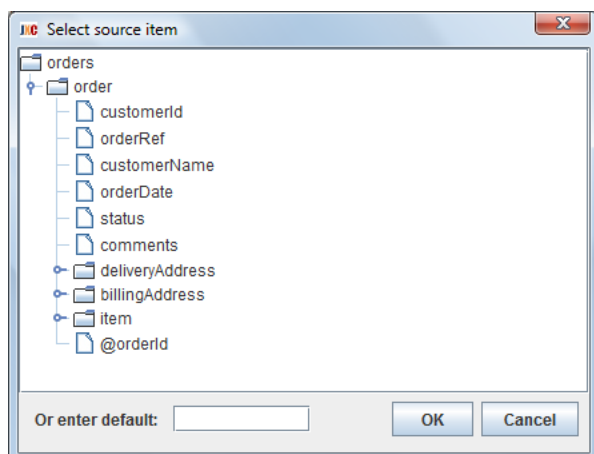


Figure 8 Select Source Item. *The Schema item selector allows items to be selected from the source schema using a tree view*

Next, click in the target column of the mapping table and then click the button that appears to show the tree view shown in Figure 9.

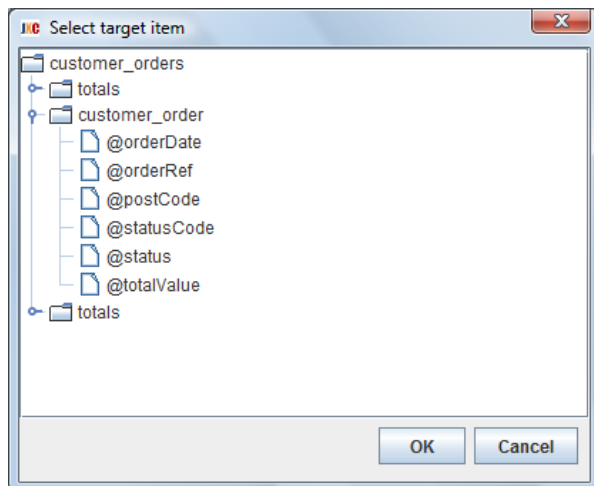


Figure 9 Select Target Item. *The Schema item selector allows items to be selected from the target schema using a tree view*

From this tree select the element or attribute you wish to map to in the Target Schema.

You may also optionally specify a mapping condition in the next column along. Any Boolean XPath expression may be entered. By selecting the toolbar button labelled Condition for the selected mapping the expression builder window described later may be launched to help you construct a valid XPath condition for the mapping.

Since the OrderSummary schema can have a statusCode attribute not available in the source schema (Orders.xsd) it must be defaulted. This is done by clicking Add for a new mapping and then entering a default value in text input field at the bottom of the popup tree that appears as illustrated by Figure 9. Defaulting can also be conditional and in the example, two mappings are used to default specific status codes based on the value of the status element in the Orders XML file.

Finally, should you wish to apply an output function to a mapping you may do so by selecting from the list box in the forth column labelled Function. In the example illustrated by Figure 7, the sum function has been used to aggregate the value of each order line element in the source schema into the summary attribute totalValue of the target schema.

Please see Table 2 for more information about how different types of mapping are processed depending upon the values entered in the source, target, condition and default cells of the mapping table.

Step 6: Test the mapping

Once satisfied with your mapping, simply click the Apply button located on the tool bar at the top of the window. The mapped results will then be displayed in the text area located to the right of the original XML document opened during step two.

Step 7: Saving and restoring the mapping

A tested XML mapping may be saved for future use by selecting the option labelled Save Mapping from the File menu. When the file chooser opens, simply specify the folder and file name where you wish to save the mapping file to. Previously saved mapping files may be opened by clicking the menu item labelled Open Mapping on the File Menu.

Step 8: Exporting the generated XSL

Should you wish to use the XSL style sheet generated externally, you can simply select the option labelled Save Generated XSL from the File menu and you will be prompted to choose a file location to export the XSL to.

Pre Filtering XML

If desired it is possible to filter the source XML prior to mapping. This may be done once steps one to four described above in Mapping XML have been completed. It is then accomplished by:

Step 1: Switch to the Source Filter tab

Select the tab header labelled Source Filter to reveal the filter criteria table as show in Figure 1.

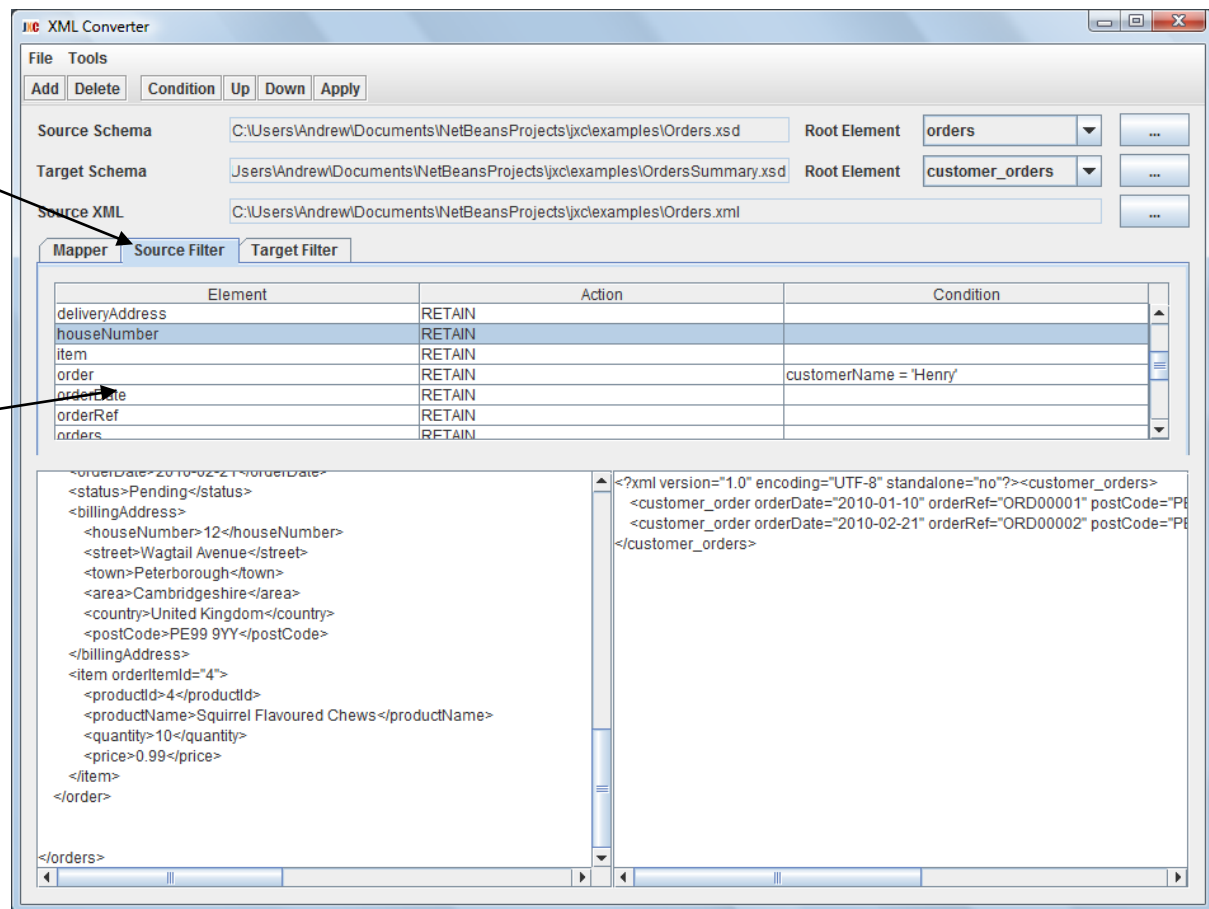


Figure 10 Source Filter. The source XML document map be pre filter prior to mapping

Step 2: Enter some filter criteria

Once the Source Filter tab has been selected you can define filter criteria by editing the action and condition cells next to each element listed in the filter criteria table.

Please note, the sequence in which criteria are processed is important because omission of an element that a subsequent criteria refers to in its expression will cause the latter criteria to fail. Use the Up and Down toolbar buttons to adjust the precedence of filter criteria.

Any valid XPath expression may be entered in the condition column for each criteria. Selecting an element row in the filter criteria table and clicking the Condition toolbar button will open the expression builder described in the section entitled Using the Expression Builder.

Step 3: Test the filter

Once satisfied with your filter criteria, simply select the Apply toolbar button found at the top of the window. The filtered results will then replace the original source XML document opened in the text area located at the bottom left hand corner of the window.

Step 4: Saving and restoring filters

Tested XML filters may be saved for future use by selecting the option labelled Save Source Filter from the File menu. When the file chooser opens, simply specify the folder and file name where you wish to save the filter file. Previously saved filter files may be opened by using the menu option labelled Open Source Filter also located on the File menu.

Similarly, it is also possible to filter XML that has been mapped by selecting the tab labelled Target Filter and entering filter criteria in the same way. Only this time the filter is applied to the mapped XML displayed in the text area to the lower right hand side of the window.

Using the Expression Builder

This section describes how to use the Expression Builder with the XML Filter and XML Mapper tools.

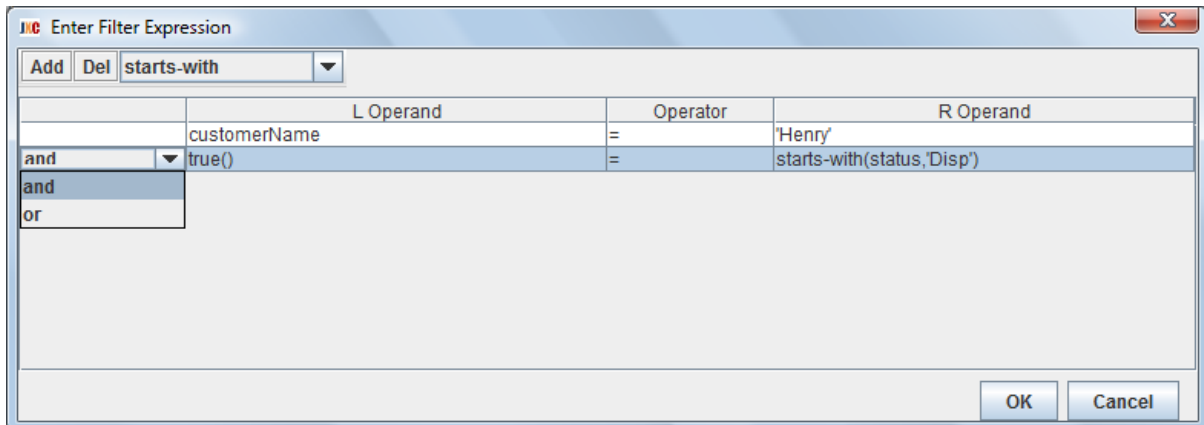


Figure 11 Expression Builder. *The Expression Builder allows XPath expressions to be constructed*

The Expression Builder allows fairly complex XPath expressions to be constructed and checks syntax validity. The operand cells allow a node to be selected from the popup schema item selector shown in Figure 12 or a value to be typed in.

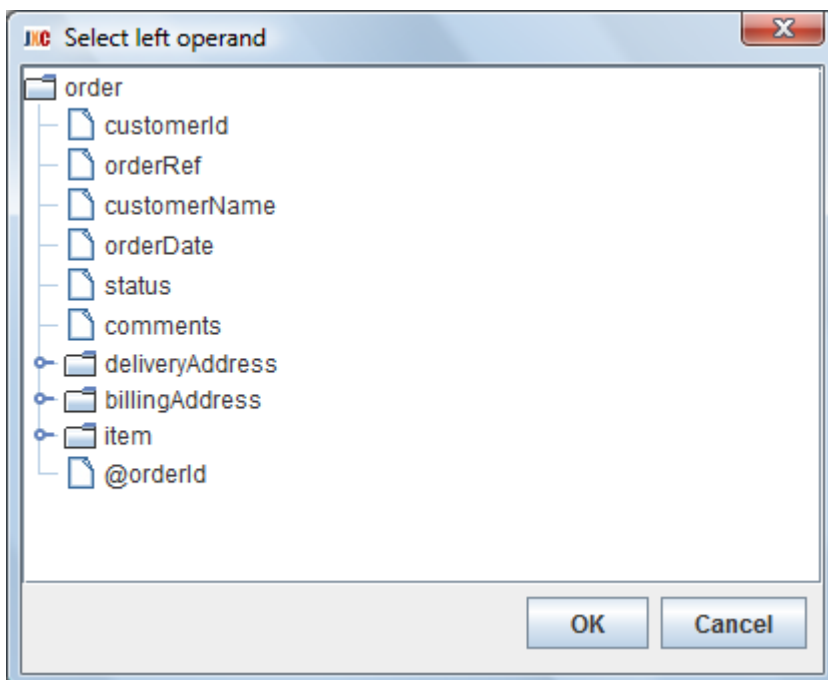


Figure 12 Schema Item Selector. *The schema item selector is used to select items from the XML schema when building XPath expressions*

Once entered, you can apply a function to the operand cell contents by selecting a function from the list box located to the right of the Del button (this can only be done with the cursor still in the operand cell, i.e. the cell must be in editing mode). Upon choosing a function you may be prompted to enter some additional function parameter values.

For example, the following prompt shown in Figure 13 would be issued if you selected the contains function in order to capture the search string.

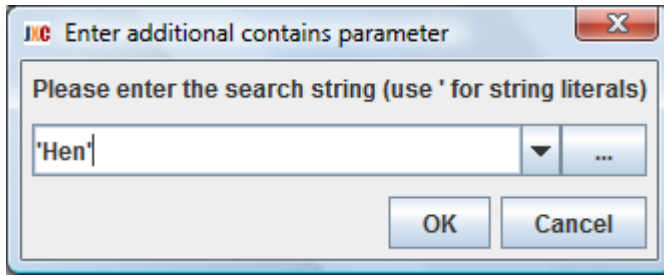


Figure 13 Additional Parameters. *Users are prompted to enter values for additional function parameters*

Note, you can either type in an additional parameter value or select an element to select it from using the '...' button. If a string literal is entered for either operand be sure to enclose in single quotation marks but be sure to omit quotation marks for numbers.

XML namespace support

Both the XML filter and XML mapper APIs are namespace aware. This means that when specifying mapping paths, filter elements or conditions for XML documents that contain elements with namespace prefixes, the prefix must be included in any element names specified. For example, consider the following XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<ns0:orders xmlns:ns0="http://jxc.orders">
  < ns0:order orderId="1">
    < ns0:customerId>1</ ns0:customerId>
    < ns0:orderRef>ORD00001</ ns0:orderRef>
    < ns0:customerName>Henry</ ns0:customerName>
    < ns0:orderDate>2010-01-10</ ns0:orderDate>
    < ns0:status>Dispatched</ ns0:status>
  ...
```

Therefore, to create a filter criteria for customer Henry's orders you would have to specify:

```
new XMLFilterCriteria( "ns0:order", XMLFilterCriteria.Action.RETAIN, "ns0:customerName = 'Henry'")
```

Likewise a mapping for the orderData element would look like:

```
new XMLMapping( "ns0:orders/ns0:order/ns0:orderDate", ...)
```

Please note it is not currently possible to map into target elements with a namespace prefix using either the API or the tools. The tools currently only support working with documents that use a single namespace prefix. However, the API can support filtering and mapping from documents that contain many namespace prefixes.

Schema recursion

Some XML schemas specify recursive elements. By default the schema item selector illustrated in Figure 8 would only recurse two times, i.e. the tree would only show two child branches for the recursive element. The number of times to recurse may be configured by selecting the menu option labelled Schema Recursion from the Tools menu at the top of both XML Filter and XML Mapper GUIs.

Known limitations

The GUI tools don't support XML schemas that import content from additional XSD files.

The expression builder only supports piping of functions if they are typed into operand cells. Selection of a second function from the Expression Builders function list will overwrite any previously applied function.

XPath 2.0 functions

By default, XPath 1.0 functions are available. See <http://www.w3.org/TR/xpath/> for details. If you wish to use newer XPath 2.0 functions (see <http://www.w3.org/TR/xpath-functions/>) you must have an XPath 2.0 compliant processor such as Saxon on your java classpath. The saxon9he.jar may be obtained from <http://sourceforge.net/projects/saxon/>.

When using JXC from within your Java application all you need do is add the saxon9he.jar to your classpath. When using the GUI tools you must also set the following Java system property in addition to adding saxon9he.jar to your classpath:

```
-DJXC.xslt.spec=XSLT_2
```