XML Feature (IN PROGRESS)

The XML Feature contains operators that allow the processing of XML documents. Add de.uniol.inf.is.odysseus.server.xml.feature to your Odysseus in order to use this feature.

Operators

ToXML

This operator allows the transformation from a Tuple object into a XML document. The transformation requieres a XML schema definition (XSD) which describes the structure of the XML document and the mapping between that document and the Tuple object. However, there are different approaches to provide that schema information to the operator:

- rootElement
- rootAttribute
- xsdFile
- xsdString
- xsdAttribute
- xPathAttributes

Firstly, the schema can be provided via file access (whereas the file can be located either locally or on a webserver).

```
transform = TOXML({
                rootelement = 'user',
                xsdfile = '../schema.xsd'
              },
              input
            )
```

Secondly, the schema information can be passed via string.

```
transform = TOXML({
               rootelement = 'user',
               xsdstring = '<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:
xs="http://www.w3.org/2001/XMLSchema">
                                                          <xs:element name="user">
                                                            <xs:complexType>
                                                              <xs:sequence>
                                                                <xs:element type="xs:string" name="nickname"/>
                                                                <xs:element type="xs:string" name="fname"/>
                                                                <xs:element type="xs:string" name="lname"/>
                                                                <xs:element type="xs:string" name="email"/>
                                                                <xs:element type="xs:byte" name="age"/>
                                                                <xs:element name="roles">
                                                                  <xs:complexType>
                                                                    <xs:sequence>
                                                                      <xs:element type="xs:string" name="role"/>
                                                                    </xs:sequence>
                                                                  </xs:complexType>
                                                                </xs:element>
                                                              </xs:sequence>
                                                            </xs:complexType>
                                                          </xs:element>
                                                        </xs:schema>'
             },
             input
           )
```

Thirdly, the schema information can be a payload on the current Tuple object which can be advantageous if the XSD has to change dynamically (`xsdattribute = 'xsd'`). Thus, it is possible to produces different documents within one data stream. Besides that, the root element of the document might change and therefore it is possible to define an attribute of the Tuple object that carries this information as well (`rootattribute = 'root'`).

```
transform = TOXML({
               rootattribute = 'root',
               xsdattribute = 'xsd'
             },
             input
           )
```

Furthermore, the operator allows to use XPath expressions for an enriched mapping between attributes of the Tuple object and the XML document. The expressions can be applied as `options` or again as a payload on the Tuple object (`xpathattributes = ['xpaths']`). If `xpathattributes` is used, each defined Tuple attribute can contain several expressions. Every expression is followed by an attribute name that has to match with an attribute of the Tuple object (e.g. [attributename][delimeter][expression]). These pairs are also separated by a delimiter (default is the , symbol). If the `options` alternative is used, the attribute-expression pair is given explicitly as option parameter.

```
transform = TOXML({
               rootelement = 'user',
               xsdfile = '../schema.xsd',
               options = [
                 ['user_id', '/user/@id'],
                 ['role_1', '/user/roles/role[1]'],
                 ['role_2', '/user/roles/role[2]'],
                 ['role_3', '/user/roles/role[3]']
               ]
             },
             input
           )
```

```
transform = TOXML({
                rootelement = 'user',
                xpathattributes = ['xpaths'],
                xsdfile = '../schema.xsd',
                options = [['delimeter', ';']]
              },
                          input
            )
```

XMLToTuple

It is also possible to transform XML documents to Tuple objects whereas the mapping between attributes is done by XPath expressions and Tuple schema.

```
transform1 = XMLTOTUPLE({
                  schema=[
                    ['nickname','String'],
                    ['fname', 'String'],
                    ['lname', 'String'],
                    ['email', 'String'],
                    ['role', 'String'],
                    ['age', 'Integer']
                  ],
                  expressions = [
                    ['nickname', '/user/nickname'],
                    ['fname', '/user/fname'],
                    ['lname', '/user/lname'],
                    ['email', '/user/email'],
                    ['role', '/user/roles/role'],
                    ['age', '/user/age']
                  ]
                },
                input
              )
```

XMLEnrich

With this operator, it is possible to combine two different documents into one single document:

- target
- predicate

The operator functions as the Enrich operator that evaluates a given predicate and only merges the two documents if the predicate is true. In order to compare two documents, predicates can consist of XPath expressions that return a single value. The second parameter is the target parameter which specifies a location in the left document where the right document should be inserted. However, the right document is the enrichment for the left document.

```
out = XMLENRICH({
          target = '/user',
          predicate = 'toString(/user/@id)==toString(/roles/@id)'
        },
        input,
        enrichment
      )
```

XMLMap

It is possible to use MEP Functions and XPath expression together within XML documents to manipulate values or project only a subset of the origin document (e.g. this operator works like Map operator).

```
mapped = XMLMAP({
                expressions = [
                                ['toLong({/user/age}) + 10000', '/user/age'],
                                ['concat(toString({/user/fname}), "__XX01")', '/user/fname'],
                                ['{/user/nickname}', '/user/nickname']
                ]
              },
              input
            )
```

XMLSplit

This operator splits a document in subsets whereas the each split point is defined by an XPath expression.

```
splitted = XMLSPLIT({EXPRESSIONS = ['/user/roles', '/user/fname']}, input)
```

XMLTransform

The XMLTransform operator allows to transform a given XML with Extensible Stylesheet Language Transformations (XSLT):

- xsltstring
- xsltfile
- dynamic

There are three different ways to provide the XSLT information. First, the information given by a string in the operator definition.

```
trans1 = XMLTRANSFORM({
                        xsltstring='<?xml version="1.0"?>
                                                <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999
/XSL/Transform">
                                                    <xsl:template match="/">
                                                        <html>
                                                            <body>
                                                                <h2>My CD Collection</h2>
                                                                <table border="1">
                                                                    <tr bgcolor="#9acd32">
                                                                        <th>Title</th>
                                                                        <th>Artist</th>
                                                                    </tr>
                                                                    <xsl:for-each select="catalog/cd">
                                                                        <tr>
                                                                            <td><xsl:value-of select="title"/><
/td>
                                                                            <td><xsl:value-of select="artist"/><
/td>
                                                                        </tr>
                                                                    </xsl:for-each>
                                                                </table>
                                                            </body>
                                                        </html>
                                                    </xsl:template>
                                                </xsl:stylesheet>'
          },
          input
        )
```

Secondly, the XSLT can be read from file (whereas the file can be located either locally or on a web server).

```
trans2 = XMLTRANSFORM({
                        xsltfile = '../transform.xslt'
                        },
                        input
                  )
```

And thirdly, the operator can be configured (`dynamic = true`) in a way that the XML document will be scanned for an embedded XSLT.

```
trans3 = XMLTRANSFORM({
               dynamic = true
             },
             input
          )
```

Wrapper

XML2 ProtocolHandler

This Protocol handler allows to load XML documents which will be represented as the KeyValueObject data type.

```
inputXML2 = ACCESS({
                 source='xml',
                 protocol='XML2',
                 transport='File',
                 wrapper='GenericPull',
                 datahandler='KeyValueObject',
                 metaattribute = ['TimeInterval'],
                 options = [
                   ['filename', '../document.xml']
                 ]
               }
```

XML3 ProtocolHandler

This Protocol handler allows to load XML documents into the Odysseus system whereas the document will be represented by the XMLStreamObject data type. Furthermore, with the `tag_to_strip` option it is possible to shorten the XML document and thus process only a part of the document. The option awaits an element name that is present in the document.

```
input = ACCESS({
             source='xml',
             protocol='XML3',
             transport='File',
             wrapper='GenericPull',
             datahandler='XMLStreamObject',
             options=[
               ['filename', '../document.xml'],
               ['tag_to_strip', 'user']

]


}

        )
```