Importing XSLT in XQuery

The ability to import and use functions from XSLT was part of the requirement for XQuery 3.0, which was pushed back for consideration in the proposed XQuery 3.1 specification. We can extend this to importing an entire XSL stylesheet in XQuery and to use and set its components, such as, global variables, parameters and modes, etc. The bugzilla issue #17601 provides some discussion on this new proposed feature. It also provides links to various implementations which implement the importing of XSLT in XQuery.

In this document we attempt to propose a standard way of importing XSLT in XQuery. We begin by reviewing existing implementations of the feature.

Vendor 1: existdb

The eXistdb has two sets of functions. The first set consists of stream based functions, which means the output to the XSLT is streamed back to the calling program. The second set of functions provides the output to the XSLT as a *node()*.

1. Stream based functions

transform:stream-transform($node-tree as node()*, $stylesheet as item(), $parameters as node()?) as item()

transform:stream-transform($node-tree as node()*, $stylesheet as item(), $parameters as node()?, $serialization-options as xs:string) as item()

2. Standard XSL functions

transform:transform($node-tree as node()*, $stylesheet as item(), $parameters as node()?) as node()?

transform:transform($node-tree as node()*, $stylesheet as item(), $parameters as node()?, $attributes as node()?, $serialization-options as xs:string?) as node ()?

COMMENTS:

Secondary document can be passed.

Can specify if an XQuery error is generated if the XSL processor reports a warning or error. This is achieved as a parameter option.

The stylesheet is specified either as an URI or a node. It is possible to pass the stylsheet in the parameters as a XML fragment.

Vendor 2: BaseX

XSL stylesheets can be transformed to a node() or text. The signatures of the functions are given below:

xslt:transform($input as item(), $stylesheet as item()) as node()
xslt:transform($input as item(), $stylesheet as item(), $params as item()) as node()

xslt:transform-text($input as item(), $stylesheet as item()) as xs:string
xslt:transform-text($input as item(), $stylesheet as item(), $params as item()) as xs:string

For the $input and the $stylesheet arguments we have the option for a filename, string or node. The $params argument is optional and can be suplied as an XML element or as a map.

COMMENTS:
The errors in the transformation are bound to a specific BaseX error. However there is no mention of how to handle named template and other options.

Vendor 3: MarkLogic

Provides two functions to invoke XSLT. Namely xslt-eval and xslt-invoke. The signatures of these functions are given below. The difference is that in xslt-eval assumes the input stylesheet and source document are of type element and node, respectively. Whereas, xslt-invoke as the stylesheet as a filename with a path. The input document is a node

```
xdmp:xslt-eval(
  $stylesheet as element(),
  $input as node()?,
  [$params as map:map?],
  [$options as node()?]
) as document-node()*
```

```
xdmp:xslt-invoke(
  $path as xs:string,
  $input as node()?,
  [$params as map:map?],
  [$options as node()?]
) as document-node()*
```

COMMENTS:
The output is one or more document nodes. In xslt-eval the stylesheet has to be first parsed and escaped in XQuery. There is potentially a problem if the context node is not a document node, i.e. an element from a query

Proposals

Norman Walsh:

```
fn:invoke-xslt(
   $stylesheet as document-node(),
   $document as document-node()?,
   $params as map()?,
   $options as map()?
 ) as map()
```

$stylesheet is the stylesheet (an xsl:stylesheet, xsl:transform or xsl:package)
$document is the (optional) document to be transformed
$params is a map that contains the stylesheet parameters
$options is a map that contains the stylesheet options

The main problem with this is that it doesn't keep pace with the variety of mechanisms for stylesheet invocation that are now available in XSLT 3.0.

Mike Kay:

```
transform($options as map(*)) as map(*)
```

We consider the input and the return type of this function to be both maps. Firstly, for the input map we have a number of options available, which are detailed below (for further details, see XSLT 3.0, section 2.3 Initiating a Transformation):

Option name	Value	Meaning
xslt-version	decimal	The minimum level of the XSLT language that the processor must support. Defaults to the version attribute within the stylesheet.
package-name	URI	The name of the top-level stylesheet package to be invoked
package-version	decimal	The version of the top-level package to be invoked
package-node	node()	The xsl:package element of the top-level package as a node
package-text	string	The top-level stylesheet package in source XML form.
package-location	URI	The location of the package to be invoked
stylesheet-location	URI	The location of a document containing the principal

Option name	Value	Meaning
		stylesheet module in source XML form.
stylesheet-node	element()	The xsl:stylesheet or xsl:transform element of the principal stylesheet module
stylesheet-text	string	The principal stylesheet module in the form of lexical XML.
base-uri	URI	The URI to be used as the base URI of the principal stylesheet module or top-level stylesheet package
static-params	map(QName, item()*)	A map holding the values to be supplied for static stylesheet parameters
stylesheet-params	map(QName, item()*)	A map holding the values to be supplied for non-static stylesheet parameters
source-node	node()	A value to be used both as the global-context-item and as the initial-match-selection
global-context-item	item()	The value to be used as the context item for evaluating global variables
initial-mode	QName	The name of the initial mode for apply-templates invocation
initial-template	QName	The name of the initial template for call-template invocation
template-params	map(QName, item()*)	A map containing values to be supplied for the non-tunnel parameters of the initial template, used with both apply-templates and call-template invocation
tunnel-params	map(QName, item()*)	A map containing values to be supplied for the tunnel parameters of the initial template, used with both apply-templates and call-template invocation
initial-function	QName	The name of the initial function to be called for call-function invocation. The arity of the function is inferred from the length of function-params
function-params	array(item()*)	An array of values to be used as

Option name	Value	Meaning
		the arguments to the initial function call
initial-match-selection	item()*	the sequence to which the initial apply-templates invocation is applied (alternative to supplying source-node)
format	"raw"|"document"|"serialized"|"saved"	"raw" indicates that the result of the initial template or function is returned directly. "document" indicates that it is wrapped in a document node. "serialized" indicates that it is wrapped in a document node and then serialized. "saved" indicates that it is wrapped in a document node, then serialized, and that the serialized form is written to some external destination rather than being returned as a result of the transform() function.
serialization-params	map{QName, string}	Serialization parameters, overriding any serialization parameters specified within the stylesheet.
output	URI	The base output URI. This acts to identify the principal result document, and if format="saved" it also provides the destination to which the principal result document is saved.
vendor-options	map{QName, string()}	Implementation specific options are given here as a map.

The above options provide several ways of supplying the same information, e.g. stylesheet-location versus stylesheet-node. The ordering in the table represents a priority order; where several parameters are in effect alternatives, the first one that is supplied wins.

The map that is returned contains one entry for the principal result document and one entry for each secondary result document. The keys are URIs, and the values depend on "format". The principal result document is keyed on the value of the base output URI; if no base output URI was supplied then the key will be "output". See below what can be returned:

format="raw" → item()*
format="document" → document-node()

format="serialized" → string
format="saved" → URI (the URI of the location to which the serialized document was saved: typically the same as the key, but perhaps different if some kind of redirection has taken place)

The context (static and dynamic) for the transformation is as defined in the XSLT specification. We might state that it is desirable for resources available in the query (schema components, collations) to also be available in the stylesheet.

Use cases

Use case 1
This use use case demonstrates the use of the transform function with a stylsheet file and source document node. We access the result document by key in the map.

```
let $v := transform(map {"stylesheet-uri" : "test.xsl", "source-node" : $input })

f($v?output)
```

==========

Use case 2
This use case demonstrates the ability to retrieve multiple result documents. We also show the use of

```
declare $xsl as xs:string :="<xsl:stylesheet xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
version='3.0'
xmlns:app='http://www.example.com'>

      <xsl:template name='app:main' >
        <xsl:for-each select='section'>
          <xsl:result-document href='section{position()}.html'>
            <!-- instructions content here -->
          </xsl:result-document>
        </xsl:for-each>
      </xsl:template>

</xsl:stylesheet>";

element{xs:QName('html')}{
         element {xs:QName('body')} {
                  for $x in transform(map{"stylesheet-text":$xsl, initial-template:"app:main"})?*
                  return $x
         }
}
```

==========

Use Case 3

Content of variable.xsl:

```
<xsl:stylesheet version='1.0'
    xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
  <xsl:param name='v'/>
    <xsl:template match='/'>
      <v><xsl:value-of select='$v'/></v>
    </xsl:template>
</xsl:stylesheet>
```

s

XQuery:

```
let $in := <dummy/>
let $style := doc('variable.xsl')
return (
  transform(map{"source-node":$in, "stylesheet-node":$style, "stylesheet-params": map
{ QName("v"): 1 } } ) )?output
```

==========

Use Case 4

```
(: Outputs the result as html. :)
declare option output:method 'html';

let $in :=
  xs:string("<books>
    <book>
      <title>XSLT Programmer’s Reference</title>
      <author>Michael H. Kay</author>
    </book>
    <book>
      <title>XSLT</title>
      <author>Doug Tidwell</author>
      <author>Simon St. Laurent</author>
      <author>Robert Romano</author>
    </book>
  </books>")

let $style :=
  xs:string("<xsl:stylesheet version='2.0' xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
  <xsl:output method='html'/>
    <xsl:template match='/'>
<html>
  <body>
    <div>
      <xsl:for-each select='books/book'>
       <b><xsl:apply-templates select='title'/></b>: <xsl:value-of select='author'/><br/>
      </xsl:for-each>
    </div>
  </body>
</html>
    </xsl:template>
  </xsl:stylesheet>")

return transform(map{"source-node":$in, "stylesheet-text":$style, "serialization-option":
map{QName("indent"): "yes"} } )?*
```