Using the Mozilla JavaScript interface to XSL Transformations

This document describes the JavaScript interface in Mozilla 1.2 and up to the XSLT Processing Engine (TransforMiiX).

Creating an XSLTProcessor

To start, you need to create an XSLTProcessor object:

```
view plain print ?
01. var processor = new XSLTProcessor();
```

Specifying the stylesheet

Before you can use it, you must import a stylesheet with the `importStylesheet()` function. It has a single parameter, which is the DOM Node of the XSLT stylesheet to import.

Note: The import is live, meaning that if you alter the stylesheet DOM after importing it, this will be reflected in the processing. Rather than modifying the DOM it is recommended to use stylesheet parameters which are usually easier and can give better performance.

```
view plain print ?
01. var testTransform = document.implementation.createDocument("", "test", null);
02. // just an example to get a transform into a script as a DOM
03. // XMLDocument.load is asynchronous, so all processing happens in the
04. // onload handler
05. testTransform.addEventListener("load", onload, false);
06. testTransform.load("test-transform.xml");
07. function onload() {
08.   processor.importStylesheet(testTransform);
09. }
```


`importStylesheet` requires one argument, a DOM Node. If that node is a document node, you can pass in a full XSL Transform or a literal result element transform, otherwise it must be an `xsl:stylesheet` or `xsl:transform` element.

Transforming the document

You can use the `transformToDocument()` or `transformToFragment()` methods to transform a document using the imported XSLT stylesheet.

transformToDocument

`transformToDocument()` takes one argument, the source node to transform, and returns a new DOM `Document` with the results of the transformation:

```
view plain print ?
01. var newDocument = processor.transformToDocument(domToBeTransformed);
```

The resultant object depends on the output method of the stylesheet:

- **html** - `HTMLDocument`
- **xml** - `XMLDocument`
- **text** - `XMLDocument` with a single root element `<transformiix:result>` with the text as a child

transformToFragment

You can also use `transformToFragment()` which will return a DOM `DocumentFragment` node. This is handy because appending a fragment to another node transparently appends all the children of that fragment, and the fragment itself is not merged. Fragments are therefore useful for moving nodes around and storing them without the overhead of a full document object.

`transformToFragment()` takes two arguments: the source document to be transformed (as above) and the `Document` object that will own the fragment (all fragments must be owned by a document).

```
view plain print ?
01. var ownerDocument = document.implementation.createDocument("", "test", null);
02. var newFragment = processor.transformToFragment(domToBeTransformed, ownerDocument);
```

`transformToFragment` will only produce HTML DOM objects if the owner document is itself an `HTMLDocument`, or if the output method of the stylesheet is HTML. It will **not** produce an HTML DOM objects if only the toplevel element of the result is `<html>` as `transformToFragment` is rarely used to create this element. If you want to

override this, you can set the output method normally in the standard way.

transforming HTML

Unfortunately it is currently not supported to transform HTML nodes using XSLT. Some things work if you use lower case node-names in patterns and expressions, and treat the nodes as if they are in the null namespace, however this is not very well tested so it might not work in all situations. It is also possible that this will change in a future release.

Transforming XHTML should work as expected though.

Setting parameters

You can control parameters for the stylesheet using the `setParameter`, `getParameter`, and `removeParameter` methods. These all take a namespace URI and a local name as the first two parameters, with `setParameter` taking a third - the value of the parameter to be set. See The XSLT/JavaScript Interface in Gecko for an example.

Resetting

The `XSLTProcessor` object also implements a `reset()` method, which can be used to remove all stylesheets and parameters then put the processor back into its initial state. This method is implemented in Mozilla 1.3 and later.

Resources

- nsIXSLTProcessor.idl will always reflect the actual interface of the `XSLTProcessor` object
- nsIXSLTProcessorObsolete.idl Obsolete : the JS interface in Mozilla versions prior to 1.2.

Using XSLTProcessor from XPCOM components

Instantiating an `XSLTProcessor` from an XPCOM component requires a different syntax as the constructor is not defined inside components.

Instead of this:

```
view plain  print  ?
01. var processor = new XSLTProcessor();
```

Do this:

```
view plain  print  ?
01. var processor = Components.classes["@mozilla.org/document-transformer;1?type=xslt"]
02.                           .createInstance(Components.interfaces.nsIXSLTProcessor);
```

See also

- The XSLT JavaScript Interface in Gecko
- document.load() regarding the loading of XML documents (as used above)

Original Document Information

- Author(s): Mike Hearn
- Last Updated Date: December 21, 2005
- Copyright Information: Copyright (C) Mike Hearn

Page last modified 01:32, 8 Mar 2010 by trevorh