FlowBasedXMLEditor

Creating and Editing XML Documents using FLow

- TARGET-AUDIENCE: beginner ***advanced*** expert
- COCOON-RELEASES: 2.1.4, 2.1.5
- DOCUMENT-STATUS: ***draft*** reviewed released

What you will get from this page

This page will hopefully give you an insight into generating and processing XML documents from within Flow.

Your basic skills

Comfortable with Flowscript

Technical prerequisites

Cocoon 2.1.5

Links to other information sources

---

page metadata

- AUTHOR: Tony Edwards
- AUTHOR-CONTACT: anthonybedwards AT optusnet DOT com DOT au
- REVIEWED-BY:BR
- REVIEWER-CONTACT:BR

My first cocoon application was a simple xml editor written under version 2.0.4.

Utilising actions extensively it became a bit of a maintenance nightmare. It basically allowed the user to create, edit and render into different formats an xml document derived from a proprietory namespace. One of my greatest reservations about this method was the heavy emphasis on Java as I'm not much of a java programmer at all.

When 2.1.4 came along I thought I'd port the application and jazz it up at the same time. I managed to scrape up enough examples and snippets to come up with a fairly usable solution.

Using flow to manipulate xml proved to be pretty straight forward, although I'm not too sure as to the efficiency and optimisability of its application.
So far so good though.

First step was to create a new flowscript file and include the necessary XML manipulation classes. I'm generally flying blind when it comes to importing all these classes so there may be some redundancy.

```
importClass(org.apache.xpath.XPathAPI);

importClass(javax.xml.parsers.DocumentBuilderFactory);

importClass(org.w3c.dom.Node);

importClass(org.w3c.dom.Element);

importClass(org.w3c.dom.NodeList);
```

Thereafter is became an exercise in wrapping the DOM functions in flow script.

Attached to this page should be my general purpose **jsUtils.js** file which contains all the necessary methods to carry out DOM manipulation. If anyone can see any obvious flaws, please let me know! To create a new document, pass the name of the document to the **newDocument** function. Adding attributes is as simple as calling **addAttribute**. If the nodeParent parameter is null, the attribute gets assigned to the document root.

The use of the **sessionManager** is interesting.

I couldn't for the life of me get a reference to the DOM if I used the **cocoon.session** object. I don't know why. I scoured all manner of places and found the use of the sessionManager, stuck the document in there, and then I was able to access the DOM from the pipeline. Maybe I missed something, but it works.

Some of the code is specific to my particular application but most of what's in jsUtils.js is generic enough to be used in your own projects. Let me know if you do and keep me informed of any improvements you make!!

```
var sXmlName = "TestDoc";

var xmlDoc = newDocument("hierarchy","description",Trim(sXmlName));

//Make its hierarchy number = 0 so we know its a newy

addAttribute(xmlDoc,null,"hrcyNumber",0);

var sessionManager = cocoon.getComponent("org.apache.cocoon.webapps.session.SessionManager");

var session = sessionManager.getSession(true);

session.setAttribute("xmlDoc", xmlDoc);

cocoon.sendPage("tree-menu");
```

The sitemap pipeline that process the evergrowing document looks like this (note the name of the session attribute):

```
<map:match pattern="internal/tree-menu">

<map:generate type="session-attr">

<map:parameter name="attr-name" value="xmlDoc"/>

</map:generate>

<map:serialize type="xml" />

</map:match>
```

I then render this up using some javascript tree widget functions to provide the user with a fairly snappy tree based XML representaiton of their document.