Flow

Future home of some really good FlowScript docs

- What is Flow?
  *An easy way to write complex Web applications with Cocoon
  *Complex multi-page interactions can be described easily as blocking function calls
- Why use Flow?
  *See above 🙂
- How do I use Flow?
  *Check the Flow samples page in current CVS.
  *Read a description of the Cocoon Flow architecture on Ovidiu's Weblog
  *Stefano's Linotype makes very good use of the Flow engine.
  *More to appear in this space
- What sorts of Cocoon-related objects are available to me in my FlowScript?
  *Look in src/java/org/apache/cocoon/components/flow/ and check out `JavaScriptInterpreter.java`. Notice how it binds Java classes to JavaScript objects.
  You can also access packages that aren't in java. by prefixing them with "Packages."
  *e.g. `user = new Packages.samples.flow.prefs.logic.User(login, password, firstName, lastName, email);` (see the prefs sample)
  *Useful for writing helper logic in Java
  *See also http://www.mozilla.org/rhino/scriptjava.html
- How can I send data back and forth between pages and the Flow?
  *Use sendPage() or sendPageAndWait()
  *The "bizData" object {"name": value} is just a Javascript object, which can be accessed using:
  *JPath Logicsheet
  *JPathTransformer
  *JXPathTemplate(Generator)
  *VelocityTransformer
  *Example: sendPageAndWait("mypipeline", {"name": value});

The JPathTransformer seems like the ideal thing to use in many cases. – TonyCollen

- What other languages can I write Flow in?
  *Javascript (part of Cocoon 2.1)
  *Java (based on the commercial product ATCT by Valare - install instructions and sources by Alex Krut for a Cocoon integration are available)
- Proposals for other implementations
  *Python would be interesting 🙂
    - Stackless Python

See also

- GettingStartedWithFlow
- WhatIsFlow
- DebugFlowScripts
- JavascriptForJavaProgrammers
- RhinoWithContinuations

or get these all as one printable page

You can also split up your flow code like so:

```
<map:flow language="JavaScript">
    <map:script src="script1.js"/>
    <map:script src="script2.js"/>
    <map:script src="script3.js"/>
    <map:script src="script4.js"/>
</map:flow>
```

Q: What Cocoon-specific objects are available in the Flow?

A: Interface to various Cocoon abstractions:

- cocoon
  *environment // property of type
- org.apache.cocoon.environment.Environment
  *parameters // JavaScript array of <not sure?>
  *request // property of type org.apache.cocoon.environment.Request
  *response // property of type org.apache.cocoon.environment.Response
  *session // property of type org.apache.cocoon.environment.Session

*context // property of type org.apache.cocoon.environment.Context
*componentManager // property of type
- org.apache.avalon.framework.ComponentManager
*`load(fileName)` // loads a script
*`createSession()` // attaches this flow script instance to session
*`removeSession()` // detaches this flow script instance from session
*action/input/output module interfaces:
*`callAction(type, source, parameters)`
*`inputModuleGetAttribute(type, attribute)`
*`outputModuleSetAttribute(type, attribute, value)`
- Flow API
*`sendPage(uri, bizData)` // call presentation layer to present bizData
*`uri` is a string to a URI which to redirect to. Usually this is within Cocoon.
*`bizData` is an object, usually to send multiple key->value pairs to the page. These are sent using JPath, and can be retrieved using the JPath Logicsheet, Transformer, or Generator. (See above)
*`sendPageAndWait(uri, bizData, timeToLive)` // call presentation layer to present bizData and wait for subsequent request
- action/input/output module interfaces:
*`act(type, src, param)`
*`inputValue(type, name)`
*`outputSet(type, name, value)`
*`outputCommit(type)`
*`outputRollback(type)`
- logging support:
*`print(args)` // prints args to standard out
*log – Provides access to cocoon logging system via its methods (error, debug, warn, info loglevels)
*`error(message)`
*`debug(message)`
*`warn(message)`
*`info(message)`
*See also: `logkit.xconf`

The info regarding how the flow functions were named (using `js*`) in Rhino has been retired. Check the Wiki diffs if you still need this info, but you probably won't. – TonyCollen

Q: How do I pass flowscript parameters to the sitemap?

A: Accessing flowscript parameters in the sitemap:

You create an object of class PipelineUtil and pass the parameters to this object along with the match pattern of the pipeline. Or you pass the parameters to cocoon.sendPage or cocoon.sendPageAndWait.

In the pipeline you access the parameter using <map:parameter name="flowparameter" value="{flow-attribute:urn}"/>

Here is an example:

---~~flowscript~~ ---

```
function enricheXML(urnforthexml){
   var enrichedXML = java.io.ByteArrayOutputStream();
   var pipeutil = cocoon.createObject(Packages.org.apache.cocoon.components.flow.util.PipelineUtil);
   pipeutil.processToStream("enricheXML", {"urn":urnforthexml}, enrichedXML);
   return enrichedXML.toString();
}
```

---~~sitemap~~ ---

```
<!-- ==Pattern for the XMLenrichement ====================================== -->
   <map:match pattern="enricheXML">
      <map:generate src="{request-param:uploadfile}" />
      <map:transform src="xsl/enrichement.xsl" >
         <map:parameter name="url" value="{request-param:uploadurl}" />
         <map:parameter name="urn" value="{flow-attribute:urn}" />
      </map:transform>
      <map:serialize type="xml" />
   </map:match>
```

---~~stylesheet~~ ---

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="xml"/>
   <xsl:param name="urn" />
   <xsl:param name="url" />

   <xsl:template match="/">
      <enriched>
         <URL><xsl:value-of select="$url"/></URL>
         <URN><xsl:value-of select="$urn"/></URN>
         <xsl:copy-of select="/"/>
      </enriched>
   </xsl:template>
</xsl:stylesheet>
```

Thanks to JanHinzmann