XSPAction

Description

To write an XSP Action you need to declarate the use of the built-in logicsheet *action*. In the header of your XSP please include:

```
xmlns:action="http://apache.org/cocoon/action/1.0"
```

Allowed Tags

Here is a description of all the tags and the parameters it uses:

- **<action:redirect-to>** - Redirects to a given URL.
  - **`uri`** - The URI to redirect-to (String)
  - **`session-mode`** - Keep session across redirect (boolean, default = true)
    `Example:` `<action:redirect-to uri="ToURI" session-mode="false"/>`
- **<action:set-result>** - Adds an entry in the action result map. Implicitly set the action status to successful
  *`name` - The entry name (String)
  *`value` - The entry value (String)
  `Example:` `<action:set-result name="MyName" value="MyValue"/>`
- **<action:get-result>** - Gets the value of an entry of the Action result map (previously set with `<action:set-result>`)
  ***name** - The entry name (String)
  `Example:` `<action:get-result name="MyName"/>`
- **<action:set-success>** - Sets the action status to successful
  *Does not have any parameter
  `Example:` `<action:set-success/>`
- **<action:set-failure>** - Sets the action status to failure _(child statements in the sitemap won't be executed)_.
  `Example:` `<action:set-failure/>`

Writing an action using XSP

This is some basic information that explains how to write an action using XSP. It might not be clean or perfect but it did work for Litrik

When you write your action in XSP, Cocoon will notice when the file has changed and recompile automagically.

Sitemap

Declaring the Action component

Your sitemap should contain the following declaration of the action component:

```
<map:actions>
  <map:action
    logger="sitemap.action.xsp-action"
    name="xsp-action"
    src="org.apache.cocoon.acting.ServerPagesAction"/>
</map:actions>
```

Note: Some other pieces of documentation use *serverpages* as the action name instead of *xsp-action*. I prefer to use *xsp-action* to avoid confusions of what it defines. Note that the name *serverpages* is already used by the XSP Generator. This is why I dont use it.

How to use in a Pipeline

And of course your sitemap should contain a pipeline that actually contains an action written in XSP. To use it we must use the <map:action>. The value of the *type* attribute must be *xsp-action* and the value of the *src* attribute will contain the full path to the XSP file.

```
<map:match pattern="test-of-xsp-action">
  <map:act type="xsp-action" src="my-xsp-action.xsp">
    <!-- This part is called when the action executes <action:set-success/> -->
    <map:read src="{my-result-filename}" mime-type="image/gif"/>
  </map:act>
  <!-- This part is called when the action executes <action:set-failure/> -->
  <map:generate type="serverpages" src="error.xsp"/>
  <map:serialize/>
</map:match>
```

Writing the Action

The *src* attribute of the <map:act> of the pipeline contains *my-xsp-action.xsp*. That is the XSP file that is actually implementing the action logic.

This is what *my-xsp-action.xsp* could look like:

```
<?xml version="1.0"?>
<xsp:page
  language="java"
  xmlns:action="http://apache.org/cocoon/action/1.0"
  xmlns:xsp="http://apache.org/xsp"
>
<dummypage>

  <xsp:logic>
  try
  {

    if( /* do some test here */ )
    {
      /* With <action:set-result/> we set return values
      ** that can be used in the calling pipeline */
      <action:set-result name="my-result-filename" value="GoodMorning.gif"/>
      <action:set-result name="some-other-result" value="5987"/>

      // You can use action:param to set the name and/or the value dynamically.
      <action:set-result>
        <action:param name="name"><xsp:expr>getName()</xsp:expr></action:param>
        <action:param name="value"><xsp:expr>getValue()</xsp:expr></action:param>
      </action:set-result>
    }
    else if( /* do some other test here */ )
    {
      <action:set-result name="my-result-filename" value="GoodAfternoon.gif"/>
    }
    else
    {
      throw new Exception() ;
    }

    // With <action:set-success/> we indicate that
    // everything went just fine
    <action:set-success/>
  }
  catch(Exception e)
  {
    e.printStackTrace() ;
    // With <action:set-failure/> we indicate that
    // something went wrong and that the action has failed
    <action:set-failure/>
  }
  </xsp:logic>

</dummypage>

</xsp:page>
```