Specialized Portlet

Table of contents

1 Introduction....2
2 flowscript.xml....2
3 Specialized Portlet Example....2

1. Introduction

The purpose of the Specialized Portlet is to cause certain conditions to occur so that the Analyzer can look a the messages and detemine if the Consumer handled the situation according to the spec. The Specialized Portlet generates markup with particular values that the Analyzer can look for in the message log.

The Specialized Portlet works by reading a file called `flow_script.xml` which tells it which JSP file to load. The JSP file actually contains the markup that the Specialized Portlet supplies on a getMarkupRequest. The `flow_script.xml` file allows test cases to be repeated a particular number of times.

Here is a sample of the `flow_script.xml` file:

2. flowscript.xml

The SpecializedPortlet reads an XML file, called `flow_script.xml`, which tells the portlet which tests to execute. The test cases are actually represented by JSP files. Here is a sample of the `flow_script.xml` file:

```
<flow>
  <case id="134">
    <pass repeatCount="2">
      <jspInclude name="test_case_1340_1.jsp" />
    </pass>
  </case>
</flow>
```

A Specialized Portlet JSP typically contains a form and has javascript logic that causes the form to be automatically submitted. This allows the Specialized Portlet to execute all the tests without further human intervention.

3. Specialized Portlet Example

Let's look at an example:

RP1340	When an activated portlet URL has specified the wsrp-navigationalState portlet URL parameter, the Consumer shall supply its value unchanged in the MarkupParams.navigationalState field.

Here is the JSP file for this test (line feeds have been added to improve readability):

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
```

```
<%@ taglib uri='/WEB-INF/tld/portlet.tld' prefix='portlet'%>
<portlet:defineObjects/>
<script>
setTimeout('wsrp_rewrite_reload()',5000);
function wsrp_rewrite_reload()
{
  // only the first instance of the portlet shall submit the form
  var form = document.forms.wsrp_test_submit_form;
  if((form.wsrp_test_counter.value == "0") &&
(form.wsrp_test_autoSubmit.value == "true"))
  {
    form.wsrp_test_autoSubmit.value = "false";
    form.wsrp_test_counter.value = "1";
    // callback function for JSP's which are including this javascript
    wsrp_test_do(form);
    form.submit();
  }
}
function wsrp_test_do(form)
{
}
</script>
<form name="wsrp_test_submit_form" method="post"
  action="wsrp_rewrite?
    wsrp-urlType=blockingAction&
    wsrp-mode=wsrp:view&
    wsrp-windowState=wsrp:normal&
    wsrp-secureURL=false&
    wsrpnavigationalState=wsrp_test_navigationalState
  /wsrp_rewrite
    &wsrp_test_expectedNavigationalState=wsrp_test_navigationalState">
  Test case #134.<br>
  <input type="hidden" name="wsrp_test_autoSubmit" value="true"></input>
  <input type="hidden" name="wsrp_test_currentCase"
    value="<%=(String)renderRequest.getAttribute("wsrp_test_currentCase")%>"></input>
  <input type="hidden" name="wsrp_test_passCounter"
    value="<%=(String)renderRequest.getAttribute("wsrp_test_passCounter")%>"></input>
  <input type="hidden" name="wsrp_test_counter" value="0"></input>
  <input type="submit" value="Submit"></input>
</form>
```

Notice that the rewrite URL contains navigationalState. Here is the Test Assertion code that examines the log:

```
private static final String CODEM_NAME_EXPECTED_NAVIGATIONAL_STATE =
"wsrp_test_expectedNavigationalState";
protected void processPerformBlockingInteractionRequest(Document reqDoc)
  throws WSIException, TransformerException {
```

```
  // find the interactionParams node
  Node iaParamsNode = NodeUtils.getNode(reqDoc, "interactionParams");
  if (iaParamsNode != null) {
    // find all form codemeters
    NodeList formParamNodes = NodeUtils.getNodes(iaParamsNode,
"formParameters");
    int formParamCount = formParamNodes.getLength();
    Node formParamNode;
    String formParamName, expectedNavStateValue = null;
    // go through the form codemeters until we find the one that
interests us (if any)
    for (int ii = 0; ii < formParamCount; ii++) {
      formParamNode = formParamNodes.item(ii);
      formParamName = NodeUtils.getAttributeValue(formParamNode, "name");
      if (CODEM_NAME_EXPECTED_NAVIGATIONAL_STATE.equals(formParamName)) {
        // find the value subnode
        Node valueNode = NodeUtils.getTextNode(formParamNode, "value");
        expectedNavStateValue =
NodeUtils.getSafeTextNodeValue(valueNode);
        break;
      }
    } // for ii
    if (expectedNavStateValue != null) {
      // compare the expected navigational state value to the one
actaully supplied
      // find the markupParams node
      Node markupParamsNode = NodeUtils.getNode(reqDoc, "markupParams");
      if (markupParamsNode != null) {
        // find the navigational state subnode
        Node navStateNode = NodeUtils.getTextNode(markupParamsNode,
"navigationalState");
        String navState = NodeUtils.getSafeTextNodeValue(navStateNode);
        if (expectedNavStateValue.equals(navState)) {
          pass();
        } else {
          fail(
            "Expected navigational state: "
              + expectedNavStateValue
              + "; supplied navigational state: "
              + navState);
        }
      } else {
        fail("Missing markupParams node");
      }
    } // expected navigational state value not null
  } // interaction codems not null
} // processPerformBlockingInteractionRequest()
```

The Test Assertion looks at the value of an interaction parameter for the value that the navigationalState should contain.