Programming Components that Use the Branch Handler

Using the branch handler in your component is quite easy.

1. Declare a member variable for the BranchHandler:

```
BranchHandler branchHandler = null;
```

2. Initialize the branch handler in your initialize() method based on your component's configuration:

```
branchHandler = BranchHandlerFactory.newInstance(config, this);
```

3. If you have some branch events which are required, check that they exist and throw an exception if they don't. This is also typically in your initialize() method:

```
if(!branchHandler.canEnqueueOrProcess("onPublish")) {
     throw new AspireException(this, "aspire.framework.FeedOne.missing-branch-with-pipelinemanager",
         "The FeedOne component needs to be configured with a branches/branch that specifies a pipeline
manager. Either the branches or branch tags are missing, or a pipeline manager is not specified with the
@pipelineManager attribute.");
}
```

4. For *new jobs or sub-jobs only*(!), you can use the [enqueue()] method to queue the job on a pipeline manager queue:

```
branchHandler.enqueue(j, "onPublish");
```

> ⚠ The above can only be done for new jobs or sub-jobs *only*. *Never* enqueue a job which is already being processed in the pipeline. If a job is enqueued on two pipeline manangers, it will be processed by two threads, causing unstable race conditions to occur.

If you ever need a single job to be processed by two threads simultaneously, create a sub-job. The sub-job can access the parent job's data object and operate on it, as necessary (assuming those operations are thread-safe or at least thread-separated).

Unit Testing with the Branch Handler

A special attribute is available on a branch which writes the job to a file, rather than to a pipeline manager:

```
<branches>
   <branch event="onPublish" writeToFile="testout/scanDirTest.out"/>
</branches>
```

This configuration is primarily for unit testing. A typical way to unit test with the branch handler would be to do something like this:

```
ScanDir s = new ScanDir();
   s.initialize(AXML.stringToDom(
       "<config>" +
        "      <fileNamePatterns>" +
        "         <include pattern=\".*.txt$\" />" +
        "      </fileNamePatterns>" +
        "     <branches>" +
        "         <branch event=\"onPublish\" writeToFile=\"testout/scanDirTest.out\"/>" +
        "      </branches> " +
        "   </config>"));

   AspireDocument doc = new AspireDocument();
   doc.add("fetchUrl", "file:testdata/scanDirTest1");
   Job j = new Job(doc, "Test-1");
   s.process(j);
   s.close();

   // Now check the testout/scanDirTest.out file for what it should contain
   // note that UnitTestHelper.compareFiles() is another good choice here
   assertTrue(UnitTestHelper.scanFileForRegex(new File("testout/scanDirTest.out"), "scanDirTest1-1/scanDirTest1-
1-1"));
   .
   .
   .
```

In the above example, the branch handler uses the "@writeToFile" attribute to write all of the sub-jobs produced by the ScanDir directory scanner to a file. This file can then be scanned or compared as appropriate for the appropriate sub jobs.