Build Script Generator XSLT 1.0 Component Specification

1.Design

The build script generator uses the incoming template and component-version objects to transform (XSLT) the template into a build file. Define the XSLT transformation that includes special tags to handle TopCoder component dependencies, external component dependencies, attribute value population, and sub templates.

In this design, the most important classes are BuildScriptGeneratorFactoryImpl and BuildScriptGeneratorImpl. BuildScriptGeneratorImpl is used to generate build script of the source component version information according the template XSLT transformation. BuildScriptGeneratorFactoryImpl is the factory to create BuildScriptGeneratorImpl instances, it also uses Template Selector to select corresponding Template instance in the hierarchy of a specified component version object, and then create a BuildScriptGenerator instance.

About the requirement 2.2.2:

First, this requirement is not clear enough. It said the sub templates are obtained using the name of the hierarchy placed within the sub template syntax, but how to know what is the parent template including the sub templates according the Template Selector component. IMO, one component may have multiple templates, but each of them is a standalone XSLT template (You can see the details in Template Loader component). That means we can generate the correct build xml file only using the corresponding template selected by Template Selector. And the XSLT sub templates can be easily implemented by using XSLT tag <xsl:import> or <xsl:include>.

I.E, if we need two templates for a component, one(build_java.xsl) is used to generate the standard JAVA build xml file and the other(build_jsp.xsl) is used to compile JSP code, so we place build_java.xsl under the “JAVA” hierarchy and place build_jsp.xsl under “JAVA/JSP” hierarchy. And we can write the extra code to compile JSP in jsp.xsl file, then copy the content of build_java.xsl to build_jsp.xsl and include jsp.xsl in the right place using the tag <xsl:include>. I think that’s what the sub templates really mean.

1.1Design Patterns

BuildScriptGeneratorImpl class and BuildScriptGenerator interface implement the strategy pattern.

BuildScriptGeneratorFactoryImpl implements the factory pattern. Factory pattern is used because various possible generator instances can be selected. For example, the user can implement generator using other technology rather than XSLT which is used by the default generator.

1.2Industry Standards

XML, XSLT.

1.3Required Algorithms

Most algorithms are quite trivial, the only slightly complex algorithm is the conversion of a ComponentVersion object into a DOM Node. The algorithm (in pseudo-code) is as described here:

1. Create a new Document, using a DocumentBuilderFactory
2. Create a 'component' tag, add it to the document
3. Set the attributes of the component tag for all the components attributes
4. Create a 'dependencies' tag, add it to the component tag
5. For each component dependency of the component:
    1. Create a 'dependent_component' tag, add it to the dependencies tag
    2. Set the attributes of the created tag
6. Create a 'external_dependencies' tag, add it to the component tag
7. For each external dependency:
    1. Create a 'external_dependency' tag, add it to the external dependencies tag
    2. Set the attributes of the created tag
8. Create a 'technology_types' tag, add it to the component tag
9. For each technology type:
    1. Create a 'technology_type' tag, add it to the technology types tag
    2. Set the attributes of the created tag
10. Return the document element of the document created in (1)

This algorithm will result in an xml tree according to the following DTD:

```
 <!ELEMENT component (dependencies, external_dependencies,
technology_types)>
 <!ATTLIST component

          id                  CDATA   #REQUIRED
          name                CDATA   #REQUIRED
          description         CDATA   #IMPLIED
          version             CDATA   #REQUIRED
          package             CDATA   #IMPLIED>
 <!ELEMENT dependencies (dependent_component*)>
 <!ELEMENT dependent_component EMPTY>
 <!ATTLIST dependent_component
          id                  CDATA   #REQUIRED
          name                CDATA   #REQUIRED
          description         CDATA   #IMPLIED
          version             CDATA   #REQUIRED
          package             CDATA   #IMPLIED>
 <!ELEMENT external_dependencies (external_dependency*)>
 <!ELEMENT external_dependency EMPTY>
```

```
<!ATTLIST external_dependency
          id                    CDATA   #REQUIRED
          name                  CDATA   #REQUIRED
          version               CDATA   #REQUIRED
          description           CDATA   #IMPLIED
          filename              CDATA   #REQUIRED>
<!ELEMENT technology_types (technology_type*)>
<!ELEMENT technology_type EMPTY>
<!ATTLIST technology_type
          id                    CDATA   #REQUIRED
          name                  CDATA   #REQUIRED
          description           CDATA   #IMPLIED
          deprecated_status     (true|false) #REQUIRED>
```

1.4Component Class Overview

BuildScriptGeneratorFactory:

This interface is the factory of BuildScriptGenerator. It defines methods to create BuildScriptGenerators. And all BuildScriptGenerator instances should be created via factory methods.

BuildScriptGenerator

This interface defines methods to generate scripts from the input stream which contains the component version information or the component version object.

BuildScriptGeneratorFactoryImpl

It is the default generator factory implementation in this component. It uses javax.xml.transform.TransformerFactory in Xalan to retriever javax.xml.transform.Transformer object to perform the real XSLT transformation. And It uses TemplateSelectionAlgorithm in Template Selector Component to select the template in template hierarchy according the component version object.

BuildScriptGeneratorImpl

This class is the default implementation of BuildScriptGenerator interface. It uses javax.xml.transform.Transformer to transform component version information to build script

1.5Component Exception Definitions

BuildScriptGeneratorException[Custom]

This exception extends exception and encapsulates all exceptions thrown in this component.

GeneratorCreationException[Custom]

This exception extends BuildScriptGeneratorException and will be thrown if fails to create generator in BuildScriptGeneratorFactory.

GenerationProcessException[Custom]

This exception extends BuildScriptGeneratorException and will be thrown if fails to generate scripts in BuildScriptGenerator.

NullPointerException:

This exception is thrown in various methods where null value is not acceptable. Refer to the documentation in Poseidon for more details.

1.6Thread Safety

This component is not thread-safe.

Because BuildScriptGeneratorImpl is not thread-safe, it uses javax.xml.transform.Transformer to perform the transformation which may not be used in multiple threads running concurrently.

However, BuildScriptGeneratorFactoryImpl is thread-safe, because instances of this class do not maintain any private state.

2.Environment Requirements

2.1Environment

java 1.3 or higher.

2.2TopCoder Software Components

Component Version Loader 1.0

ComponentVersion, ExternalComponentVersion and TechnologyType are defined in this component.

Template Loader 1.0

Template and TemplateHierarchy are defined in this component..

Template Selector 1.0

It will be used in BuildScriptGeneratorFactoryImpl to select the corresponding template of the specified ComponentVersion object.

Base Exception 1.0

All the custom exceptions extend BaseException in this component.

2.3Third Party Components

xalan-j 2.7.0 – XSLT support if java 1.3 is used, for java 1.4 and newer xalan is not necesary.

3.Installation and Configuration

3.1Package Name

com.topcoder.buildutility

3.2Configuration Parameters

None

3.3Dependencies Configuration

None

4.Usage Notes

4.1Required steps to test the component

Extract the component distribution.

Follow Dependencies Configuration.

Execute ‘ant test’ within the directory that the distribution was extracted to.

4.2Required steps to use the component

Follow demo.

4.3Demo

```
// Assume the ant build script template XSLT document is
// docs/script/build_ant.xsl, and the nant one is
// docs/script/build_nant.xsl.

// Prepare ComponentVersion object
ComponentVersion component= new ComponentVersion(1, "Build Script
Generator XSLT", null, "1.0");
component.addAttribute("package", "com.topcoder.buildutility");
// set technology types
component.addTechnologyType(new TechnologyType(1, "java", "java
component", false));
component.addTechnologyType(new TechnologyType(2, "ant", null, false));
component.addTechnologyType(new TechnologyType(3, "distribution", null,
false));
// set dependencies, first component dependencies
ComponentVersion dep = new ComponentVersion(2, "Template Selector", null,
"1.0");
dep.addAttribute("package", "com.topcoder.buildutility");
comp.addComponentDependency(dep);
// add an external dependency
ExternalComponentVersion ext = new ExternalComponentVersion(1, "xalan",
null, "2.7.0", "xalan_j_2_7_0.jar");
comp.addExternalComponentDependency(ext);

// creation of templates is beyond the scope of this component
// assume a TemplateLoader has been defined with a proper hierarchy
TemplateHierarchy hier = templateLoader.loadTemplateHierarchy(“demo”);
// somehow retrieve the ant template, exact implementation details are
// not relevant to this demo
Template ant = getNamedTemplate(hier, “ant”);
```

4.3.1Create BuildScriptGenerator instance to perform script generation.

```
BuildScriptGeneratorFactory fac = new BuildScriptGeneratorFactoryImpl();
// 1. create BuildScriptGenerator via input stream
fac.createGenerator(new FileInputStream("docs/script/build_ant.xsl"));
// 2. create BuildScriptGenerator via template
```

```
fac.createGenerator(ant);
// 3. create BuildScriptGenerator via template hierarchy
fac.createGenerator(hier, component);
```

4.3.2Generate script by BuildScriptGenerator

```
OutputStream out = new FileOutputStream(“build.xml”);
// 1. generate script from input stream to output stream
generator.generate(new FileInputStream("component.xml"), out);
// 2. generate script for component version object
generator.generate(component, out);
// 3. generate script as a DOM Document
DOMResult result = new DOMResult();
generator.generate(component, result);
// 4. generate script from a Source instance
generator.generate(new StreamSource(new FileInputStream
("component.xml")), result);
```

5.Future Enhancements

Add more template XSLT documents to support other scripts for component version.