[TOPCODER]
SOFTWARE

Document Generator 2.1 Requirements Specification

1. Scope

1.1 Overview

The Document Generator creates text files based on templates. It will be used to automatically generate build scripts for TopCoder components, driven by a template stored on the file system.

Version 2.1 supports the if-condition element in the template. Backward compatibility must be ensured.

1.2 Logic Requirements

1.2.1 File Driven Templates

File templates should be stored in a file on the file system or in a jar file.

1.2.2 Multiple Templates

Client must have the ability to work from multiple templates at once. For example, TopCoder uses both ant and nant, and for each has two different build files, one used internally and by members, one used by clients.

1.2.3 Template Fields

The templates may have a variety of fields to be filled in, which should be based on the template itself. It should be easy for the component client to see what fields (ideally with sample input) are used in a template.

1.2.4 Field with multiple entries

Some fields may allow for multiple entries. Examples of this include sections of the build file to deal with dependencies. A component may have none, one or many dependencies that need to be included in the build class path.

1.2.5 XML Input

Users of this component should be able to provide template data in the form of an XML input source, whose structure must be defined by the designer.

1.2.6 API Input

Users of this component should be able to determine and fill the fields of the template through the API.

1.2.7 If-Construct

Users should be able to specify if-constructs in the template, besides the currently supported loop construct. This corresponds to the <xsl:if> element in the XSL transformation.

The if-condition element consists of three parts: field name, operator and condition value. The component is required to support these operators: =, !=, <, >, <=, >=.

The template text within the if-construct will be written only if the condition is satisfied. For completeness, the API Input will support the if-element as well.

The example below is one possible way to specify it. Using this format is not required.

```
%loop:PROJECT{loop for each project}%
%if:PROJECT_TYPE='component'%
Component: %PROJECT_NAME%
%endif%
%loop:REVIEWER%
%REVIEWER_ROLE%: %REVIEWER_HANDLE% $%REVIEWER_PAYMENT%
```

```
%endloop%
Product Manager: %PM_HANDLE%
%endloop%
```

1.2.8 Special Characters

Special characters such as “<”, “>” in the template text should be escaped properly during the transformation. This is known as a bug in the previous version.

1.2.9 Documentation

Design diagrams should be updated according to the latest standards.

1.3 Required Algorithms

None

1.4 Example of the Software Usage

TopCoder generates a build file for each component as it goes in to development. This file is based on a template, and modified by hand to suit the particular component.

1.5 Future Component Direction

This component will be used in the future to allow TopCoder to maintain a single template, and always work from that whenever a build is executed at TopCoder, whether for development and design distributions, or for customer distributions.

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None

2.1.2 External Interfaces

This component should be executable from the command line, as well as via API.

2.1.3 Environment Requirements

- Development language: Java1.4
- Compile target: Java1.3, Java1.4

2.1.4 Package Structure

com.topcoder.util.file

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

Template location must be configurable.

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

No

3.2.2 TopCoder Software Component Dependencies:

Configuration Manager
Command Line Utility
**Please review the TopCoder Software component catalog for existing components that can be

used in the design.

3.2.3 Third Party Component, Library, or Product Dependencies:

None

3.2.4 QA Environment:

- Solaris 7
- RedHat Linux 7.1
- Windows 2000

3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines. Modifications to these guidelines for this component should be detailed below.

3.4 Required Documentation

3.4.1 Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification
- Test Plan

3.4.2 Help / User Documentation

- Design documents must clearly define intended component usage in the 'Documentation' tab of Poseidon.
- Implementations of the design must clearly translate the design documentation into the corresponding language format (i.e. java -> javadoc, .NET -> XML). Source code documentation must be sufficient to enable component usage without additional documentation.