1. Requirements Specification

2. Scope

2.1 Overview

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.

2.2 Logic Requirements

2.2.1 Define the XSLT format and translations for Nant and Ant..

2.2.2 Sub templates

Each template can optionally contain zero, one or more sub templates. A sub template is template that is embedded within another template. These templates are obtained using the name of the hierarchy placed within the sub template syntax. The named hierarchy is retrieved and uses Template Selector component to resolve the sub templates (the same as normal templates). The name of the hierarchy refers to the name of the first node in the hierarchy. One such example of this would be to place extra code in a build script to compile JSPs or ASP pages.

2.2.3 *Component Dependencies*

Allows developers to loop through the component dependencies adding them into the build files.

2.2.4 *External Component Dependencies*

Allows developers to loop through the external component dependencies adding them into the build files.

2.2.5 Attribute Value Populate

Allows developers to populate the attribute values from the database into the templates based on the attribute names.

2.2.6 Perform the translation from templates into build files

Implement the transformations to support TopCoder Ant and Nant build files. Other transformations to Makefiles should be possible, but not created.

2.3 Required Algorithms

None

2.4 Example of the Software Usage

This component can be used to create build scripts for all of TopCoder's components.

2.5 Future Component Direction

Integration into the web site to allow automated build script generation for all components.

3. Interface Requirements

3.1.1 Graphical User Interface Requirements

None

3.1.2 *External Interfaces*

The classes described below are the responsibility of other components and designers. The component can expect to see the public methods outlined below (the class designers are not limited to these methods but will provide them for this component).

```
package com.topcoder.buildutility.component;

/**
 * <p>
 * The component version represents a version of a component that was not created by
TopCoder and is not a part of TopCoder's component catalog.
 * </p>
 * <p>This class diagram for the component version class is by no means fully completed.
It is merely a stubbed
 * representation of the functionality such that other dependent components can know what
to expect.
 * The underlying implementation is left to the designer responsible for this class.</p>
 */
public class ComponentVersion {

        /**
         * <p>Identifier that uniquely distinguishes this component version from all
        others.</p>
         *
         * @return Required, long unique identifier
         */
            public Long getId() {…}

        /**
         * <p>Returns the name of the component.</p>
         *
         * @return Required, returns the name of the component
         */
            public String getName() {…}

        /**
         * <p>Returns a description of the component </p>
         *
         * @return Optionally, returns a description of the component
         */
            public String getDescription() {…}

        /**
         * <p>Returns the version number using TopCoder's defined build versioning format.
        Please consult the Build Versioning and Lifetime support documentation for a
        description of this string.</p>
         *
         * @return Requried, Version string.
         */
            public String getVersion() {…}

        /**
         * <p>Returns the vallue associated with the specified attribute</p>
         *
         * @param name Required, name of the attribute
         * @return String attribute value
         */
            public String getAttribute(String name) {…}

        /**
         * <p>Returns an array of strings that are the names of the attributes. Use the
        names and the getAttribute method to obtain the list of values corresponding to
        each name.</p>
         *
         * @return Required, cannot return null, must return an empty array if there are
        zero entries
         */
            public String[] getAttributeNames() {…}
```

```
        /**
         * <p>Returns a list of external components by version that this component depends
        upon.</p>
         *
         * @return Required, cannot be null (in such cases should be an empty array).
         */
            public com.topcoder.buildutility.component.ExternalComponentVersion[]
        getExternalDependencies() {…}

        /**
         * <p>Returns a list of other components from within TopCoder's component catalog
        that this component depends upon.</p>
         *
         * @return Required, list of components or an empty list, cannot be null.
         */
            public com.topcoder.buildutility.component.ComponenetVersion[]
        getDependencies() {…}

        /**
         * <p>Returns a list of the technology types associated with this component.</p>
         *
         * @return Required cannot be null, can ben an empty array.
         */
            public com.topcoder.buildutility.component.TechnologyType[]
        getTechnologyTypes() {…}
 }

/**
 * <p>
 * The external component version represents a version of a component that was not
created by TopCoder and is not a part of TopCoder's component catalog.
 * </p>
 * <p>This class diagram for the external component version class is by no means fully
completed.  It is merely a stubbed
 * representation of the functionality such that other dependent components can know what
to expect.
 * The underlying implementation is left to the designer responsible for this class.</p>
 */
public class ExternalComponentVersion {

        /**
         * <p>Returns the name of the external component version.</p>
         *
         * @return Returns the name of the external component.
         */
            public String getName__() {…}

        /**
         * <p>Optionally, returns a description of this external component.</p>
         *
         * @return null or a description of the external component
         */
            public String getDescription() {…}

        /**
         * <p>Returns the version of the external component. Note that the version
        numbering scheme depends on the external component provider and may or may not be
        the same as TopCoder's</p>
         *
         * @return Required, string representation of the version
         */
            public String getVersion() {…}

        /**
         * <p>Represents the name of the file used to include this component within our
        build scripts (typically a jar file).</p>
         *
         * @return Required, string file name.
         */
            public String getFileName() {…}
```

```
        /**
         * <p>Unique identifier used by our database layer to distinguish between
        external component versions.</p>
         *
         * @return Required, long unique identifier.
         */
            public Long getId() {…}
 }

/**
 * <p>This class diagram for the technology type class is by no means fully completed.
It is merely a stubbed
 * representation of the functionality such that other dependent components can know what
to expect.
 * The underlying implementation is left to the designer responsible for this class.</p>
*/
public class TechnologyType {

        /**
         * <p>Required, name of the technology..</p>
         *
        * @return Required, name of the technology.
         */
            public String getName() {..}

        /**
         * <p>Optional description of the technology</p>
         *
        * @return
         */
            public String getDescription() {…}
}

package com.topcoder.buildutility.template;

/**
 * <p>This class diagram for the template hierarchy class is by no means fully completed.
It is merely a stubbed
 * representation of the functionality such that other dependent components can know what
to expect.
 * The underlying implementation is left to the designer responsible for this class.</p>
 */
public class TemplateHierarchy {

    /**
     * <p>Returns the template hierarchy node's name.  The name of the node in the
    template hierarchy
     * is used as part of the template selection process.  The name corresponds to either
    technology types
     * or component attributes.</p>
     *
     *
     * @return The name of the node in the template hierarchy is used as part of the
    template selection process.  The name corresponds to either technology types or
    component attributes.
     */
        public String getName() {…}

    /**
```

```
     * <p>Each node in the template hierarchy can have zero to many template associations.
    An empty array must be returned, a null return value is not allowed..</p>
     *
     * @return Template[] Each node in the template hierarchy can have zero to many
    template associations. An empty array must be returned, a null return value is not
    allowed.
     */
        public com.topcoder.buildutility.template.Template[] getTemplates() {…}

    /**
     * <p>Returns all of the template hierarchy nodes directly below the current node in
    the hierarchy.</p>
     *
     * @return Returns all of the template hierarchy nodes directly below the current node
    in the hierarchy. An empty array must be returned if there are no children, a null
    value must not be returned.
     */
        public com.topcoder.buildutility.template.TemplateHierarchy[] getChildren() {…}
 }



/**
 * <p>This class diagram for the template class is by no means fully completed.  It is
merely a stubbed
 * representation of the functionality such that other dependent components can know what
to expect.
 * The underlying implementation is left to the designer responsible for this class.</p>
 */
public final class Template {

    /**
     * <p>Returns the name of the template. Each template name must uniquely identify that
    template (i.e. no two templates can have the same name).</p>
     *
     * @return Returns the name of the template.
     */
        public String getName() {…}

    /**
     * <p>Returns the description of this template. Descriptions are optionaly but they
    will help a great deal when developers are trying to determine the purpose of the
    template.</p>
     *
     * @return String Description of the temlpate
     */
        public String getDescription() {…}

    /**
     * <p>Returns the intended or suggested file name corresponding to the template.  This
    file name is more than likely not the
     * name used to store the content on the server side.  This file name was designed to
    be the suggested name when
```

```
         * the transformation occurs from the template to the actual file and the file is
        written in destination location.
         * The file name is required and cannot be null.</p>
         *
         * @return Suggested file name, cannot be null.
         */
            public String getFileName() {…}

        /**
         * <p>Returns the tempalte as a data stream. A template must contain data.</p>
         *
         * @return Returns the tempalte as a data stream.
         */
            public java.io.InputStream getData() {…}
}
```

3.1.3 Environment Requirements

- Development language: Java 1.4
- Compile target: Java 1.3, Java 1.4

3.1.4 Package Structure

com.topcoder.buildutility

4. Software Requirements

4.1 Administration Requirements

4.1.1 What elements of the application need to be configurable?

Please use good design principals to maximize future flexibility through configuration

4.2 Technical Constraints

4.2.1 Are there particular frameworks or standards that are required?

XSLT

4.2.2 TopCoder Software Component Dependencies:

Java Template Selector

**Please review the TopCoder Software component catalog for existing components that can be used in the design.

4.2.3 Third Party Component, Library, or Product Dependencies:

None

4.2.4 QA Environment:

- Windows 2000
- Windows 2003
- RedHat Linux 7.1
- Solaris 7

4.3 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.

4.4 Required Documentation

4.4.1 Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification
- XML Schema
- XSLT format

4.4.2 Help / User Documentation

- Design documents must clearly define intended component usage in the ‘Documentation’ tab of Poseidon