Software Documentation : Java Custom Component Dependency Extractor

This page last changed on May 12, 2008 by zjq.

1. Scope

1.1 Overview

A TopCoder component may have many dependencies, including other TopCoder components or third-party software. This component's task is to extract the dependencies for one or more components. The current version of the component will parse the build file of the component to extract the dependency information needed.

1.1.1 Version

1.0

1.2 Logic Requirements

1.2.1 Dependency Property

1.2.1.1 Dependency Type

- Internal : Other TopCoder components, generic or custom.
- External : Third-party softwares / components, system components, etc.

1.2.1.2 Dependency Category

- Compile dependency : The dependency is used when compiling the target component.
- Test dependency : The dependency is used when testing the target component.

1.2.2 Supported Languages

For this version, only Java and .NET (C#) components are required.

1.2.3 Supported Format of Build Files

1.2.3.1 Old Ant build.xml File

Refer to attached sample build file.

1.2.3.2 New Ant build-dependencies.xml File

Refer to attached sample build file.

1.2.3.3 NAnt default.build File

Refer to attached sample build file.

1.2.4 Dependency Content

A complete Component Dependency will contain its name, version, type, category and path. All the information can be extracted from build file. The relative path of the component dependency will not contain property (Ant property, NAnt property) variables.

1.2.5 Support Component Distribution Archive

The extractor will support component distribution archive file as input, since in SVN, and in catalog, the components are stored as a single archive file. The extractor will be able to automatically extract the

archive file to get the build file, then extract the dependency information from the extracted build file. The original archive must be left unchanged.

- Java : Archive file will be named like this: component_name-version.jar
- .NET : Archive file will be named like this: component_name-version.zip

1.2.6 Persistence

The component will provide persistence functionality. The extracted dependency information will be able to be persisted. The basic file persistence will be provided. The details of how the dependency information is persisted in the file is left to designer to determine. For example, XML can be used like this:

```
<component name="logging_wrapper" version="2.0.0" language="java" >
<dependency type="internal" category="compile" name="base_exception" version="2.0.0" path="../tcs/
lib/tcs/base_exception/2.0.0/base_exception.jar" />
<dependency type="internal" category="compile" name="typesafe_enum" version="1.1.0" path="../tcs/
lib/tcs/typesafe_enum/1.1.0/typesafe_enum.jar" />
<dependency type="internal" category="compile" name="object_formatter" version="1.0.0" path="../
tcs/lib/tcs/object_formatter/1.0.0/object_formatter.jar" />
<dependency type="external" category="compile" name="log4j" version="1.2.14" path="../tcs/lib/
third_party/log4j/1.2.14/log4j-1.2.14.jar" />
<dependency type="external" category="test" name="junit" version="3.8.2" path="../tcs/lib/
third_party/junit/3.8.2/junit.jar" />
</component>
```

1.2.7 Logging

The logging functionality is required, but can be turned off when the component is used.

- INFO level
  - Start to process a component
  - Extracted a dependency
  - End of processing a component
- WARN level
  - The extracted information is ambiguous
  - The extracted information is not well-formed
  - Cannot find expected bulid file from a archive file
- ERROR level
  - Exception catched
  - Other fatal error during extraction

1.2.8 Extract Dependency from Command Line

The extractor will be able to run as a standalone application. The parameters can be passed by command line options or configuration file. We may want to get the dependency information for all the components in the catalog, for now the components are stored as distribution archives in separated directories, so make sure the input configuration can handle this kind of scenario.

1.3 Required Algorithms

The algorithm for parsing the build file and extracting the dependency information will be fault-tolerant.

1.4 Example of the Software Usage

This component can be used to generate the dependency report for components. When we need to build a component, the Component Dependency Report Generator component can be used get all direct/ indirect dependencies for the component.

1.5 Future Component Direction

- Support more formats of build file.
- Support more languages (C++, VB, etc.).

- Support different build tools and so build file formats.

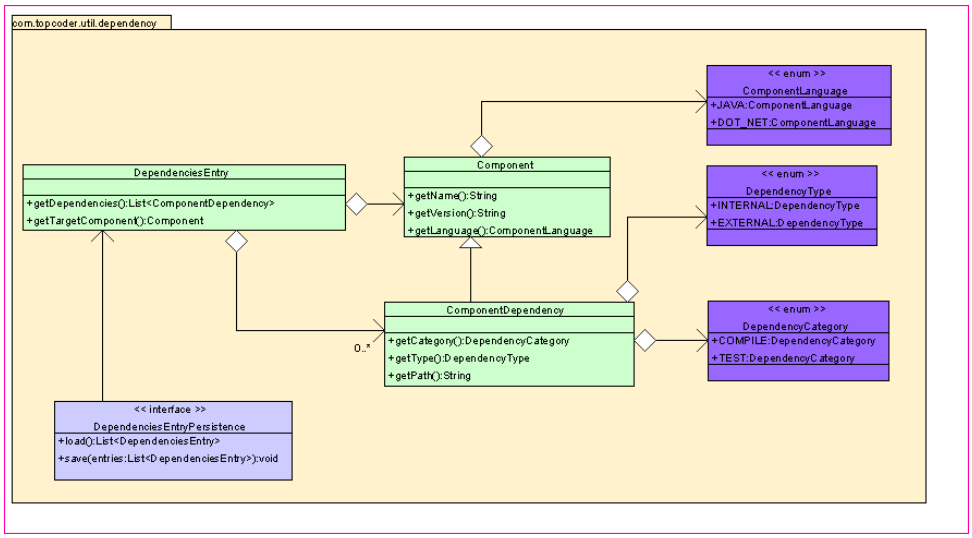
2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None.

2.1.2 External Interfaces

Design must adhere to the class diagram below.



2.1.3 Environment Requirements

- Development language: Java 1.5
- Compile target: Java 1.5 & 6

2.1.4 Package Structure

com.topcoder.util.dependency

com.topcoder.util.dependency.extractor

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

- Location of the build file(s) or distribution file(s) to parse.
- Location of persistence file.
- Dependency types to extract.
- Dependency categories to extract.
- Whether or not to log the extraction process.
- Whether or not to stop the extraction process when error occurs.

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

XML is required for the format of the configuration file.

3.2.2 TopCoder Software Component Dependencies:

- Command Line Utility 1.0
- Logging Wrapper 2.0
- Configuration API 1.0
- Compression Utility 2.0.1

**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
- Windows 2003

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 will be detailed below.

3.4 Required Documentation

3.4.1 Design Documentation

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

3.4.2 Help / User Documentation

- Design documents must clearly define intended component usage in the 'Documentation' tab of Poseidon.