Component Dependency Report Generator 1.0 Component Specification

1. Design

A TopCoder component may have many dependencies, including other TopCoder components or third-party software. This component's task is to generate the detailed dependency report for one or more components. The current version of the component depends on the Component Dependency Extractor component, and uses the extracted dependency information from Extractor component.

This component defines DependencyReportGenerator interface. It provides methods that allow to generate dependencies reports to various destination types (String, file or output stream). The caller can specify the list of components to be included in the report or generate report for all available components. The caller also can specify whether only direct dependencies must be present in the report or indirect dependencies also must be retrieved and included in it.

BaseDependencyReportGenerator is an abstract implementation of DependencyReportGenerator. It provides an implementation of all basic logic of DependencyReportGenerator interface that don't touch the exact report format. It provides implementation of retrieving indirect dependencies and filtering all dependencies by type and category.

In this component 3 concrete implementations of DependencyReportGenerator are provided: XmlDependencyReportGenerator, CsvDependencyReportGenerator and HtmlDependencyReportGenerator. They can be used to generate reports in XML, CSV and HTML formats correspondingly. HTML generator generates reports that have the most user-friendly appearance.

Also this component includes a standalone application. Its main class is DependencyReportGeneratorUtility. This utility can be used by the user to generate dependency reports from command line. It accepts configuration parameters from configuration file and command line arguments.

To generae XML report with proper indentation with JDK5, the "**indent-number**" attribute of **TransformerFactory** should be set.

See http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6296446

1.1 Design Patterns

Template method – BaseDependencyReportGenerator provides a template method generate() that uses an abstract method writeReport().

Strategy – DependencyReportGeneratorUtility can use various implementations of DependencyReportGenerator; it creates them via reflection.

1.2 Industry Standards

XML, CSV, HTML

1.3 Required Algorithms

1.3.1 *Generation of XML report*

This algorithm corresponds to the method XmlDependencyReportGenerator#writeReport(). It writes report in XML format to the given output stream.
The output XML report is generated with proper indentation that makes the report readable for the user. The output of this algorithm has the following format:

```
<?xml version="1.0"?>
<components>
  <component name="component1_name" version="component1_version"
language="component1_language">
    <dependency type="c1_dep1_type" category="c1_dep1_category" name="c1_dep1_name"
version="c1_dep1_version" path="c1_dep1_path />
```

```
    <dependency type="c1_dep2_type" category="c1_dep2_category" name="c1_dep2_name"
version="c1_dep2_version" path="c1_dep2_path />
  </component>
  <component name="component2_name" version="component2_version"
language="component2_language">
    <dependency type="c2_dep1_type" category="c2_dep1_category" name="c2_dep1_name"
version="c2_dep1_version" path="c2_dep1_path />
  </component>
</components>
```

Attributes “type”, “category” and “path” of “dependency” elements are optional (the user/application that uses this component can specify whether to include them in the report or not).
Value “componentX_language” of “language” attribute can be one of “java” and “dot_net”.
Value “cX_depX_type” of “type” attribute can be one of “internal” and “external”.
Value “cX_depX_category” of “category” attribute can be one of “compile” and “test”.

This is XML DTD for the provided report format:

```
<!ELEMENT components (component*)>
<!ELEMENT component (dependency*)>
<!ELEMENT dependency EMPTY>

<!ATTLIST component name CDATA #REQUIRED>
<!ATTLIST component version CDATA #REQUIRED>
<!ATTLIST component language (java|dot_net) #REQUIRED>
<!ATTLIST dependency type (internal|external) #IMPLIED>
<!ATTLIST dependency category (compile|test) #IMPLIED>
<!ATTLIST dependency name CDATA #REQUIRED>
<!ATTLIST dependency version CDATA #REQUIRED>
<!ATTLIST dependency path CDATA #IMPLIED>
```

Please see the demo section for a sample of such report.

The following steps produce the required output (developers of this component may use another, more efficient algorithm):

```
1. Create document build factory:
   DocumentBuilderFactory dbfac = DocumentBuilderFactory.newInstance();
2. Create document builder:
   DocumentBuilder docBuilder = dbfac.newDocumentBuilder();
3. Create new document:
   Document doc = docBuilder.newDocument();
4. Create root XML element:
   Element root = doc.createElement("components");
5. Add element to the document:
  doc.appendChild(root);
6. For each entry from entries do:
6.1. Create new subelement:
   Element component = root.createElement("component");
6.2. Set name attribute:
   component.setAttribute("name", entry.getTargetComponent().getName());
6.3. Set version attribute:
   component.setAttribute("version", entry.getTargetComponent().getVersion());
6.4. Set language attribute:
   component.setAttribute("language", (entry.getTargetComponent().getLanguage() ==
ComponentLanguage.JAVA) ? "java" : "dot_net");
6.5. For each dependency from entry.getDependencies() do:
6.5.1. Create new dependency element:
   dependencyElement = component.createElement("dependency");
6.5.2. If isDependencyTypeIncluded() then
6.5.2.1. Set type attribute:
```

dependencyElement.setAttribute("type", (dependency.getType() == DependencyType.INTERNAL) ? "internal" : "external");
6.5.3. If isDependencyCategoryIncluded() then
6.5.3.1. Set category attribute:
dependencyElement.setAttribute("category", (dependency.getCategory() == DependencyCategory.COMPILE) ? "compile" : "test");
6.5.4. Set name attribute:
dependencyElement.setAttribute("name", dependency.getName());
6.5.5. Set version attribute:
dependencyElement.setAttribute("version", dependency.getVersion());
6.5.6. If isDependencyPathIncluded() then
6.5.6.1. Set path attribute:
dependencyElement.setAttribute("path", dependency.getPath());
6.5.7. Add element to the document:
component.appendChild(dependencyElement);
6.6. Add element to the document:
root.appendChild(component);
7. Create transformer factory:
TransformerFactory transfac = TransformerFactory.newInstance();
7.1 Set the “indent-number” attribute:
transfac.setAttribute("indent-number", new Integer(2));
8. Create transformer:
Transformer trans = transfac.newTransformer();
9. Switch on the indentation flag:
trans.setOutputProperty(OutputKeys.INDENT, "yes");
10. Create string writer:
StringWriter sw = new StringWriter();
11. Create stream result:
StreamResult result = new StreamResult(sw);
12. Create DOM source:
DOMSource source = new DOMSource(doc);
13. Transform the document:
trans.transform(source, result);
14. Convert result stream to string:
String xmlString = sw.toString();
15. Create a text writer:
writer = new OutputStreamWriter(os);
16. Write text to the output stream:
writer.write(xmlString, 0, xmlString.length());
17…Flush Writer
writer.flush();

1.3.2 *Generation of CSV report*

This algorithm corresponds to the method CsvDependencyReportGenerator#writeReport(). It writes report in CSV format to the given output stream.
Each line of the output of this algorithm has the following format (no spaces and line breaks):

```
language-component_name-component_version,[dependency1_type]
[dependency1_category]dependency1_name-dependency1_version
[dependency1_path],[dependency2_type][dependency2_category]
dependency2_name-dependency2_version[dependency2_path],...
```

Elements "[dependecyX_type]", "[dependencyX_category]" and "[dependencyX_path]" are optional (the user/application that uses this component can specify whether to include them in the report or not).
Value of “language” can be “java” or “dot_net”.

Value of “dependencyX_type” can be “int” or “ext”.
Value of “dependencyX_category” can be “compile” or “test”.

Please see the demo section for a sample of such report.

The following steps produce the required output:

```
1. Create text writer:
   writer = OutputStreamWriter(os);
2. For each entry from entries do:
2.1. Create string builder:
   StringBuilder str = new StringBuilder();
2.2. Append language parameter:
   str.append((entry.getTargetComponent().getLanguage() == ComponentLanguage.JAVA) ? "java" :
"dot_net");
2.3. Append delimiter:
   str.append('-');
2.4. Append name parameter:
   str.append(entry.getTargetComponent().getName());
2.5. Append delimiter:
   str.append('-');
2.6. Append version parameter:
   str.append(entry.getTargetComponent().getVersion());
2.7. For each dependency from entry.getDependencies() do:
2.7.1. Append delimiter:
   str.append(',');
2.7.2. If isDependencyTypeIncluded() then
2.7.2.1. Append type parameter:
   str.append((dependency.getType() == DependencyType.INTERNAL) ? "[int]" : "[ext]");
2.7.3. If isDependencyCategoryIncluded() then
2.7.3.1. Append category parameter:
   str.append((dependency.getCategory() == DependencyCategory.COMPILE) ? "[compile]" : "[test]");
2.7.4. Append name parameter:
   str.append(dependency.getName());
2.7.5. Append delimiter:
   str.append('-');
2.7.6. Append version parameter:
   str.append(dependency.getVersion());
2.7.7. If isDependencyPathIncluded() then
2.7.7.1. Append char:
   str.append('[');
2.7.7.2. Append path parameter:
   str.append(dependency.getPath());
2.7.7.3. Append char:
   str.append(']');
2.8. Append CRLF:
   str.append("\r\n");
2.9. Get string from string builder:
   string = str.toString();
2.10. Write string to output stream:
      writer.write(string, 0, string.length());
2.11…Flush Writer
        writer.flush();
```

1.3.3 *Generation of HTML report*

This algorithm corresponds to the method HtmlDependencyReportGenerator#writeReport(). It writes report in HTML format to the given output stream.

Indentation is not used while generating the HTML document. The user must use a browser to read reports in this format.
The output of this algorithm has the following format (indentation and line breaks are provided for clarity only):

```
<html>
<head>
    <title>Component Dependency Report</title>
</head>
<body>
    <table border="1">
        <tr align="center">
            <td colspan="3">Component</td>
            <td colspan="5">Dependencies</td>
        </tr>
        <tr align="center">
            <td>Language</td>
            <td>Name</td>
            <td>Version</td>
            <td>Name</td>
            <td>Version</td>
            <td>Type</td>
            <td>Category</td>
            <td>Path</td>
        </tr>
        <tr>
            <td rowspan="2">component1_language</td>
            <td rowspan="2">component1_name</td>
            <td rowspan="2">component1_version</td>
            <td>c1_dep1_name</td>
            <td>c1_dep1_version</td>
            <td>c1_dep1_type</td>
            <td>c1_dep1_category</td>
            <td>c1_dep1_path</td>
        </tr>
        <tr>
            <td>c1_dep2_name</td>
            <td>c1_dep2_version</td>
            <td>c1_dep2_type</td>
            <td>c1_dep2_category</td>
            <td>c1_dep2_path </td>
        </tr>
        <tr>
            <td rowspan="1">component2_language</td>
            <td rowspan="1">component2_name</td>
            <td rowspan="1">component2_version</td>
            <td colspan="5"></td>
        </tr>
    </table>
</body>
</html>
```

Columns “Type”, “Category” and “Path” are optional in the generated report (the user/application that uses this component can specify whether to include them in the report or not). Thus values of “colspan” attributes must be adjusted properly by this algorithm. The format above shows how to process components with no dependencies (see “component2_XXX”).
Value “componentX_language” can be one of “Java” and “.NET”.
Value “cX_depX_type” can be one of “internal” and “external”.
Value “cX_depX_category” can be one of “compile” and “test”.

Please see the demo section for a sample of such report.

The following steps produce the required output:

1. Create text writer:
   writer = OutputStreamWriter(os);

```
2. Variable for the number of dependency columns in the table:
   dependenciesColsNum = 2;
3. If isDependencyTypeIncluded() then dependenciesColsNum++;
4. If isDependencyCategoryIncluded() then dependenciesColsNum++;
5. If isDependencyPathIncluded() then dependenciesColsNum++;
6. Create string builder:
   StringBuilder str = new StringBuilder();
7. Append tags:
   str.append("<html><head><title>Component Dependency Report</title></head><body><table
border=\"1\"><tr align=\"center\"><td colspan=\"3">Component</td><td colspan=\"");
8. Append dependency columns number:
   str.append(dependenciesColsNum);
9. Append tags:
   str.append("\">Dependencies</td></tr><tr
align=\"center\"><td>Language</td><td>Name</td><td>Version</td><td>Name</td><td>Version</td
>");
10. If isDependencyTypeIncluded() then
10.1. Append type caption tag:
   str.append("<td>Type</td>");
11. If isDependencyCategoryIncluded() then
11.1. Append category caption tag:
   str.append("<td>Category</td>");
12. If isDependencyPathIncluded() then
12.1. Append path caption tag:
   str.append("<td>Path</td>");
13. Append closing tag:
   str.append("</tr>");
14. Convert HTML header to string:
   string = str.toString();
15. Write header to output stream:
   writer.write(string, 0, string.length());
16. For each entry from entries do:
16.1. Create string builder:
   str = new StringBuilder();
16.2. Get number of dependencies for the current component:
   dependencyNum = entry.getDependencies().size();
   If dependencyNum = 0 then dependencyNum = 1;
16.3. Append tags:
   str.append("<tr><td rowspan=\"" + dependencyNum + "\">");
16.4. Append tags:
   str.append((entry.getTargetComponent().getLanguage() == ComponentLanguage.JAVA) ? "Java" :
".NET");
16.5. Append tags:
   str.append("</td><td rowspan=\"" + dependencyNum + "\">" +
entry.getTargetComponent().getName() + "</td><td rowspan=\"" + dependencyNum + "\">"+
entry.getTargetComponent().getVersion() +"</td>");
16.6. For each dependency from entry.getDependencies() do:
16.6.1. If not first dependecy then str.append("<tr>");
16.6.2. Append tags:
   str.append("<td>" + dependency.getName() + "</td><td>" + dependency.getVersion() + "</td>");
16.6.3. If isDependencyTypeIncluded() then
16.6.3.1. Append type tags:
   str.append("<td>" + ((dependency.getType() == DependencyType.INTERNAL) ? "internal" :
"external") + "</td>");
16.6.4. If isDependencyCategoryIncluded() then
16.6.4.1. Append category tags:
   str.append("<td>" + ((dependency.getCategory() == DependencyCategory.COMPILE) ? "compile" : "test") + "</td>");
16.6.5. If isDependencyPathIncluded() then
16.6.5.1. Append path tags:
   str.append("<td>" + dependency.getPath() + "</td>");
16.6.6. Append closing tag:
   str.append("</tr>");
16.7. If entry.getDependencies().size() = 0 then
16.7.1. Append tags for no-dependencies component:
   str.append("<td colspan=\"" + dependenciesColsNum + "\"></td></tr>");
16.8. Get string from string builder:
   string = str.toString();
16.9. Write string to output stream:
   writer.write(string, 0, string.length());
17. Set footer string:
   string = "</table></body></html>";
18. Write footer to output stream:
      writer.write(string, 0, string.length());
19…Flush Writer
     writer.flush();
```

1.3.4 *Retrieving the list of direct/indirect dependencies for a component*

1.3.4.1 The algorithm to retrieve direct dependencies corresponds to the method BaseDependencyReportGenerator# processDirectDependencies ().

```
1. If cachedDirectResults.containsKey(componentId), return
   DependenciesEntry entry = dependencies.get(componentId);
     If entry == null , return
2. Create a new List< ComponentDependency > filteredDependenciesList
3. For each ComponentDependency in  entry.getDependencies().
      Check whether isAllowedDependency(ComponentDependency)
      If allowed, then add it to filteredDependenciesList
4. Put to cache:
      cachedDirectResults.put(componentId, new DependenciesEntry(entry.getTargetComponent(), filteredDependenciesList));
```

The direct dependencies are subset of indirect dependencies, so processIndirectDependencies() will call processDirectDependencies() first to get direct dependencies, and then merge the indirect dependences.

1.3.4.2 The algorithm to retrieve indirect dependencies corresponds to the method BaseDependencyReportGenerator#processIndirectDependencies().

This is a recursive algorithm. It uses progressComponents collection to detect circular component dependencies and throws an exception when detects them. If there is no entry for the component with the given ID, this algorithm just does nothing. Otherwise it first puts all direct dependencies to the result list, and then recursively processes indirect dependencies of dependency components and adds all indirect dependencies to the result list. This algorithm ensures that duplicate dependencies don't appear in the result list. The result dependencies list is put to the cachedResults collection. The caller method can then extract this list from the collection.

Required steps for this algorithm:

```
1. If progressComponents contains componentId then throw an exception (circular dependency error).
   Before throw exception, clear the progressComponents collection, clear the cache
2. Get cached result:
   result = cachedResults.get(componentId);
3. If result != null then return.
4. processDirectDependencies ()
5. Get entry for the specified component from cachedDirectResults:
  entry = cachedDirectResults.get(componentId);
  If entry = null then return;
6. Add component ID to the list of component being processed:
   progressComponents.add(componentId);
7. Create a result list of dependencies:
   resList = entry.getDependencies();//This is a shollow copy of the direct dependencies
~~8. For each dependency from entry.getDependencies() do:~~
~~8.1. Check whether dependency can be added to the report:~~
~~   allowed = isAllowedDependency(dependency);~~
~~8.2. If allowed then~~
~~8.2.1. Add dependency to the result list:~~
~~   resList.add(dependency);~~
9. Get the number of direct dependencies:
   directDepNum = resList.size();
10. For i = 0..directDepNum-1 do:
10.1. Process dependencies of dependency component:
   processIndirectDependencies(<ID for resList.get(i) component, see section 1.3.6>);
10.2. Get indirect dependencies of dependency component:
   dependencyComponent = cachedResults.get(<ID for resList.get(i) component, see section 1.3.6>);
10.3. If dependencyComponent != null then
10.3.1. Get the list of dependencies:
   indirectDependencies = dependencyComponent.getDependencies();
10.3.2. For each indirectDependency from indirectDependencies do:
10.3.2.1. Check whether dependency is duplicate:
   isDuplicate = isInDependencyList(resList, indirectDependency);
10.3.2.2. If not isDuplicate then
10.3.2.2.1. Add dependency to the result list:
   resList.add(indirectDependency);
11. Create a result dependency entry:
   result = new DependenciesEntry(entry.getTargetComponent(), resList);
12. Put result entry to the cache:
   cachedResults.put(componentId, result);
13. Remove component ID from the list:
   progressComponents.remove(componentId);
```

1.3.5 *Execution of standalone application*

This algorithm corresponds to the static method DependencyReportGeneratorUtility#main().
It read the configuration parameters from configuration file, extracts them from the command line arguments, reads dependency entries from persistence, generates a dependency report and writes it to the standard output or to the specified file.

Please see sections 3.2.2 and 3.2.3 of CS for details about configuration parameters and supported command line switches and arguments.

This method MUST NOT throw any exception. Instead it must print out the detailed error explanation to the standard output and terminate.

Required steps for this algorithm:

```
1. Create a command line utility:
   CommandLineUtility commandLineUtility = new CommandLineUtility();
2. For each command line switch (see CS 3.2.3) do:
2.1. Create switch:
   Switch switch = new Switch(...);
2.2. Add switch to the command line utility:
   commandLineUtility.addSwitch(switch);
3. Parse command line arguments:
   commandLineUtility.parse(args);
4. If error occurred while parsing the arguments, print it to System.out and return.
5. If user requests the help (with -h, -? or -help) then
5.1. Get command line usage string:
   usageString = commandLineUtility.getUsageString();
5.2. Print string to standard output:
   System.out.println(usageString);
5.3. Return.
6. Create a configuration file manager:
   ConfigurationFileManager manager = new ConfigurationFileManager();
7. Get configuration file name (use default if not specified):
   configFileName = commandLineUtility.getSwitch(...).getValue();
   If the config file specified by switch does not exist, use default: configFileName = “config.xml”
8. Load configuration file (print warning message if file doesn't exist and use default configuration
values from the section 3.2.2 for all configuration parameters mentioned below):
   manager.loadFile("com.topcoder.util.dependency.report.utility.DependencyReportGeneratorUtility",
filePath);
9. Get configuration for this component (only if configuration file exists):
   config =
manager.getConfiguration("com.topcoder.util.dependency.report.utility.DependencyReportGenerator
Utility").getChild(“com.topcoder.util.dependency.report.utility.DependencyReportGeneratorUtility”);
10. If config is null (and configuration file exists) then print error message and return.
11.
   If “-pclass” switch is present:
      persistenceClassName = commandLineUtility.getSwitch(...).getValue();
   Else Get persistence class name from config (use default if not specified):
      persistenceClassName = config.getPropertyValue(...);
12. Get persistence configuration from config (use default is not provided):
   persistenceConfig = config.getChild(...);
13. If input file information is provided in the command line (-i switch) and persistenceClassName is
DefaultXmlDependenciesEntryPersistence or BinaryFileDependenciesEntryPersistence then
13.1. Get input file name:
   inputFileName = commandLineUtility.getSwitch(...).getValue();
13.2. Put inputFileName to the persistence configuration:
   persistenceConfig.setPropertyValue(...);
14. Create a persistence:DependenciesEntryPersistence via reflection. Use persistenceClassName
to locate the class and persistenceConfig as the only constructor argument.
15. If “-f” or “-gclass” switch is present:
      generateClassName  = commandLineUtility.getSwitch(...).getValue();
   Else Get generator class name from config (use default if not specified):
      generatorClassName = config.getPropertyValue(...);
16. Get generator configuration from config (use default if not provided):
   generatorConfig = config.getChild(...);
17. If dependecy types were specified in the command line (-dtype switch) then
17.1. Get allowed dependency types string:
   allowedDependencyTypes = commandLineUtility.getSwitch(...).getValue();
17.2. Put allowedDependencyTypes to generatorConfig:
   generatorConfig.setPropertyValue(...);
18. Repeat step 17 for dependency categories (-dcat switch).
19. Create a generator:DependencyReportGenerator via reflection. Use generatorClassName to
locate the class. Use persistence and generatorConfig as constructor arguments.
20. If output file name is provided in the command line (-o switch) then
20.1. Get output file name:
   outputFileName = commandLineUtility.getSwitch(...).getValue();
21. Else get output file name from config (use null if not specified):
   outputFileName = config.getPropertyValue(...);
22. Get the list of components to be included in the report (-id switch; set to null if not specified):
   componentsStr = commandLineUtility.getSwitch(...).getValue();
23. If componentsStr != null then transform it to components:List<String>.
24. If "-indirect"/"-noindirect" switch is specified, then override indirect configuration parameter.
25. Else get indirect parameter from config (use default if not specified):
   indirect = config.getPropertyValue(...);
26. If outputFileName = null then
26.1. Check whether can write to the standard output:
   stdout = config.getPropertyValue(...);
26.2. If not stdout then print error message and return.
26.3. Else
26.3.1. String output;
26.3.2. If componentsStr != null then output = generate(components, indirect);
26.3.3. Else output = generateAll(indirect);
26.3.4. System.out.println(output);
27. Else
27.1. If componentsStr != null then generate(components, outputFileName, indirect);
27.2. Else generateAll(outputFileName, indirect);
```

1.3.6 *Generation of component IDs*

This component uses the identifiers for components in the following format:

```
language-component_name-component_version

E.g. java-logging_wrapper-2.0.0
     dot_net-id_generator-1.1.1
```

“language” can be “java” or “dot_net”. The user can pass such IDs to the generate() methods of DependencyReportGenerator implementations. Also these IDs are used internally when storing information about component direct/indirect dependencies in maps.

1.4 Component Class Overview

DependencyReportGenerator [interface]

This interface must be implemented by classes that can generate component dependency reports. It provides methods for generating reports for the specified components or all components from the given dependency list. To be compatible with DependencyReportGeneratorUtility implementations of this interface must have constructors that accepts two parameters: (dependencies:List<DependencyEntry>, configuration:ConfigurationObject) and (persistence: DependenciesEntryPersistence, configuration:ConfigurationObject). This interface provides methods that allow generating reports to various destinations: OutputStream, file and String. Also implementations of DependencyReportGenerator must support retrieving of indirect dependencies (dependencies of dependencies) for each component and including them in the report.

BaseDependencyReportGenerator [abstract]

This is a base implementation of DependencyReportGenerator. It provides the basic logic of each generator. This class provides template method overloads generate() and generateAll()

that use abstract method writeReport() which should be implemented by subclasses. Thus subclasses should not worry about writing report data to various destination types and should only be able to write report to any OutputStream. Also BaseDependencyReportGenerator extracts indirect dependencies from the given dependency entry list and filters dependencies by type and category. Thus subclasses in writeReport() accept all dependencies that must be included in the report. Results of direct and indirect dependency searches are cached to increase the performance of this component. This class checks for circular dependencies when searching indirect dependencies of the component. BaseDependencyReportGenerator is configured with Configuration API object that can hold information about what dependency types and categories must be included in the report. Also configuration can hold information about what optional fields must be included in the report. Subclasses can access this information with use of methods isDependencyXXIncluded().

XmlDependencyReportGenerator

This dependency report generator writes reports in XML format. It extends BaseDependencyReportGenerator, thus supports generating reports to various destination types, processing indirect dependencies, filtering dependencies by type and category. Also this class supports optional including of dependency type, category and path to the report. This class just overrides writeReport() method of BaseDependencyReportGenerator. It doesn't have additional configuration parameters. Indentation is used in the output XML reports, thus such reports can be easily read by the user without any additional tools. Please see CS for the detailed description of report format used by this component.

HtmlDependencyReportGenerator

This dependency report generator writes reports in HTML format. It extends BaseDependencyReportGenerator, thus supports generating reports to various destination types, processing indirect dependencies, filtering dependencies by type and category. Also this class supports optional including of dependency type, category and path to the report. This class just overrides writeReport() method of BaseDependencyReportGenerator. It doesn't have additional configuration parameters. HTML format (unlike XML and CSV formats) is the most user-oriented report format. Reports in HTML format are more comfortable and easy-to-read. HTML tables are used to combine the dependency information in the most suitable manner. Please see CS for the detailed description of the report format used by this component.

CsvDependencyReportGenerator

This dependency report generator writes reports in CSV format. It extends BaseDependencyReportGenerator, thus supports generating reports to various destination types, processing indirect dependencies, filtering dependencies by type and category. Also this class supports optional including of dependency type, category and path to the report. This class just overrides writeReport() method of BaseDependencyReportGenerator. It doesn't have additional configuration parameters. Please see CS for the detailed description of report format used by this component.

DependencyReportGeneratorUtility

This is the main class of standalone application that can be used for generating component dependency reports from the command line. It reads all configuration data from the configuration file. Some of this data can be overridden by the user with use of command line switches and arguments. By default, this utility uses DefaultXmlDependenciesEntryPersistence to get the list of direct component dependencies and XmlDependencyReportGenerator to write the report to the standard output. But the user can specify another DependenciesEntryPersistence and DependencyReportGenerator implementation to be used and some file where report must be written to. Please see section 3.2.3 for the full list of configuration parameters and command line switches supported by this component.

1.5 Component Exception Definitions

DependencyReportGenerationException

This exception is the base for all other custom exceptions defined in this component. It can also be thrown by implementations of DependencyReportGenerator when I/O or persistence error occurs while generating the dependency report (in such cases this exception is used as a wrapper for other exceptions).

CircularComponentDependencyException
This exception is thrown by implementations of DependencyReportGenerator when circular component dependency is found. This error can occur only when the user tries to retrieve indirect dependencies of a component.

ComponentIdNotFoundException
This exception is thrown by implementations of DependencyReportGenerator when the component with the given ID cannot be found in the input dependencies entry list. Component ID includes information about component language, name and version. This error can occur only when generate() method is used, not generateAll().

DependencyReportConfigurationException
This exception is thrown by BaseDependencyReportGenerator and its subclasses when error occurred while reading the configuration (e.g. when configuration property is missing or has invalid format).

1.6 Thread Safety

This component is not thread safe. All its classes are not thread safe too. Implementations of DependencyReportGenerator can be not thread safe. The user must access BaseDependencyReportGenerator subclasses from a single thread or synchronize all calls to their methods. DependencyReportGeneratorUtility serves as the main class of the standalone application. This application works in a single thread. But the user must be careful when executing multiple instances of this application if they use the same persistence or report destination file name.

2. Environment Requirements

2.1 Environment

Development language: Java 1.5
Compile target: Java 1.5 & 6
QA Environment: Solaris 7, RedHat Linux 7.1, Windows 2000, Windows 2003

2.2 TopCoder Software Components

Command Line Utility 1.0 – is used by standalone utility to parse command line arguments.

Component Dependency Extractor 1.0 – defines component dependency entities and persistence.

Base Exception 2.0 – is used for exceptions handling.

Configuration API 1.0 – is used while configuring classes of this component.

Configuration Persistence 1.0.1 – is used by standalone utility when reading configuration file.

NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Name

com.topcoder.util.dependency.report – the main package.
com.topcoder.util.dependency.report.impl – implementations of DependencyReportGenerator.

com.topcoder.util.dependency.report.utility – standalone report generator application.

3.2 Configuration Parameters

3.2.1 *Configuration of BaseDependencyReportGenerator*

The following table describes the structure of ConfigurationObject passed to the constructor of BaseDependencyReportGenerator.

Parameter	**Description**	**Values**
dependency_types	The list of dependency types those must be included into the generated report. The semicolon-separated list of case-insensitive DependencyType values. Default is "internal;external".	String. Not empty. (case insensitive) Optional.
dependency_categories	The list of dependency categories those must be included into the generated report. The semicolon-separated list of case-insensitive DependencyCategory values. Default is "compile;test".	String. Not empty. (case insensitive) Optional.
include_dependency_type	Indicates whether information about dependency type must be included in the generated report. Default is "true".	String. "true" or "false". (case sensitive) Optional.
include_dependency_category	Indicates whether information about dependency category must be included in the generated report. Default is "true".	String. "true" or "false" (case sensitive). Optional.
include_dependency_path	Indicates whether information about dependency component path must be included in the generated report. Default is "false".	String. "true" or "false" (case sensitive). Optional.

3.2.2 *Configuration file of DependencyReportGeneratorUtility*

Configuration of DependencyReportGeneratorUtility consists of two parts: data stored in the configuration file and command line parameters. Both of them are optional. The full configuration can be stored in configuration file only (i.e. utility can successfully work with no command line arguments specified), thus command line parameters are used just to override or complement the parameters from the configuration file. This section describes the configuration file parameters. See section 3.2.3 for the command line parameters.

The following table describes the structure of ConfigurationObject that is retrieved with use of Configuration Persistence from the configuration file of the standalone utility. Note that configuration file should have a <Config> element with name "com.topcoder.util.dependency.report.utility. DependencyReportGeneratorUtility". Its properties are described below:

Parameter	**Description**	**Values**
persistence_class	The full class name of the DependenciesEntryPersistence implementation that is used by the utility for retrieving dependencies list. Can be specified or overridden in the command line. Default is "com.topcoder.util.dependency.persistence. DefaultXmlDependenciesEntryPersistence".	String. Not empty. Optional.

persistence_config	The configuration for the used DependenciesEntryPersistence implementation. See Component Depenedency Extractor component docs for details. Some attributes of this object can be specified or overridden in the command line. Default is empty instance of DefaultConfigurationObject.	ConfigurationObject. Optional.
generator_class	The full class name of the DependencyReportGenerator implementation. Its instance is used while generating the report and writing it to the file. Can be specified or overridden in the command line. Default is "com.topcoder.util.dependency.report.impl. XmlReportGenerator".	String. Not empty. Optional.
generator_config	The configuration for the used DependencyReportGenerator implementation. Some attributes of this object can be specified or overridden in the command line. Default is empty instance of DefaultConfigurationObject.	ConfigurationObject. Optional.
stdout	Indicates whether report must be printed to the standard output or to the file. Note that if this parameter is "true" and report_file_name is provided then report is still written to the file. Default is "true".	String. "true" or "false". (case sensitive) Optional.
report_file_name	The name of the file where report is written. Can be specified or overridden in the command line.	String. Not empty. Optional.
indirect	Indicates whether indirect dependencies must be included into report. This parameter can be overridden in the command line. Default is "true".	String. "true" or "false". (case sensitive) Optional.

3.2.3 *DependencyReportGeneratorUtility command line parameters*

The following table describes the command line switches and arguments those are supported by DependencyReportGeneratorUtility standalone application.

Switch and arguments	**Description**
-c <file_name>	Optional. The name of the configuration file for this utility. This file is read with use of Configuration Persistence component. The structure of this file is described in the section 3.2.2. Default is "config.xml". Note that configuration file is optional and the utility can work when it doesn't exist.
-pclass <class_name>	Optional. The full class name of the DependenciesEntryPersistence implementation to be used. This switch specifies or overrides the parameter "persistence_class" from the section 3.2.2. DefaultXmlDependenciesEntryPersistence is a default persistence implementation.

-i <file_name>	Optional. The name of the input file that is used by DependenciesEntryPersistence implementation to load the list of dependency entries. This switch is not ignored only if DefaultXmlDependenciesEntryPersistence or BinaryFileDependenciesEntryPersistence is used as a persistence implementation. This switch overrides the file name specified in “persistence_config” mentioned in the section 3.2.2. Please see Component Dependency Extractor component docs for details on how to provide this file name with ConfigurationObject passed to the constructor of DefaultXmlDependenciesEntryPersistence.
-o <file_name>	Optional. The report output file name. Can override the parameter “report_file_name” mentioned in the section 3.2.2.
-f <format_name>	Optional. The name of the report format to be used. Currently the following values are supported(case sensitive): “xml” – represents XmlDependencyReportGenerator, “csv” – represents CsvDependencyReportGenerator and “html” – represents HtmlDependencyReportGenerator. This switch specifies or overrides the parameter “generator_class” from the section 3.2.2. This switch must not be specified together with “-gclass” switch. It duplicates functionality of “-gclass”, but is much more easy-to-use. Default format is “xml”.
-gclass <class_name>	Optional. The full class name of the DependencyReportGenerator implementation to be used. This switch specifies or overrides the parameter “generator_class” from the section 3.2.2. This switch must not be specified together with “-f” switch. XmlDependencyReportGenerator is a default generator implementation.
-dtype <types_list>	Optional. The list of dependency types those must be included into the generated report. The semicolon-separated list of case-insensitive DependencyType values. This switch can override the “dependency_types” parameter mentioned in the section 3.2.1. In this case the configuration of BaseDependencyReportGenerator is provided with “generator_config” parameter from the section 3.2.2. Default is “internal;external”.
-dcat <categories_list>	Optional. The list of dependency categories those must be included into the generated report. The semicolon-separated list of case-insensitive DependencyCategory values. This switch can override the “dependency_categories” parameter mentioned in the section 3.2.1. In this case the configuration of BaseDependencyReportGenerator is provided with “generator_config” parameter from the section 3.2.2. Default is “compile;test”.
-id <component_ids>	Optional. The semicolon-separated list of component IDs (see section 1.3.6 for format; e.g. “java-logging_wrapper-2.0.0;object_factory-2.1.0,base_exception-1.0”). If this switch is provided, then report is generated for specified components only. If this switch is not specified, report is generated for all components that have a dependency entry provided.
-indirect	Optional. Indicates that indirect dependencies should be included into the generated report. Report an error if this switch is present together with “-noindirect”. This switch specifies or overrides the parameter “indirect” from the section 3.2.2 (when this switch is provided, parameter is set to “true”).

-noindirect	Optional. Indicates that indirect dependencies should not be included into the generated report. Report an error if this switch is present together with “-indirect”. This switch specifies or overrides the parameter “indirect” from the section 3.2.2 (when this switch is provided, parameter is set to “false”).
-help -? -h	When one of the specified switches is provided, the application prints out the usage string to the standard output and terminates immediately.

3.3 Dependencies Configuration

None

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute ‘ant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

Please see the demo.

4.3 Demo

4.3.1 API usage

```
        //Create configuration object for generator
        ConfigurationObject config = new DefaultConfigurationObject("config");
        //Include only internal components in the report
        config.setPropertyValue("dependency_types", "internal");
        //Include only compile dependencies in the report
        config.setPropertyValue("dependency_categories", "compile");
        //Don’t include information about dependency type and category in the
report
        config.setPropertyValue("include_dependency_type", "false");
        config.setPropertyValue("include_dependency_category", "false");
        //Include information about dependency path in the report
        config.setPropertyValue("include_dependency_path", "true");

        //Get dependency entries persistence (see docs of Component Dependency
Extractor)
        DependenciesEntryPersistence persistence = new
DefaultXmlDependenciesEntryPersistence("/dependencies.xml");

        //Get dependency entries from persistence
        List < DependenciesEntry > entries = persistence.load();

        //Create XML report generator for entries list
        DependencyReportGenerator xmlGenerator = new
XmlDependencyReportGenerator(entries, config);

        //Create CSV report generator for dependencies persistence
        DependencyReportGenerator csvGenerator = new
CsvDependencyReportGenerator(persistence, config);

        //Create HTML report generator for entries list
        DependencyReportGenerator htmlGenerator = new
HtmlDependencyReportGenerator(entries, config);

        //Get XML report for all available components as a string, include
indirect dependencies
        String report = xmlGenerator.generateAll(true);
        System.out.println(report);

        //Create a list of component identifiers those must be included in the
report
        List < String > componentIds = Arrays.asList(
            new String[]{"java-logging_wrapper-2.0.0", "dot_net-
command_line_executor-1.0"});

        //Generate CSV report for the specified components, don't include
indirect dependencies
        //Save the generated report to report.txt
        csvGenerator.generate(componentIds, "report.txt", false);
        System.out.println(this.readFile("report.txt"));

        //Create output stream for HTML report
        OutputStream reportOutputStream = new ByteArrayOutputStream();

        //Generate and save HTML report for specified component to the given
output stream
        //Include indirect dependencies in the generated report
        htmlGenerator.generate(componentIds, reportOutputStream, true);
              System.out.println(reportOutputStream.toString());
```

4.3.2 *Usage of command line utility*

This command line can be used to print out the usage string:

```
java DependencyReportGeneratorUtility –help
```

```
DependencyReportGeneratorUtility.main(new String[]{"-help"});
```

If all required configuration for the utility is stored in file config.xml, then the application can be executed without additional arguments:

```
java DependencyReportGeneratorUtility
```

```
DependencyReportGeneratorUtility.main(new String[]{});
```

To use the custom configuration file the user can use “-c” switch:

```
java DependencyReportGeneratorUtility -c custom_config.xml
```

```
DependencyReportGeneratorUtility.main(new String[]{"-c", "custom_config.xml"});
```

The user can specify the input and output file names:

```
java DependencyReportGeneratorUtility -i dependencies.xml –o report.xml
```

```
DependencyReportGeneratorUtility.main(new String[]{"-i", "dependencies.xml", "-o",
"report.html"});
```

The user can specify the custom persistence options:

```
java DependencyReportGeneratorUtility –pclass myPackage.CustomDependenciesEntryPersistence –i
custom.dat
```

```
DependencyReportGeneratorUtility.main(
            new String[]{"-pclass", CustomDependenciesEntryPersistence.class.getName(), "-i",
"custom.dat"
            });
```

Generator implementation class can be specified as command line argument:

```
java DependencyReportGeneratorUtility –gclass myPackage.CustomDependencyReportGenerator
```

```
DependencyReportGeneratorUtility.main(
            new String[]{"-gclass", CustomDependencyReportGenerator.class.getName()});
```

For easiness "-f" switch can be used to specify one of XML, CSV and HTML output report formats:

```
java DependencyReportGeneratorUtility –f html –o report.html
```

```
DependencyReportGeneratorUtility.main(new String[]{"-f", "html", "-o", "report.html"});
```

To specify that only external component dependencies must be included in the report the user can use "-dtype" switch:

```
java DependencyReportGeneratorUtility –dtype external
```

```
DependencyReportGeneratorUtility.main(new String[]{"-dtype", "external"});
```

To specify that only test dependencies must be included in the report the user can use "-dcat" switch:

```
java DependencyReportGeneratorUtility –dcat test
```

```
DependencyReportGeneratorUtility.main(new String[]{"-dcat", "test"});
```

To include in the report direct dependencies only this argument can be used:

```
java DependencyReportGeneratorUtility –noindirect
```

```
DependencyReportGeneratorUtility.main(new String[]{"-noindirect"});
```

To generate report for Java Logging Wrapper 2.0.0 and Java Object Factory 2.1.0 only the user can use "-id" switch in the following way:

```
java DependencyReportGeneratorUtility –id java-logging_wrapper-2.0.0;java-object_factory-
2.1.0
```

```
DependencyReportGeneratorUtility.main(new String[]{"-id",
            " java-logging_wrapper-2.0.0;java-object_factory-2.1.0"});
```

Usually the command line will look like the following one:

```
java DependencyReportGeneratorUtility -c custom_config.xml –f xml –i dependencies.xml –o
report.xml –id java-logging_wrapper-2.0.0
```

```
DependencyReportGeneratorUtility.main(new String[]{
            "-c", "custom_config.xml",
            "-f", "xml",
            "-i", "/dependencies.xml",
            "-o", "report.xml",
            "-id", "java-logging_wrapper-2.0.0"});
```

4.3.3 *Sample configuration file for standalone application*

```
<?xml version="1.0"?>
<CMConfig>
  <Config
    name="com.topcoder.util.dependency.report.utility.DependencyReportGeneratorUtility">
    <Property name="persistence_config">
      <Property name="file_name">
        <Value>dependencies.xml</Value>
      </Property>
    </Property>
    <Property name="generator_class">
      <Value>com.topcoder.util.dependency.report.impl.HtmlReportGenerator</Value>
    </Property>
    <Property name="generator_config">
      <Property name="dependency_types">
        <Value>internal</Value>
      </Property>
      <Property name="dependency_categories">
        <Value>compile</Value>
      </Property>
      <Property name="include_dependency_category">
        <Value>false</Value>
      </Property>
      <Property name="include_dependency_path">
        <Value>true</Value>
      </Property>
    </Property>
    <Property name="stdout">
      <Value>false</Value>
    </Property>
    <Property name="report_file_name">
      <Value>report.html</Value>
    </Property>
    <Property name="indirect">
      <Value>false</Value>
    </Property>
  </Config>
</CMConfig>
```

4.3.4 *Sample XML report*

This report is generated for Java Logging Wrapper 2.0.0 component with direct dependencies only. Dependency type and category are included; dependency path is not included (default values).
All dependency types and categories are included.
Please see build-files from the TopCoder catalog for input information for this report.

```
<?xml version="1.0"?>
<components>
  <component name="logging_wrapper" version="2.0.0" language="java" >
    <dependency type="internal" category="compile" name="base_exception" version="2.0.0" />
    <dependency type="internal" category="compile" name="typesafe_enum" version="1.1.0" />
    <dependency type="internal" category="compile" name="object_formatter" version="1.0.0" />
    <dependency type="external" category="compile" name="log4j" version="1.2.14" />
    <dependency type="external" category="compile" name="junit" version="3.8.2" />
    <dependency type="internal" category="test" name="base_exception" version="2.0.0" />
    <dependency type="internal" category="test" name="typesafe_enum" version="1.1.0" />
    <dependency type="internal" category="test" name="object_formatter" version="1.0.0" />
    <dependency type="external" category="test" name="log4j" version="1.2.14" />
    <dependency type="external" category="test" name="junit" version="3.8.2" />
  </component>
</components>
```

4.3.5 *Sample CSV report*

This report is generated for Java Logging Wrapper 2.0.0 component with direct dependencies only. Dependency type and category are included; dependency path is not included (default values).
All dependency types are included. Only compile dependencies are included.
Please see build-files from the TopCoder catalog for input information for this report.

```
java-logging_wrapper-2.0.0,[int][compile]base_exception-2.0.0,[int][compile]typesafe_enum-
1.1.0,[int][compile]object_formatter-1.0.0,[ext][compile]log4j-1.2.14,[ext][compile]junit-
3.8.2
```

4.3.6 *Sample HTML report*

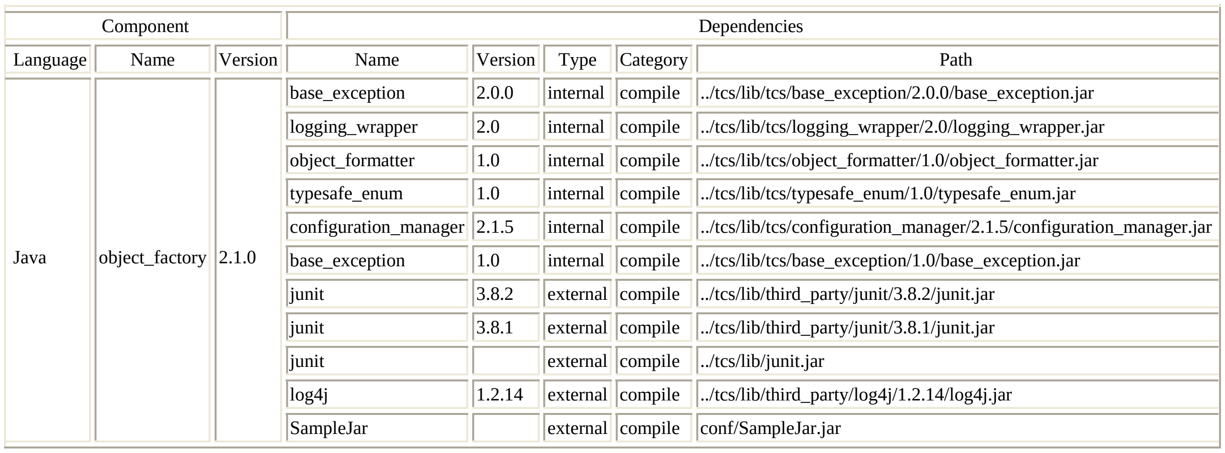
This report is generated for Java Object Factory 2.1.0 component with indirect dependencies. Dependency type, category and path are included. All dependency types and categories are included. Indentation is used for clarity only (report doesn't include line breaks and whitespaces between tags).
Please see build-files from the TopCoder catalog for input information for this report.

```
<html>
<head>
    <title>Component Dependency Report</title>
</head>
<body>
    <table border="1">
```

```
<tr align="center">
    <td colspan="3">Component</td>
    <td colspan="5">Dependencies</td>
</tr>
<tr align="center">
    <td>Language</td>
    <td>Name</td>
    <td>Version</td>
    <td>Name</td>
    <td>Version</td>
    <td>Type</td>
    <td>Category</td>
    <td>Path</td>
</tr>
<tr>
    <td rowspan="12">Java</td>
    <td rowspan="12">object_factory</td>
    <td rowspan="12">2.1.0</td>
    <td>base_exception</td>
    <td>2.0.0</td>
    <td>internal</td>
    <td>compile</td>
    <td>../tcs/lib/tcs/base_exception/2.0.0/base_exception.jar</td>
</tr>
<tr>
    <td>logging_wrapper</td>
    <td>2.0</td>
    <td>internal</td>
    <td>compile</td>
    <td>../tcs/lib/tcs/logging_wrapper/2.0/logging_wrapper.jar</td>
</tr>
<tr>
    <td>object_formatter</td>
    <td>1.0</td>
    <td>internal</td>
    <td>compile</td>
    <td>../tcs/lib/tcs/object_formatter/1.0/object_formatter.jar</td>
</tr>
<tr>
    <td>typesafe_enum</td>
    <td>1.0</td>
    <td>internal</td>
    <td>compile</td>
    <td>../tcs/lib/tcs/typesafe_enum/1.0/typesafe_enum.jar</td>
</tr>
<tr>
    <td>configuration_manager</td>
    <td>2.1.5</td>
    <td>internal</td>
    <td>compile</td>
    <td>../tcs/lib/tcs/configuration_manager/2.1.5/configuration_manager.jar</td>
</tr>
<tr>
    <td>base_exception</td>
    <td>1.0</td>
    <td>internal</td>
    <td>compile</td>
    <td>../tcs/lib/tcs/base_exception/1.0/base_exception.jar</td>
</tr>
<tr>
    <td>junit</td>
    <td>3.8.2</td>
    <td>external</td>
    <td>compile</td>
    <td>../tcs/lib/third_party/junit/3.8.2/junit.jar</td>
</tr>
<tr>
    <td>junit</td>
    <td>3.8.1</td>
    <td>external</td>
    <td>compile</td>
```

```
            <td>../tcs/lib/third_party/junit/3.8.1/junit.jar</td>
        </tr>
        <tr>
            <td>junit</td>
            <td></td>
            <td>external</td>
            <td>compile</td>
            <td>../tcs/lib/junit.jar</td>
        </tr>
        <tr>
            <td>log4j</td>
            <td>1.2.14</td>
            <td>external</td>
            <td>compile</td>
            <td>../tcs/lib/third_party/log4j/1.2.14/log4j.jar</td>
        </tr>
        <tr>
            <td>SampleJar</td>
            <td></td>
            <td>external</td>
            <td>compile</td>
            <td>conf/SampleJar.jar</td>
        </tr>
    </table>
</body>
</html>
```

This HTML report in a web browser looks like the following table:



Component			Dependencies				
Language	Name	Version	Name	Version	Type	Category	Path
Java	object_factory	2.1.0	base_exception	2.0.0	internal	compile	../tcs/lib/tcs/base_exception/2.0.0/base_exception.jar
			logging_wrapper	2.0	internal	compile	../tcs/lib/tcs/logging_wrapper/2.0/logging_wrapper.jar
			object_formatter	1.0	internal	compile	../tcs/lib/tcs/object_formatter/1.0/object_formatter.jar
			typesafe_enum	1.0	internal	compile	../tcs/lib/tcs/typesafe_enum/1.0/typesafe_enum.jar
			configuration_manager	2.1.5	internal	compile	../tcs/lib/tcs/configuration_manager/2.1.5/configuration_manager.jar
			base_exception	1.0	internal	compile	../tcs/lib/tcs/base_exception/1.0/base_exception.jar
			junit	3.8.2	external	compile	../tcs/lib/third_party/junit/3.8.2/junit.jar
			junit	3.8.1	external	compile	../tcs/lib/third_party/junit/3.8.1/junit.jar
			junit		external	compile	../tcs/lib/junit.jar
			log4j	1.2.14	external	compile	../tcs/lib/third_party/log4j/1.2.14/log4j.jar
			SampleJar		external	compile	conf/SampleJar.jar

4.3.7 *Indirect dependency sample*

If component A depends on component B and component B depends on component C, then indirect dependency report for component A must include information about both B and C components. This can be checked with the following code:

```
        //Construct the dependency entries
        Component componentA = new Component("A", "1.0", ComponentLanguage.JAVA);
        ComponentDependency componentB = new ComponentDependency("B", "1.0",
ComponentLanguage.JAVA,
            DependencyCategory.COMPILE, DependencyType.INTERNAL, "PathB");
        ComponentDependency componentC = new ComponentDependency("C", "1.0",
ComponentLanguage.JAVA,
            DependencyCategory.COMPILE, DependencyType.INTERNAL, "PathC");

        List < ComponentDependency > dependencyA = new
ArrayList<ComponentDependency>();
```

```
        dependencyA.add(componentB);
        DependenciesEntry entryA = new DependenciesEntry(componentA,
dependencyA);
        List < ComponentDependency > dependencyB = new
ArrayList<ComponentDependency>();
        dependencyB.add(componentC);
        DependenciesEntry entryB = new DependenciesEntry(componentB,
dependencyB);
        List < DependenciesEntry > entries = new ArrayList<DependenciesEntry>();
        entries.add(entryA);
        entries.add(entryB);

        //Create configuration object for generator
        ConfigurationObject config = new DefaultConfigurationObject("config");

        //Don't include information about dependency type, category
        config.setPropertyValue("include_dependency_type", "false");
        config.setPropertyValue("include_dependency_category", "false");

        //Create CSV report generator
        DependencyReportGenerator csvGenerator = new
CsvDependencyReportGenerator(entries, config);

        //Generate the report for the component A with indirect dependencies
        List < String > componentIds = Arrays.asList(new String[]{"java-A-1.0"});
        String report = csvGenerator.generate(componentIds, true);

        //Print the report to the standard output
              System.out.println(report);
```

The code above should produce the following output:

```
java-A-1.0,B-1.0,C-1.0
```

5. Future Enhancements

Support more report format.