File Statistics 2.0c Requirements Specification

1. Scope

1.1 Overview

The File Statistics component collects statistical information for one or more files. Usually these statistics are associated with the file content (number of lines, number of occurrences of a string, etc.) as opposed to the file itself (file size, last modified time, etc.) although both types of statistics could potentially be calculated by the component. The code that calculates a statistic is pluggable so that many different types of statistics can be collected. Furthermore, a mechanism exists that allows each statistic plug-in to specify what types of files it can or cannot handle. For example, a plug-in that counts lines of code would be applicable to *.java files but not *.txt files. After all files have been examined, a report is generated showing the results on a per-file, per-directory and aggregate basis. The report generation code is also pluggable and a default implementation is supplied. Finally, the component can optionally be run as a standalone application from the command line.

1.2 Changes in 2.0c

In the initial specification, .zip files would have typically been treated as binary files, meaning no useful statistics could be gathered from the contents of the archive. Version 2.0c removes this restriction and allows the component to "drill-down" into archive files and treat the zipped entries as normal files. The code that extracts entries from an archive file should be made pluggable so that different file formats can be handled. If one of these archive files contains another archive, it too can optionally be expanded, within a configurable (via Configuration Manager or command-line) nesting limit. An attribute should be added to the default XML report to indicate whether or not a file was extracted from an archive file. A "made-up" file path should be created when this expansion occurs, using a ">" character in place of a slash "/". Example: `/dir/archive.zip>dir/file.txt`. Unzipped files are extracted to a temporary, configurable directory, processed, and then deleted.

1.3 Logic Requirements

1.3.1 Determine Files to be Examined

- The component accepts one or more file path strings that may contain wildcards. The component is able to determine what files match these inputs. The paths can be absolute, relative to the current working directory, or relative to a specified directory.
- The component can optionally use recursion to traverse through subdirectories.
- Archive files (.zip, .jar, .rar, etc) can optionally be expanded to include their contents in the results. These files are given a virtual name of the form `zip-path>zip-entry`.

1.3.2 Statistics Gathering

- For each file that is to be examined, determine its type either by its file extension or by analyzing its contents.
- For each statistic plug-in that is applicable to this type of file, execute the plug-in against the file to calculate its statistics.
- Two default statistic plug-in implementations are provided:
    - A plain-text line counter that simply counts new lines (\n) in the document.
    - A naïve C, C++, and Java style line counter that estimates number of statements by counting opening braces and semicolons in the document, except when they appear in a string or a comment.

1.3.3 Reporting

- After all files that match the filters have been processed, a report is generated showing the results. This is done through a pluggable interface that allows different types of reports to be generated.
- The sections presented in the generated report are configurable. The following types of sections are available:
    - File listing, showing the results for each individual file.
    - Directory listing, showing the results for each directory.
    - File Filter listing, showing the results for each file filter input.
    - Overall results, showing the results for all scanned files.
- One default report generator implementation is provided:
    - An XML report that is saved to a configurable file location.
    - An attribute for each file element of the XML report should specify whether or not the file was extracted from an archive file.

1.4 Required Algorithms

No required algorithms.

1.5 Example of the Software Usage

The component can be used to determine the number of lines in a single file, a small project, or an entire code base. The component can also be used to count only lines of test code, even when they may be mixed in with the source files (by using file filters).

1.6 Future Component Direction

Future versions will likely include more plug-ins to gather more interesting data, such as number of classes, number of methods, number of string occurrences, number of variable declarations, etc. Future work may also include new report generators that create reports in CSV, HTML, or some other format.

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

No requirements.

2.1.2 External Interfaces

No external interfaces.

2.1.3 Environment Requirements

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

2.1.4 Package Structure

com.topcoder.file.statistics

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

- Sections included in the generated report.
- Support for passing custom parameters to individual statistic plug-ins.
- All configuration settings are able to be set or overridden from the command-line, when run as a standalone application. Therefore, the precedence is as follows, from highest to lowest:
    - Command-line value
    - Configuration manager value
    - Hard-coded value
- Option to expand archive files and examine their contents. Option to set the nesting level for this expansion in the case that an archive contains another archive.

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

No required frameworks.

3.2.2 TopCoder Software Component Dependencies:

- Configuration Manager: 2.1.3
- Magic Numbers: 1.0
- Command Line Utility: 1.0
- File Class 1.0
- Compression Utility 2.0

3.2.3 Third Party Component, Library, or Product Dependencies:

No dependencies.

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 should be detailed below.

3.4 Required Documentation

3.4.1 Design Documentation

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

- Test Cases

3.4.2 Help / User Documentation

- Design documents must clearly define intended component usage in the ‘Documentation’ tab of Poseidon.
- Javadocs must provide sufficient information regarding component design and usage.