Release Notes

UCC v.2015.12

1 Introduction

This document provides the release notes for the UCC v.2015.12. Unified Code Count (UCC) is a code metrics tool that allows the user to count physical and logical software lines of code, compare and collect logical differentials between two versions of source code of a software product, and obtain Cyclomatic Complexity results. With the counting capabilities, users can generate the physical, logical SLOC counts, and other sizing information such as comment and keyword counts of the target program. The differencing capabilities allow users to count the number of added/new, deleted, modified, and unmodified logical SLOC of the current version in comparison with the previous version. The Cyclomatic Complexity results are based on McCabe's research on this metric.

This release supports various languages including Ada, ASP/ASP.NET, Assembly, Bash, C/C++, C Shell, COBOL, ColdFusion, ColdFusion Script, CSS, C#, DOS Batch, Fortran, HTML, Java, JavaScript, JSP, Makefiles, MATLAB, NeXtMidas, Pascal, Perl, PHP, Python, Ruby, Scala, SQL, VB, VBScript, Verilog, VHDL, XML, and X-Midas. It also supports physical counting of data files.

2 Compatibility Notes

UCC v.2015.12 is released in C++ source code that allows users to compile and run on various platforms. This release has been tested on Windows using MS Visual Studio, Cygwin, and MinGW, and on Unix/Linux using the g++ compiler.

The UCC v.2015.12 does not support PL/1 and Jovial, although these may be included in future releases. For the need of counting of code in these languages, users may consider using the CodeCount Tools Release 2007.07, which does not provide the differencing capability but uses the counting rules compatible to those of UCC v.2015.12.

3 Requirements

Minimum Software Requirements:

- Compiler: a compatible C++ compiler that can load common C++ libraries including IO and STL, such as MS Visual Studio, MinGW, and g++.
- Qt: Optional. Required to use the GUI front-end.
- Boost C++ Library: Optional. Required for multithreading capability.
- Operating systems: any platforms that can compile and run a C++ application. The software has been tested on Windows XP/7, Unix, Linux, Solaris, and Mac OS X.

Minimum Hardware Requirements:

- RAM: 512 MB. Recommended: 1024 MB.
- HDD: 100 MB available. Recommended: 200 MB available.

4 Changes and Upgrades

This section describes main changes and upgrades to the tool since the release v.2014.08. Changes were in the areas of less use of RAM, performance improvements due to threads and optimizations, better ease of use due to more informative processing progress and better wording of Error/Warning/Information messages, and some metrics regarding UCC operation.

1) Bug Fixes:

   a. Improved error checking and messages for incorrect inputs and switch usage
   b. Improved progress bar
   c. Improved help for **`-extfile`**
   d. Parsing of pointer syntax corrected for C, C++, and Pascal
   e. Long command lines cause delay in performance
   f. Error checking for threshold parameters
   g. Fixed display of function names in Cyclomatic Complexity results
   h. Fixed problems with custom fileList
   i. Resolved compilation warning messages
   j. Fixed GUI failing if fileList exists
   k. Sometimes duplicate files would be given when file extensions were different

2) New Features:

   a. Multithreading with Qt or Boost C++ Library using **`—threads`** switch
   b. Much reduced use of RAM for metrics and differencing
   c. Cyclomatic Complexity Ring 4 for C++ (Identify identical decision branches) using **`—cc4enable`** switch
   d. Estimation of possible lack of RAM needed for successful processing using the **`—ramlimit`** switch
   e. Suppress warning messages from being shown using the **`—nowarnings`** switch
   f. Suppress uncounted file messages from the display and output files using the **`—nouncounted`** switch
   g. Stack Dump will give more information when C++ exceptions occur on the display and in the error log

h. Support for smart quotes
i. Make EOL compatible with DOS and Unix
j. C and C++ parsers added default .inc and .inl file extensions to support parsing projects

3) New Languages:

a. Assembly (NASM, MASM, GAS, MIPS, PowerPC)
b. COBOL
c. IDL
d. P/L SQL
e. Scala

4) New and Modified Reports:

a. UCC_Performance_<date stamp>.txt captures number of files processed, the time needed to run various steps and the UCC command line arguments used.
b. Added extra information if a C++ exception occurs in the error_log_<date stamp>.txt as output from the Stack Dump
c. Cyclomatic Complexity Ring 4 results added in the outfile_cyclomatic_cplx.csv file and more summary information is included

5 Performance Improvement

Release v2015.12 was changed to use less RAM and process a set of files as fast or faster than earlier v2014.08 of UCC and definitely faster when using 2 or more threads. The test system for the comparisons shown in this section is an AMD Athlon II Dual-Core M300 CPU clocked at 2.00 GHz laptop running Windows 7.1 64-bit with 3 GB of RAM.

RAM Usage Comparison

2014.08 RAM Usage

Test: Differencing between Boost C++ Library 1.48.0 and 1.58.0 source files

Result: Did not finish. Stopped when RAM usage reached maximum at 2.68GB. After UCC stopped, RAM use was 0.74GB for a total of 1.94 GB used.

2015.12 RAM Usage (no threads)

Test: Differencing between C++ Library 1.48.0 and 1.58.0 source files

Result: Finished successfully. Highest RAM usage was 2.52 GB. After UCC finished, RAM use was 0.785 GB for a total of 1.735 GB used. Savings is at minimum about 200 MB of RAM, or 89% compared to 2014.08.

Conclusion

RAM usage decreased 11% or more (exact percent not known as 2014.08 did not finish).

Run Time Comparison

2014.08 Run Time

Test: Linux 3.11.6 arch directory and included subdirectories of source files

Result: 163 seconds total time

2015.12 Run Time

Test: Linux 3.11.6 arch directory and included subdirectories of source files

0 threads

Result: 117 seconds total time

2 threads

Result: 76 seconds total time

Conclusion

UCC v2015.12 reduced the run time by 27% with no threads, and 53% with 2 threads.

6 Known Issues and Limitations

#	Issue
1	For JavaScript code, the tool does not count the statement that is not terminated by a semicolon.
2	The tool only detects and handles C# and VB as code-behind languages for the ASP.NET.
3	Users have reported that when large numbers of files or files with large SLOC counts are run, UCC would take several hours to process, or would hang. To improve the performance, users may choose to use the **`–nodup`** flag, which disables duplicate file separation; duplicates are counted and reported along with original files. In the situation where UCC hangs, the problem is that the host computer has run out of memory. A workaround is to break the input file list into several lists and process in multiple runs. Additional work is being done in this area, and more improvements may be available in a future release. If you suspect your process is hanging due to memory limitations, it would be appreciated it if you would report the number of files, total file size, and the host computer's memory size to UnifiedCodeCount@gmail.com.
4	The UCC is designed to process well-formed, compilable code, and does not check to see if the provided files are compilable. Files that contain software that is not compilable, or is in non-standard format, may not process correctly.
5	The Fortran counter uses the FORTRAN90 and above format for the continuation character being an & at the end of the line. FORTRAN77 and lower versions used a non-zero character in column 6 as the continuation character. There are plans to develop a separate counter for FORTRAN77 and lower in the future.
6	Due to the recent addition of cyclomatic complexity metrics, UCC is able to handle slightly smaller loads than the 2013.04 version. For counting purposes, errors can be avoided by using the **`–nocomplex`** flag. The UCC development team is working on optimizing the performance to handle larger input.