UCC Release Notes

UCC v.2017.01

1 Introduction

This document provides the release notes for the UCC v.2017.01. 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.2017.01 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, Macintosh OS X and on Unix/Linux using the g++ compiler.

The UCC v.2017.01 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.2017.01.

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 5.7.0 and Qt Creator 4.0.2: Optional. Required to use the GUI front-end and allows faster performance using threads.
- Boost C++ Library: Optional. Maybe used if not building with Qt. Gives faster performance when using threads.

- Operating systems: any platforms that can compile and run a C++ application. The software has been tested on Unix, Linux, Solaris, Mac OS X and Windows XP, 7, 8 and 10.

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.2015.12. 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. Fixed file extension unknown error when adding .swift to C_CPP as a new file extension.
   b. Fixed crashing of 'help' command.
   c. Resolved segmentation fault error when running UCC on MATLAB source code.
   d. Not counting the document type definition as a SLOC has been fixed.
   e. Not counting invocation of '<!ELEMENT' S Name S contentspec S? '>' as a SLOC has been fixed.
   f. Not counting multiple ELEMENT and ATTLIST tags have been fixed.
   g. Resolved SLOC counting error in SQL.
   h. Fixed documentation errors in SQL counting standards.

2) Feature Enhancements:

   a. Cyclomatic complexity support for Midas language.
   b. An option to specify non-default comment delimiters (for line and block commens) to the language counter through the extfile feature.
   c. Allowing a file to specify multiple file filters using the **–filespecs** command.

3) New Languages

a. Objective C
Since Objective C hares the .h file with C++, .m with Matlab, and .mm with NeXtMidas. Users need to use –extfile <extfilename> feature to specify the extensions with Objective C when trying to count Objective C files. An example of how the extfile needs to be formatted is below:

```
C_CPP=.cpp,.c,.cc,.cxx,.inl,.hh,.hpp,.hxx,.inc
MATLAB=
NeXtMidas=
OBJC=.h,.m,.mm
```

4) New and Modified Reports:

a. Customization of header in the report by using **–header** command and removal of default header using **–noheader** command.
b. Removal of unmodified files from visual differencing output.

5 Performance Improvement

Release v2017.01 was changed to use less RAM and process a set of files as fast or faster than earlier v2015.12 of UCC and definitely faster when using 2 or more threads. The test system for the comparisons shown in this section is an Intel(R) Core(TM) i5-5200U CPU clocked at 2.20 GHz laptop running Windows 10 64-bit with 12 GB of RAM.

RAM Usage Comparison

2015.12 RAM Usage (no threads)

Test: Differencing between Linux-4.7.10 and Linux-4.8.10 source files.

Result: Finished successfully. Highest RAM usage was 4.5 GB.

2017.01 RAM Usage (no threads)

Test: Differencing between Linux-4.7.10 and Linux-4.8.10 source files.

Result: Finished successfully. Highest RAM usage was 3.6 GB.

Conclusion

RAM usage decreased by 20%.

Run Time Comparison

2015.12 Run Time

Test: Differencing between Linux-4.7.10 and Linux-4.8.10 source files.

0 threads

Result: 1548 seconds or 25 minutes 48 seconds total time

2 threads

Result: 1529 seconds or 25 minutes 29 seconds total time

5 threads

Result: 1528 seconds or 25 minutes 28 seconds total time

2017.01 Run Time

Test: Differencing between Linux-4.7.10 and Linux-4.8.10 source files.

0 threads

Result: 1581 seconds or 26 minutes 21 seconds total time

2 threads

Result: 1532 seconds or 25 minutes 32 seconds total time

5 threads

Result: 843 seconds or 14 minutes 3 seconds total time

Conclusion

UCC v2017.01 increased the run time by 2% with no threads, 0.1% with 2 threads, and reduced the run time by 44% with 5 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.