# WHAT'S NEW IN C++LINT
## Last update: 20-Oct-2014

This document details the new and improved features of C++lint **2.0--3.5**, including those made possible with new information available from version **9.00L** of the PC-lint/Flexelint core engine.

This document is organized by functionality rather than version history, which makes it hard for existing users to see new changes at a glance. So as of version 3.1, we added Section 0, next.

## 0.0 Update History

Version 3.5:

- GUI: Dramatic improvement in startup time
- GUI: Clicking on message number in Analysis Report brings up a detailed discussion of the analysis message; see Section 1.7.
- IDE: **New** Rowley Crossworks
- IDE: **New** support for Visual Studio through 2014
- Core: Powerful support for MISRA 2012 and updated support for MISRA C++ 2008
- Core: 109 new error messages for a total of 1162
- Core: Support for 116 compilers and versions of compilers
- Core: Numerous feature enhancements. With version 3.5, we are adding a companion document, readnewcore.txt, located in your 'doc' subdirectory. Newest features are at the ***bottom.*** Version 9.00L is the most significant release of the core in some time, so we encourage you to read its section at the bottom of this readme file.

Version 3.1:

- Many key executables have now been compiled for x64 – a significant performance improvement – including a 64-bit PC-lint core, **a Cleanscape exclusive.** The installer handles the selection of 32- or 64-bit executables for your machine; on Linux there is a script in the bin subdirectory to select 32/64.

## 1.0 Support for IDEs

With the 3.5 release, we now support

- Visual Studio from 6 through new 2014
- **New** Rowley Crossworks

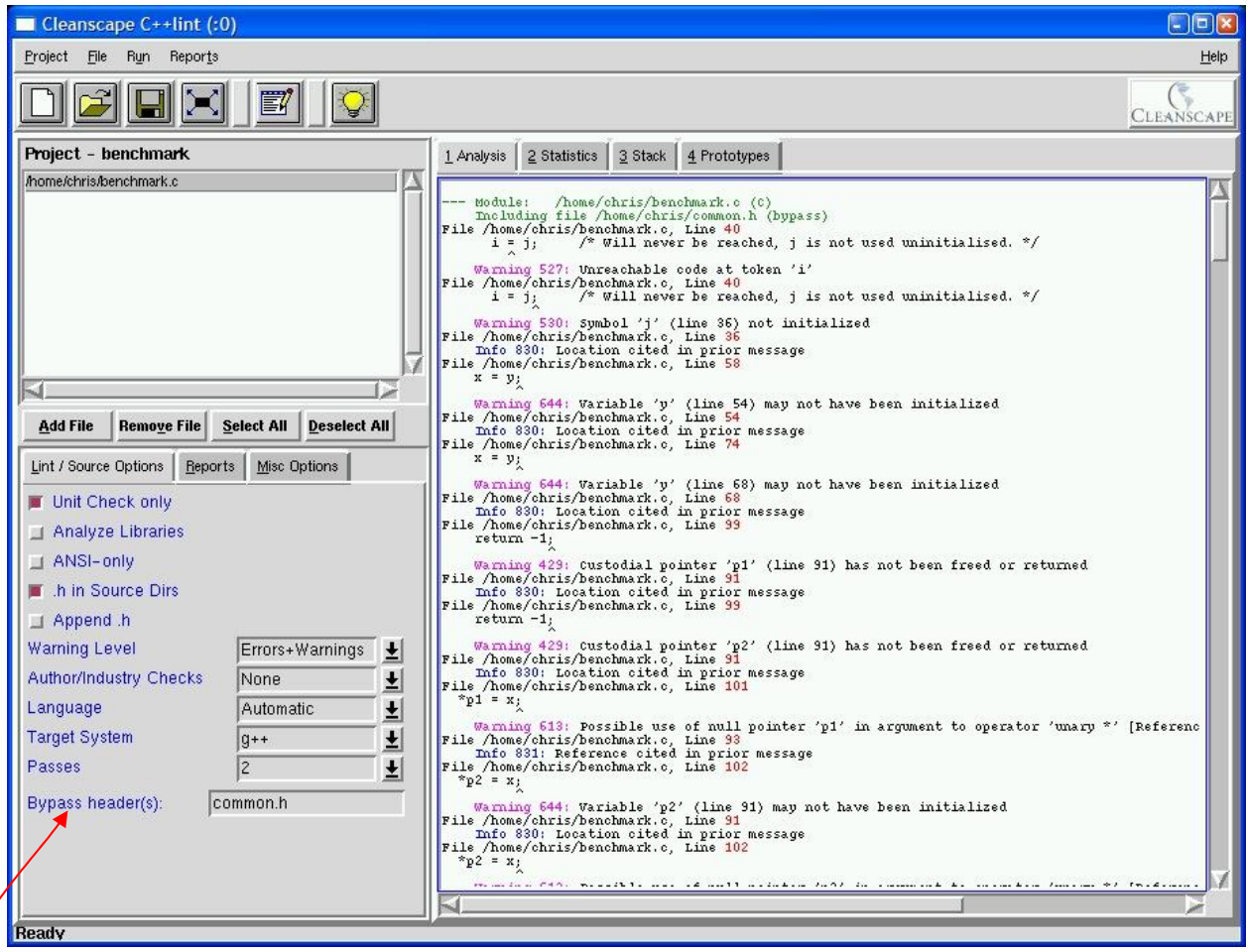
With version 3.1 we integrate with four IDEs (many more if you count versions!)

- Visual Studio from 6 through new 2012
- **New** IAR Embedded Workbench
- **New** Microchip MPLAB
- **New** Renesas High-Performance Embedded Workshop (HPEW)

By “integrate”, we mean

- We read the project files created by the IDE to obtain the source file list (omitting files manually excluded by the user), include directories, defines/undefines, and any compiler settings which may affect our analysis.
- From this data we build a <project>.lnt file to make the analysis exactly track the compilation.
- The analysis can be started from the IDE.
- Results are presented in the output window of the IDE.
- Results are formatted so that double-clicking the analysis message jumps to the “offending” line of source code in the IDE’s internal editor

## 1.1 Support for bypass headers (GUI) or precompiled headers (IDE)




Header processing, especially for large application frameworks, can represent a large – even a majority – of the time spent analyzing the application. Consequently, by managing the "long poles in the header tent", the time savings can be quite substantial.

Bypass headers can greatly improve the speed of your C++lint analysis by physically scanning the specified include (header) files only on their first encounter; the analyzer recreates the effect of these files and their subheaders during subsequent encounters.

Precompiled headers, like `stdafx.h` in Visual Studio compilations, can be specified in the project's `std.lnt` file using the `-pch()` command option; when processed, the C++lint analyzer builds a `.lph` (lint precomiled header) binary file which is referenced thereafter.

## 1.2 Inheritance Report – a Cleanscape Exclusive new in version 3.0

This report shows the relationships between classes in the selected C++ source files. It is a hyperlinked graph of the class hierarchy, indicating parents and unconnected nodes.


## 1.3 Include Tree Report – a Cleanscape Exclusive new in version 3.0

This report demonstrates the nesting structure of the Include files used by the selected C/C++ source file(s). Filenames in angle brackets indicate system include files; these can optionally be excluded from the report. Also optional is the "Redundant Includes" check: a scan of all directories in the Include path to determine if multiple occurrences of the same Include file name are present. This could indicate problems if different users define their Include paths differently.

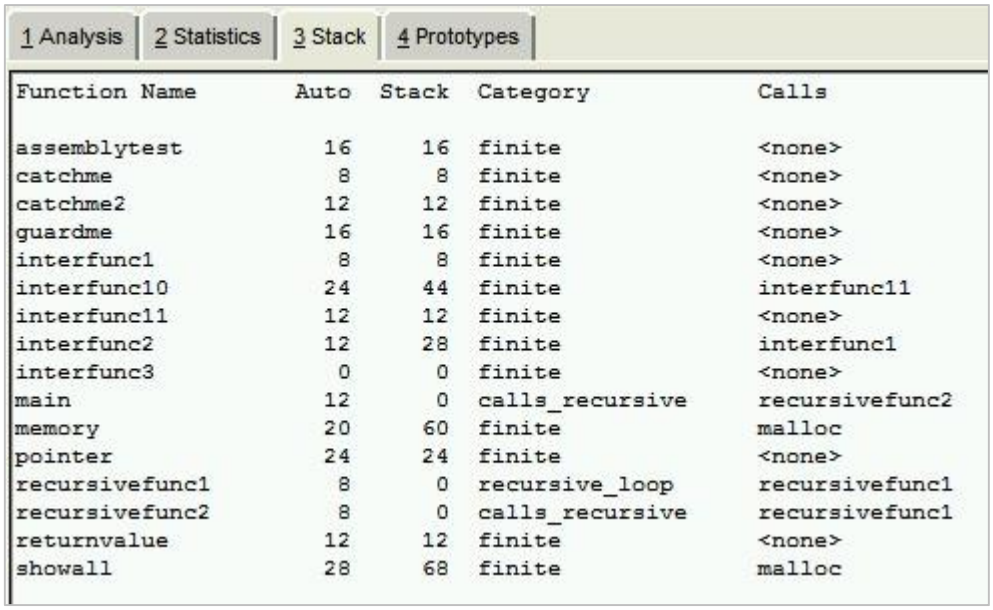
Filenames in red are hyperlinked to open it using the specified external editor.

1 Analysis | 2 Statistics | 3 Inheritance | 4 Include Tree | 5 Stack | 6 Prototypes

```
BEGIN INCLUDE TREE

Source File example3.cpp
 SubClassTwo.cpp
  SubClassOne.cpp
   InheritanceRoot.cpp
  TwosOtherParent.cpp
 SubClassSix.cpp
  SubClassSeven.cpp
 ThirdParent.cpp
 Other.cpp

Source File fpatch.c
 <stdio.h>
  <crtdefs.h>
   <sal.h>
    <sourceannotations.h>
   <vadefs.h>
  <swprintf.inl>
   <vadefs.h>
 <stdlib.h>
  <crtdefs.h>
   <sal.h>
   <vadefs.h>
  <limits.h>
   <crtdefs.h>
    <sal.h>
    <vadefs.h>
 <ctype.h>
  <crtdefs.h>
   <sal.h>
   <vadefs.h>
 <string.h>
  <crtdefs.h>
   <sal.h>
   <vadefs.h>

REDUNDANT INCLUDE SEARCH RESULTS

<limits.h>==>c:\progra~1\micros~2.0\vc\include\limits.h
          ==>c:\progra~1\micros~1.0\vc\include\limits.h

InheritanceRoot.cpp==>c:\progra~1\cleans~1\lpp\examples\incsub~1\inheri~1.cpp
                   ==>c:\progra~1\cleans~1\lpp\examples\inheri~1.cpp

SubClassOne.cpp==>c:\progra~1\cleans~1\lpp\examples\incsub~1\subcla~1.cpp
               ==>c:\progra~1\cleans~1\lpp\examples\subcla~3.cpp

<ctype.h>==>c:\progra~1\micros~2.0\vc\include\ctype.h
         ==>c:\progra~1\micros~1.0\vc\include\ctype.h

<string.h>==>c:\progra~1\micros~2.0\vc\include\string.h
          ==>c:\progra~1\micros~1.0\vc\include\string.h

<crtdefs.h>==>c:\progra~1\micros~2.0\vc\include\crtdefs.h
           ==>c:\progra~1\micros~1.0\vc\include\crtdefs.h

<stdlib.h>==>c:\progra~1\micros~2.0\vc\include\stdlib.h
          ==>c:\progra~1\micros~1.0\vc\include\stdlib.h

<stdio.h>==>c:\progra~1\micros~2.0\vc\include\stdio.h
         ==>c:\progra~1\micros~1.0\vc\include\stdio.h

<sal.h>==>c:\progra~1\micros~2.0\vc\include\sal.h
       ==>c:\progra~1\micros~1.0\vc\include\sal.h

<swprintf.inl>==>c:\progra~1\micros~2.0\vc\include\swprintf.inl
              ==>c:\progra~1\micros~1.0\vc\include\swprintf.inl

<vadefs.h>==>c:\progra~1\micros~2.0\vc\include\vadefs.h
          ==>c:\progra~1\micros~1.0\vc\include\vadefs.h
```

## 1.4 Stack Usage Report — a Cleanscape Exclusive

This report is a tabular list of auto storage and stack requirements for each function in your project; it is very useful for embedded systems development where the amount of stack (and overall memory) can be mission critical.



1 Analysis | 2 Statistics | 3 Stack | 4 Prototypes

| Function Name | Auto | Stack | Category | Calls |
|---|---|---|---|---|
| assemblytest | 16 | 16 | finite | <none> |
| catchme | 8 | 8 | finite | <none> |
| catchme2 | 12 | 12 | finite | <none> |
| guardme | 16 | 16 | finite | <none> |
| interfunc1 | 8 | 8 | finite | <none> |
| interfunc10 | 24 | 44 | finite | interfunc11 |
| interfunc11 | 12 | 12 | finite | <none> |
| interfunc2 | 12 | 28 | finite | interfunc1 |
| interfunc3 | 0 | 0 | finite | <none> |
| main | 12 | 0 | calls_recursive | recursivefunc2 |
| memory | 20 | 60 | finite | malloc |
| pointer | 24 | 24 | finite | <none> |
| recursivefunc1 | 8 | 0 | recursive_loop | recursivefunc1 |
| recursivefunc2 | 8 | 0 | calls_recursive | recursivefunc1 |
| returnvalue | 12 | 12 | finite | <none> |
| showall | 28 | 68 | finite | malloc |

## 1.5 Statistics/Summary Report — a Cleanscape Exclusive

This report is the melding information from several sources to produce a report with multiple categories, as shown in the sample on the next page:

1 Analysis | 2 Statistics | 3 Stack | 4 Prototypes

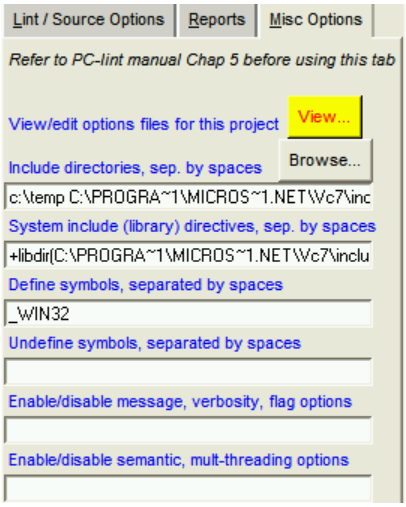
```
Analysis options used to generate this report were:
+fdi +macros +linebuf +linebuf -width(0) -zero(400)
-ic:\progra~1\cleanscape\cpplint\main\lbin\lf co.lnt
-iC:\PROGRA~1\MICROS~1.NET\Vc7\include -u
-summary(c:\progra~1\cleanscape\cpplint\main\reports\cpplint.stt)
+stack(&file=c:\progra~1\cleanscape\cpplint\main\reports\cpplint.stk)
-od100(c:\progra~1\cleanscape\cpplint\main\reports\cpplint.pro) -wlib(1) -w2
-passes(2) +byph(common.h) -vf -d"_WIN32" -"format_stack=%f\t%a\t%n\t%t\t%c" +e974
+program_info(output_prefix=c:\progra~1\cleanscape\cpplint\main\reports\cpplint_)
C:\temp\benchmark_files.lnt

>>> Statistics:

Number of source files:        1
Number of files processed:     4

Total number of symbols:     135
   auto                      110
   static                      3
   function                   18
   member_function             0
   instance_member             0
   enumerator                  0
   type                        0
   label                       0
   class_template              0
   function_template           0
   namespace                   1
   template_parameter          0
   using_declaration           0
   <indeterminate>             3

Individual message summary
--------------------------
Count   Type    Number  Text

4       Error   10      Expecting ___
1       Warning 413     Likely use of null pointer '___' in ___argument to operator '_
16      Warning 429     Custodial pointer '___' (___) has not been freed or returned
4       Warning 438     Last value assigned to variable '___' (___) not used
2       Warning 449     Pointer variable '___' previously deallocated___
15      Warning 522     Highest operation, ___ '___', lacks side-effects
2       Warning 527     Unreachable code at token '___'
2       Warning 529     Symbol '___' (___) not subsequently referenced
4       Warning 530     Symbol '___' (___) not initialized
1       Warning 533     function '___' should___ return a value (see ___)
11      Warning 534     Ignoring return value of function '___' (compare with ___)
2       Warning 550     Symbol '___' (___) not accessed
2       Warning 564     variable '___' depends on order of evaluation
4       Warning 593     Custodial pointer '___' (___) possibly not freed or returned
22      Warning 613     Possible use of null pointer '___' in ___argument to operator
5       Warning 644     Variable '___' (___) may not have been initialized
1       Note    974     Worst case function for stack usage: ___ See +stack for a full

Errors Warnings Notes Info
------ -------- ----- ----
     4       93     1    0

Total messages: 98
```

## 1.6 Functional upgrades: IDE

- In version 3.1 you can now lint an entire Visual Studio solution.
- Also in version 3.1, better integration with GUI for preparing inheritance, include tree, stack usage, and statistics reports, and added to the VS Tools menu.
- The std.lnt file created by the C++lint integration into Visual Studio includes a `-pch(stdafx.h)` line (commented by default).
- The "Open `.lnt`" tool in Visual Studio now opens *`project.lnt`* in addition to `std.lnt`.
- Icon added for 1-button creation of IDE `std.lnt` file from the Cleanscape GUI:

- Visual studio indirect files (`.lnt`) reordered for improved operation.
- Streamlined handshakes between GUI, VS integration mode, and core PC-lint engine.
- A number of small bug fixes, including VS integration when installing and not "owner" of the machine and VS tool removal.

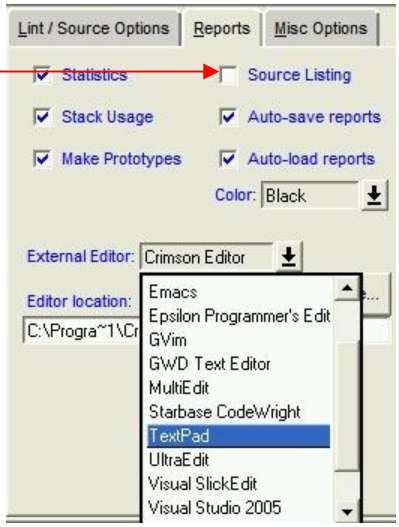
## 1.7 Functional upgrades: GUI



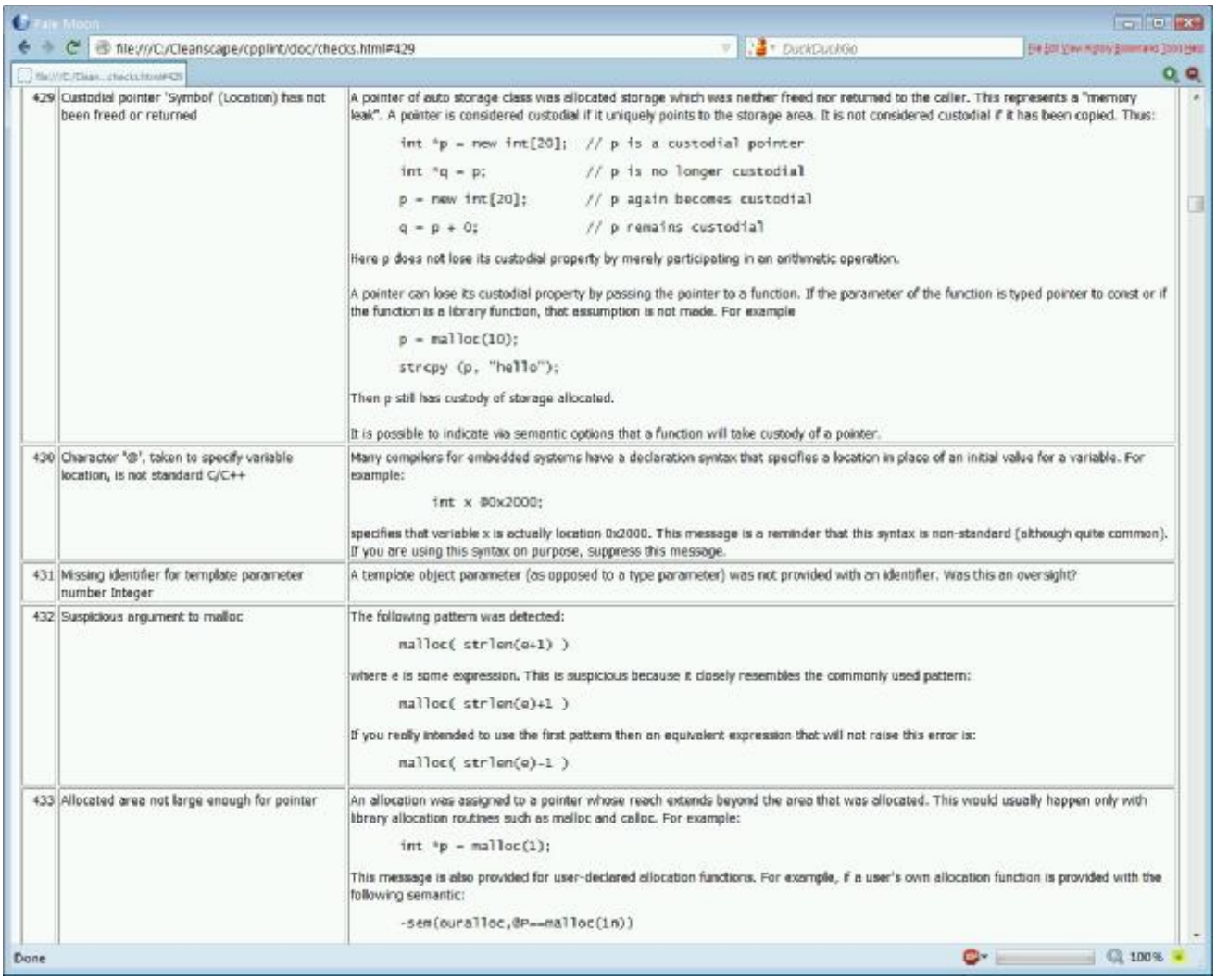

- Greatly improved startup performance in version 3.5.
- Projects within the Cleanscape GUI may now have their own *`project.lnt`* file, which may be used to
  - refine or override some of the GUI controls,
  - add PC-lint commands such as `-strong()` not controlled from the GUI, and
  - supercede default options for the Target System options specified in associated the `co-`*`xxx`*`.lnt` file

  The *`project.lnt`* file is created by clicking the View... button on the Misc Options tab, which also opens the `co-`*`xxx`*`.lnt` file in the user's specified External Editor.
- Some hard-coded controls now made into checkboxes (e.g., number of analysis passes, whether to search source directories for double-quoted header files – see sample screen in 1.0 intro).
- System ("library") include directories added by C++lint are now also flagged as such with `+libh()` PC-lint directive.
- Icon added for 1-button creation of IDE/command line `std.lnt` file from the Cleanscape GUI:

- May now list source code in the analysis report; great for documentation.
- Numerous visibility enhancements and small bug fixes



- In version 3.5, clicking on an error message *number* (as opposed to the *line number* causing the message) opens a detailed description of the error message in your system browser. So for example, if you clicked on **Warning 429** in the GUI's Analysis Report, your system browser would open up to Message 429 like this:




So to summarize mouse clicking in the Reports window:

- If you click on any message number (colors vary), your system browser will open the error message file so you can view a detailed description of the error.
- If you click on a source line number in bright red, that source line will open in your favorite code editor for immediate review/ correction!

## 1.8 Functional upgrades: Command line

- Icon added for 1-button creation of command line/IDE `std.lnt` file from the Cleanscape GUI:

- That save now captures files you select during the GUI session, speeding up the process of setting up a command line analysis project. A sample `std.lnt` file is shown below.

```
//  *** This file created by Cleanscape C++lint GUI on 21-Nov-08 ***
// Refer to the PC-Lint Users Manual for details on control settings.

+fdi
+macros
+linebuf
+linebuf
-width(0)
-zero(400)
-ic:\progra~1\cleanscape\cpplint\main\lbin\lf

// Uncomment the next line(s) for command line (non-IDE) operation:
//co-msc71.lnt
//--iC:\PROGRA~1\MICROS~1.NET\Vc7\include
//+libdir(C:\PROGRA~1\MICROS~1.NET\Vc7\include)

-summary(c:\progra~1\cleanscape\cpplint\main\reports\cpplint.stt)
+stack(&file=c:\progra~1\cleanscape\cpplint\main\reports\cpplint.stk)
-wlib(1)
-w2
-passes(2)
+byph(common.h)
-vf
-d_WIN32

// Add any additional PC-lint controls next:


// For command line operation, uncomment the sourcefile names below.  Add
// additional files as necessary; enclose in double quotes if name has spaces.
//C:\PROGRA~1\cleanscape\cpplint\examples\example1.c
//C:\PROGRA~1\cleanscape\cpplint\examples\example2.c
```

- Operational improvements in the `cpplint.bat` control program.
- Extensive updates to the documentation and the Online Help system.  Since Online Help provides important operational data on each GUI function and it associated PC-lint command, Online Help has been formatted as a `.pdf` document for reference/review.

## 1.9 Fully automated gcc/g++ compiler setup

Since gcc and g++ are custom-built at each site, it is not possible to know beforehand all the defines, macros, include directories, etc., specified during the build. Cleanscape provides an *exclusive* tool (external program `setupgcc`) to automate extraction of this information and place it in helper (`.h` and `.lnt`) files later consumed by C++lint.

The result?  A comprehensive analysis of code intended for gcc/g++ with no false positives!

A `setupgcc` session on Linux is shown next; user input has been colored green.

```
Redhat:/usr/local/cleanscape/bin# setupgcc

This program extracts unique settings (internal defines/macros and default
include directories) for gcc/g++ on this system for use by C++lint.  An
interactive command line session confirms the correct compiler, gathers
the info, and stores it.  If your gnu compiler is rebuilt, then force
this program to retrieve new data by running  setupgcc –FORCE

Both gcc and g++ found; which do you use (gcc/g++)? G++

To get the right predefined macros, enter the compiler options you specify
during builds, or <ENTER> to accept the default in parentheses below.
(-O)

Built /home/lint/build/unixbcsi/prod/cpplintgui.dir/main/lbin/lf/mygpp.lnt
and   /home/lint/build/unixbcsi/prod/cpplintgui.dir/main/lbin/lf/mygpp.h.

Command line: Add 'co-gpp.lnt' as a parameter when running 'cpplint'.
         GUI: Select 'g++' in the Target System dropdown.

To test your configuration, try the command line or GUI on file
'testcfg.cpp' located in the examples subdirectory.

Press <ENTER> to exit…
```

## 1.10 Semi-automated process for adding new compilers to C++lint

If gcc/g++ integration can be automated, why not *any* compiler (namely, those that are not yet integrated into the analyzer)? It turns out that we can do a great deal of automation in this regard, although ultimately the solution depends on experience with the compiler and the lint analysis of source code intended for that compiler.

Cleanscape provides an *exclusive* tool (`scavenge`) that extracts the built-in macros for your compiler, then prepares a template `co-`*`name`*`.lnt` file with descriptive section headers for you to fill in other important data about your compiler, and finally provides a mechanism to input your resulting compiler options file into the Cleanscape GUI. A sample `scavenge` session is shown next; user input has been colored green.

```
C:\cleanscape\cpplint\bin> scavenge

This program extracts built-in macros for compilers not yet supported in
C++lint.  An interactive command line session obtains compiler basics,
gathers the info, and creates a co-xxx.lnt template which you later edit.

In this interactive session, you will need the following data:
- The full pathname of your compiler
- The full pathname to or the compiler options for an internal preprocessor
- The compiler options specified during builds (or within makefile)
- The preprocessor options to output to a named file
- The directories containing the compiler's header files (the ones specified
  using angle brackets in your #include statements)

If you don't have this info, hit <CTRL-C> now to exit and rerun later, or
press <ENTER> to continue…

Great!  Let's get started.

We need a short label name for your compiler which will be used to name your
co-xxx.lnt compiler.  For instance, if your compiler is the Greenhills C
compiler for the i960 processor, you might choose GHC960, whereupon the
resulting compiler options file would be named 'co-GHC960.lnt'.

Enter a label name for your compiler: BCC55

Is your project C or C++? (c/cpp): cpp

Enter the full pathname of your compiler: c:\borland\bcc55\bin\bcc32.exe
```

```
Now we need some preprocessor information.
Is it a unique program or just compiler switches? (P/S): p
Enter the full pathname of the preprocessor: c:\borland\bcc55\bin\cpp32.exe

Now enter the preprocessor options to
- Enable preprocessor output, if necessary.
- Disable any source line info, if necessary.
- Output to a specified file; use the word FILE place the filename.
  If the preprocessor only outputs to 'stdout', type the word STDOUT.

EX. 1: Microsoft.    /EP STDOUT
EX. 2: Borland.      -P- -oFILE
EX. 3: Generic Unix. -P STDOUT

Enter the preprocessor options: -P- -oFILE

Next, we need the directories containing the compiler's include files (the
'system header files'). If already present in the INCLUDE environment variable,
type just the single word INCLUDE; type DONE when done.

NOTE: The contents of INCLUDE env. Var. are Excluded by default, so you must
      specify the word INCLUDE if you want its contents.

Enter dir path, INCLUDE, or DONE: c:\borland\bcc55\include
c:\borland\bcc55\include added OK.

Enter dir path, INCLUDE, or DONE: done

Enter the compiler options you specify during builds (e.g., as specified in
your makefile), or press <ENTER> to accept the default in parentheses below.
(-O)

Obtaining built-in macros; this may take a couple minutes… done.
Processing macro list… done.
Cleaning up… done.
Building co-BCC55.lnt… done.
Adding bcc32.exe to GUI Target System dropdown… done.
To remove it later, delete the BCC55 line from file
   C:\PROGRA~1\cleanscape\cpplint\bin\mycompiler.lst

Now opening in Notepad; add data for each section as described in the comments.

Now you are ready to test your configuration:
- Add co-BCC55.lnt on the command line  -OR-
  Select BCC55 in the GUI Target System dropdown.
- Specify '-w1'  (command line)  -OR-  select 'Errors_only' (GUI).
- Specify '+fdh' (command line)  -OR-  select 'Append .h'   (GUI)
  if your compiler uses the same .h header file for both C and C++.
- Use input file 'testcfg.cpp' located in the examples subdirectory.

Press <ENTER> to exit…
```

The `co-BCC55.lnt` file resulting from the above session is shown below:

# WHAT'S NEW IN PC-LINT COMMAND LINE (CORE) PROGRAM

As of version 3.5, we only hit the highlights of new features and instead include the readme file from Gimpel as to the changes in the core analyzer. This file is named readnewcore.txt and is located in your 'doc' subdirectory. It is organized by release, with the latest release notes at the *bottom.*

Version 9.00L is the most significant release of the core in some time. You are encouraged to read its section at the bottom of readnewcore.txt and is located in your 'doc' subdirectory.

## 2.1 Major New Features

- Pre-compiled headers, as users of C and C++ systems are well aware, can dramatically reduce the time spent in processing multiple modules. See Section 7.1 "Pre-compiled Headers".

- We now incorporate variables of static storage duration in our value tracking. These include not only variables that are nominally static, as local to a function and local to a module, but also external variables. See –static_depth(n) and flag –fsv.

- We examine multi-threaded programs for correct mutex locking and report on variables shared by multiple threads that are used outside of critical sections. See Chapter 12. Multithread Support

- Stack usage: We can report (Note `974`) on the overall stack requirements of any program whose function calls are non-recursive and deterministic (i.e. calls not made through function pointers). This is very useful for embedded systems development where the amount of stack required can be mission critical. A complete detailed report of stack usage for each function is available as well (See `+stack()` in Section 5.7 Other Options).

- We now support through the strong type mechanism the classical dimensional analysis that engineers and physicists have traditionally employed in verifying equations. A 'type' can now be a ratio or product of other types and the compound types are checked for consistency across assignment boundaries. See Section 9.4 Multiplication and Division of Strong Types.

- The user may deprecate particular symbols in any of the following categories: `function`, `keyword`, `macro` and `variable`. See `–deprecate`.

- Message Enhancement and Control
  See readnewcore.txt, located in your 'doc' subdirectory.

- Enhanced MISRA Checking
  Significantly enhanced support for MISRA C3 (MISRA C 2012). A number of new messages to support MISRA C3 have been added and the au-misra3.lnt author file has been updated accordingly. We now provide some level of support for all but 3 of the decidable rules and 4 of the undecidable rules, with full support being provided for the vast majority of the rules.

  We have broken out the 1960/1963 diagnostics (used to support MISRA C++) into individual messages to provide a greater level of control and flexibility, especially with regards to message suppression. The 1960/1963 messages will continue to be supported for backwards compatibility.

A new author file, au-misra-cpp-alt.lnt, can be used to take advantage of the new messages.
**Cleanscape note:** The GUI uses this new author file for MISRA_CPP_2008.

- A comprehensive collection of information about your program is optionally provided yielding information on files, types, symbols and macros for simple viewing or in a manner absorbable by a database or spreadsheet. This information can be used for many purposes, including naming-style conventions. See `+program_info`.
  **Cleanscape note:** The GUI supplies this information in the Statistics Report.

- Macro Scavenging: This feature turns PC-lint/FlexeLint into a seeker of built-in macros supported by a compiler and lying about within compiler header files. This is perfect for the unknown compiler with long and forgotten macros ready to trip up a third party processor such as PC-lint/FlexeLint. See
  `-scavenge`.
  **Cleanscape note:** This is the basis for the Cleanscape-exclusive `scavenge` program.

- We now provide significantly improved support for VS2012 through an updated co-msc110.lnt file, available on our patches page.

- Patch 9.00L introduces support for range-based for statements, the 'final' class specifier, standard layout types, improved handling of null pointer constants, significantly improved handling of C++11 core language features used in Visual Studio 2012 headers. As with other C++ 2011 features, these are available using the option: -A(C++2011).

- We now support binary literals and digit separators from C++14. These features are available using the option: -A(C++2014).

## 2.2 New Messages of Note
See file readnewcore.txt, located in your ‘doc’ subdirectory.

## 2.3 New Error Inhibition Options – See Section 5.2 Error Inhibition Options
See file readnewcore.txt, located in your ‘doc’ subdirectory.

## 2.4 New Verbosity Options – See Section 5.4 Verbosity Options
See file readnewcore.txt, located in your ‘doc’ subdirectory.

## 2.5 New Flag Options – See Section 5.5 Flag Options
See file readnewcore.txt, located in your ‘doc’ subdirectory.

## 2.6 New Message Presentation Options – See Section 5.6.1 Message Height Options and 5.6.3 Message Format Option
See file readnewcore.txt, located in your ‘doc’ subdirectory.

## 2.7 Additional Other Options – See Section 5.7 Other Options
See file readnewcore.txt, located in your ‘doc’ subdirectory.

## 2.8 Compiler Adaptation – See Section 11.1.1 Special Functions

See file readnewcore.txt, located in your ‘doc’ subdirectory.

## 2.9 New Messages

See file readnewcore.txt, located in your ‘doc’ subdirectory.