

Using PC-lint Plus in Safety Critical Applications

Version 2.0

Vector Informatik GmbH

Updated 2023-02-07

Overview

The static analysis tools developed by Vector Informatik GmbH are trusted by thousands of organizations across virtually every industry around the world. A significant portion of our user base utilizes our tools to develop applications for embedded and safety critical environments. We have long supported this community by implementing language extensions used by embedded compilers, supporting the various MISRA coding standards, and adding diagnostics to address issues specific to embedded developers. As part of our continued commitment to the embedded and safety critical developer communities, we have prepared this guide for using PC-lint Plus in such environments.

This guide provides:

- A list of Best Practices which, when followed, will help to maximize the correct use of PC-lint Plus by making you aware of areas of potential problems, steps that can be performed to avoid or detect anomalous operation, and suggestions for policies that support a robust linting environment.
- A discussion of environment considerations and how PC-lint Plus may be affected by the environment in which it is run.

Best Practices for employing PC-lint Plus in Safety Critical Applications

Ensure proper configuration

Develop a short test program that includes library headers and utilizes any compiler-specific features being relied on. Run PC-lint Plus on this program to ensure it produces the correct results and that no errors are being emitted from the processing of library headers.

Before employing new library headers, create an empty C or C++ file that includes those headers and run PC-lint Plus on that file. Before running PC-lint Plus on any production code that utilizes those library headers, ensure that any errors emanating from the library headers are addressed.

Validate the tool for a specific purpose

We recommend the use of a test harness to ensure that PC-lint meets your needs and expectations to build confidence in the configuration of PC-lint Plus and validate its capabilities for a specific purpose.

For example, if you are relying on PC-lint Plus to detect specific MISRA violations, you should 1) review our documentation to determine the level of support claimed, 2) review and understand any documented caveats associated with the support claim for the rule, and 3) create tests that contain known violations of the MISRA rules and verify that, using your configuration, PC-lint Plus produces expected corresponding diagnostics.

Such tests should be run regularly including when utilizing new facilities of PC-lint Plus, when your configuration changes, and when PC-lint Plus is upgraded. Maintaining such tests will help provide “increased confidence from use”, and provide evidence of proper operation as well as to quickly identify potential issues.

Understand how options are processed and the scope of their effects

PC-lint Plus processes options in the order in which they appear and allows options to be provided on the command line, in lint configuration files, via the LINT environment variable, and within lint comments in source code. While this provides significant flexibility, it is important to understand the scope of options, the impact of employing options at different points, and how options interact with each other (including how options can undo previous options).

Because options are processed in order, you want to ensure that your main configuration is processed before any of the source code that should be subject to such options. This is typically done by having one or more files that contain the desired options, and a separate file that contains the names of the source files to be processed. The configuration files are then specified on the command line before the list file.

It is important to remember that many options specified before source code processing can be overridden for individual modules by lint comments encountered in the source module (options within a source module have a scope that is limited to the module in which they appear).

Understand the direct impact of options on the delivery of diagnostic messages

All options affect the behavior of PC-lint Plus in some way and may therefore impact the result of analysis. As such, the function of an option should be understood before it is employed. Some

options have a primary goal of directly impacting the issuance of diagnostics. Suppression options are the most obvious example as they exist solely to control delivery of diagnostics. There are other options which can directly impact the behavior of diagnostics including:

- Options which control which files, or file regions, are treated as library including `-libh`, `+libh`, `-libm`, `+libm`, `-libdir`, `+libdir`, `-library`, and `+libclass` and the `flb` flag option. Since library code is often subject to less scrutiny than non-library code, such options may directly affect diagnostics issued for files that are impacted by these options.
- The `-limit` and `++limit` options can artificially limit the number of diagnostics issued to a provided value.
- The `-misra_interpret` and `+misra_interpret` options may change the criteria under which certain messages are emitted.
- The `-skip_function` option will cause PC-lint Plus to skip the body of the specified function effectively preventing analysis of the function.
- The `-unit_check` option disables global wrap-up and the messages that are issued during the global wrap-up phase.
- The `flf` flag option controls to what extent library function definitions are analyzed.
- The `fia` flag option controls whether supplementary messages are issued.
- Certain flag options such as `fcv` and `fdx` directly impact message issuance by altering the criteria under which the message is issued.

Preventing Key MessageSuppressions

It may be desirable to prevent the suppression of certain messages. For example, there may be specific rules that project policy does not allow to be deviated from and in such cases the corresponding messages should never be suppressed.

The `++efreeze` option can be used to prevent the suppression of either an entire warning level or a message pattern. For example, `++efreeze(w1)` prevents the suppression of all errors, `++efreeze(123)` prevents suppression of message 123, and `++efreeze(123?)` prevents the suppression of messages 1230-1239. The `++efreeze` option is not reversible within the option context it appears (although it is possible to switch to another option context later using the `-env_pop` and `-env_restore` options, see the Reference Manual for details) and suppression messages that appear after the option will have no effect on the frozen messages. Note that options, such as `-esym`, that appear before the `++efreeze` option are not affected and may still exert influence on message suppression decisions following a `++efreeze` option.

In addition to the permanent `++efreeze` option, there is also a `+efreeze` version that is identical except that its effect can be suspended with a corresponding `-efreeze` option.

Do not suppress message 686 - Option is suspicious

Message 686 is a vehicle for conveying warnings about options which are likely to produce unexpected behavior or have unanticipated side effects which should not be suppressed or ignored. The approach taken to resolve 686 messages should include an understanding of why the message is being generated, and the option causing the message should then be adjusted appropriately. The option `++efreeze(686)` placed at the beginning of your configuration can be used to ensure the message is never suppressed.

Enable message 686 for library files

The default warning level for libraries is 1 which causes PC-lint Plus to only issue errors for library files. Being a warning and not an error, message 686 is not reported for library files by default. Library files typically do not include lint options but if they do their effects will be realized so suspicious options executed in library files should be reported. You can enable message 686 for library files using the option `+elib(686)`.

Do not use the `-wlib(0)` or `-w0` options

The `-wlib(0)` option suppresses **all** messages from *library* code and the `-w0` option suppresses **all** messages from *all* code. These options should not be used in safety critical applications as they will prevent even the most serious of errors from being emitted. The `-wlib(1)` and `-w1` options are much safer alternatives.

The desire to employ the `-wlib(0)` and `-w0` options typically stem from one of four reasons:

1. To eliminate errors emanating from library headers.

Use of `-wlib(0)` as an attempt to suppress messages stemming from library headers is never an advisable option. Such an attempt masks only the symptoms of the problem instead of addressing the root of the issue. As such, source code encountered later may not be properly evaluated leading to unreliable output.

Errors generated from library headers are often the result of an incomplete or incorrect configuration. An effort should be made to understand the cause of the problem and address the issue based on that understanding.

It may be desired to suppress non-error messages from library files if the library files are not in the scope of your linting policy. In such cases, the messages should be suppressed using `-elib*` options to limit the effect of the suppression to library files.

2. To create a “clean slate” from which to enable particular messages.

A “clean slate” that does not include basic error messages is not good practice. Even if error messages are later enabled, there may be a period of time when serious messages are not emitted, hiding fundamental issues.

3. When PC-lint Plus is being used for a purpose other than standard analysis:

In this scenario, the `-w0` option is used with an option such as `-stack_info` where the standard output is not going to be used. While standard output may not be the focus when running lint in such a mode, suppressing error messages can still hide serious issues that can result in incorrect or incomplete operation. The `-w1` option is a much better option.

4. When PC-lint is being used for testing or troubleshooting purposes.

It is unlikely that testing or troubleshooting will justify the use of these options but, if there is no possible adverse impact on a project by such testing, the use of these options may be allowable simply by virtue of the fact that such operation is out of scope of the project.

Using `-wlib(0)` or `-w0` will result in a 686 message warning such as:

```
Warning 686: Option '-wlib(0)' is suspicious because of 'the likelihood of
causing meaningless output'; receiving a syntax error in a library file
most likely means something is wrong with your Lint configuration
```

As indicated previously, message 686 should never be suppressed.

Do not use a suppression option whose message pattern consists entirely of question marks and/or stars

A suppression option whose message pattern is one or more stars will effectively suppress all messages as per the scope of the option. For example, `-elib(*)` has the same effect as `-wlib(0)` and as such carries the same caution. Similarly, `-e*` has the same effect as `-w0`. A message pattern of `*` should be avoided for scoped suppression options like `-efunc` as well.

For example, `-efunc(*,foo)` will disable all messages within the body of the `foo` function. This may be a mistake or an attempt to suppress several different messages that occur while processing `foo`. In the latter case, it would be more prudent to suppress the individual messages so that later modifications to `foo` that may introduce serious issues are not unintentionally suppressed.

A suppression pattern consisting entirely of question marks will be treated the same as a star. For example, `-e??` is equivalent to `-e*` and will suppress all options, not just those numbered 10-99. Because of this behavior, any message suppression pattern consisting solely of question marks is suspect as it implies a misunderstanding of how such an option will behave.

Consider using the `+flm` option to lock the message format

The format of the messages produced by PC-lint are controlled with the `-h`, `-width`, and various `-format` options. Output can be redirected to a specified file with the `-os` option. If the output of PC-lint is being processed by an automated system, the format of messages is important because improperly formatted messages could result in the loss of information by a system that is not able to parse the messages.

To prevent the accidental change of message format during processing (such as by a rogue `-format` option provided in a lint comment inside of source code), it is suggested that the `+flm` option be employed immediately after the format is configured. The `+flm` flag option will cause future `-h`, `-width`, `-format`, and `-os` options to be ignored until a `-flm` option is encountered.

Do not use the `+fce` option

The `+fce` option instructs PC-lint Plus to continue processing upon encountering an `#error` directive. While this option can be useful for troubleshooting purposes, it should not be employed in a production environment. Encountering an `#error` directive is typically indicative of configuration errors such as missing or incorrect macro definitions. Forcing processing to continue after such a directive hides such issues and may result in incomplete or incorrect analysis.

Do not use the `+fcs` option

The `+fcs` option instructs PC-lint Plus to continue processing upon encountering a failed static assertion. Like `+fce` described above, while this option can be useful for troubleshooting purposes, it should not be employed in a production environment. A failed static assertion is typically indicative of configuration errors such as missing or incorrect macro definitions. Forcing processing to continue at such a point hides such issues and may result in incomplete or incorrect analysis.

Use the `+flf` option if library code must be checked

Library function definitions are typically not processed which means that the corresponding analysis is not performed for these functions. This is done for several reasons:

- It is usually not desired to receive messages about the implementation of library functions as these are not normally in the scope of analysis.
- Such definitions may contain implementation-specific constructs not supported by PC-lint without additional configuration.
- There is a small time savings realized by not processing such definitions.

If library code should be subject to the same analysis as project code, use the **+flf** option to force the proper analysis to take place.

Verify that project headers are never marked as library headers

Use the verbosity option **-vf** to produce verbosity output when a header is included. This will emit:

```
Including file example.h (library)
```

for a library header, or:

```
Including file example.h (hdr)
```

otherwise. Ensure that a project header is never included as **(library)** as this will mark its symbols as library symbols and inhibit messages that are disabled for library code.

Use the **-function** and **-sem** options to enforce function semantics

PC-lint Plus contains innate knowledge of the semantics of many Standard C functions. For example, the fact that a call to **exit** or **abort** will never return, that **fgets** may return a null pointer, or that the 3rd argument to **memchr** should not be larger than the size of the buffer given by the first argument are all known to lint.

These same semantics can be copied to user-defined functions using the **-function** option. Completely new semantics can be defined using the **-sem** option. Specifying semantics for user-defined functions not only serves to document assumptions and requirements of the function but also provides additional insight to lint about the use of the function which can result in more comprehensive analysis.

See the “Semantics” chapter of the Reference Manual for additional information.

Review the Revision History of new releases to understand potential deficiencies in the software

Each release of PC-lint Plus documents detailed information about changes made to the product in the Revision History chapter of the Reference Manual. The description of these changes provide a brief overview of new features, changes in existing behavior, and bugs corrected. It is important to understand the changes made to the software and how they may impact a project before upgrading, but it is also important to review the bugs that have been fixed to gain an understanding of deficiencies present that will remain if the lint software is *not* upgraded. It is recommended that all Revision History items are reviewed for each release and potential impacts of described changes be assessed before using the new version.

Always use **+e900** to ensure successful completion of processing

The **+e900** option will enable message 900. Message 900 is emitted at the very end of the linting process to provide an indication of successful completion along with the number of messages

generated and the number of modules processed. The count of generated messages does not include the 900 message itself.

Message 900 will not be generated in the event that PC-lint Plus encounters a fatal error or otherwise terminates prematurely, so the presence of message 900 is an indication that analysis was completed in its entirety. In addition, the number of generated messages specified in the 900 message can be compared to the number of messages that an external system received with any discrepancy representing a possible message formatting or parsing issue. The number of modules reported by message 900 can likewise be compared to the number of modules that PC-lint Plus was expected to analyze.

Managing Defects in PC-lint Plus

Like all complex software products, PC-lint Plus contains defects. Known issues and limitations are published in our Reference Manual but new issues may be encountered by users. When a potential defect is discovered, it should be reported to Vector Informatik GmbH for verification and correction of the issue. It is important to understand the scope of the impact that the defect represents and to document the defect along with any work-around provided.

Document the use of Environment Variables

While most of the configuration for PC-lint is achieved through plain-text configuration files, there are two optional environment variables that can alter the behavior of PC-lint. These variables are documented in the Reference Manual and are **LINT** and **INCLUDE**. If the **LINT** environment variable is set when PC-lint is started, its contents are prepended to the argument list. If the **INCLUDE** environment variable is set, the list of directories specified is used to search for files as specified in section 15.2.1 of the Reference Manual.

PC-lint also provides an option, **-incvar**, which allows an alternate name for the **INCLUDE** environment variable to be specified. In this case, PC-lint will use the value of the provided environment variable instead of **INCLUDE**.

Finally, note that user-created Indirect files (configuration files) may contain references to arbitrary environment variables to specify the location of files and within certain options (such as the **-i** option option) that support the expansion of environment variables. Such expansions are relatively easy to spot as they must be surrounded by **%** signs, e.g. **%PATH%**. The use of any environment variables in a project's lint configuration should be documented.

Platform difference with the PC-lint Plus executable

PC-lint Plus is distributed as native binary files for Windows, macOS, and Linux. The functional differences between these builds are:

- The default value for the **fff** flag is **ON** for the Windows build and **OFF** for other builds.
- The initial default sub-options for the **+dump_queries** option are equivalent to **+dump_queries(unicode,colors,nocompact)** on the Linux and macOS builds and equivalent to **+dump_queries(nounicode,nocolors,nocompact)** on the Windows build.

Environmental Considerations

Dynamically Linked Libraries

The dynamically linked libraries (dlls) required by PC-lint Plus for Windows are:

- SHELL32.dll
- ole32.dll
- KERNEL32.dll
- ADVAPI32.dll

Versions of PC-lint Plus that incorporate license management software may additionally require the following dlls whose functionality is used only for license validation purposes:

- IPHLPAPI.DLL
- WS2_32.dll
- USER32.dll
- OLEAUT32.dll

Windows Registry

PC-lint Plus does not utilize the Windows registry in any way.

Installation Directory

A PC-lint Plus installation consists of the PC-lint Plus executable file and a set of zero or more configuration files. If the executable file is deleted, the program cannot run (it doesn't exist). If one or more of the configuration files is removed the effect will be based on the remaining configuration and the contents of the deleted files. If PC-lint Plus fails to find or open a referenced configuration file, a fatal error message will be presented and the program will terminate (this is the default behavior).

Parallel Execution

During execution, PC-lint Plus can be instructed to write output to file. For example, the `-oe` and `-os` options specify where standard error and standard output are written, the `-write_file` option can be used to write data to arbitrary files, the `-pch` option may cause precompiled header files to be written, and the `-summary` and `-stack` options can be directed to write output to a file. It is recommended that concurrent instances of PC-lint Plus not be run when there is any possibility of overlap between the files that one instance may write and that which another instance may read or write.

PC-lint Plus does not utilize system-wide shared resources such as shared memory and typically does not create temporary files so running multiple instances of PC-lint Plus on separate projects should be safe except for the caveats mentioned above. Note that the `-max_threads` option can be used to safely perform a multi-threaded analysis on a project by employing multiple analysis threads which use mutexes to protect against race conditions such as those described above.

Limited Processing Resources

In the event of limited CPU resources, PC-lint may take longer than normal to perform its processing but there should be no other impact. The results of the analysis performed by PC-lint should not be affected.

Limited Memory Resources

PC-lint allocates RAM at various points in its processing, the amount of RAM used can vary significantly depending on a number of factors including the sizes and contents of files being processed and the number of passes being performed. If the operating system is ever unable to fulfill the request for memory by PC-lint, the program will terminate prematurely and will not produce message 900 as described above.

Revision History

Version 2.0

- Added version number and revision date to title page.