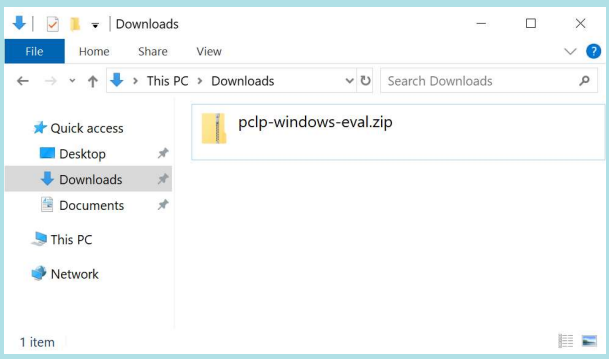
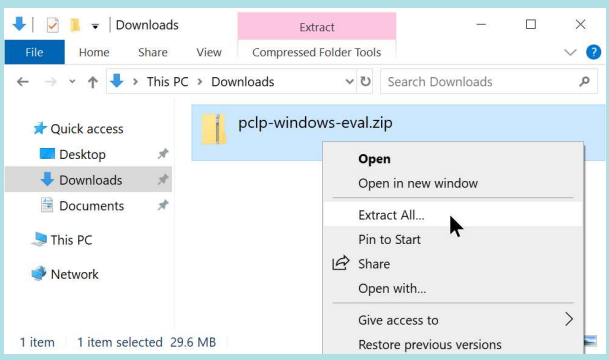
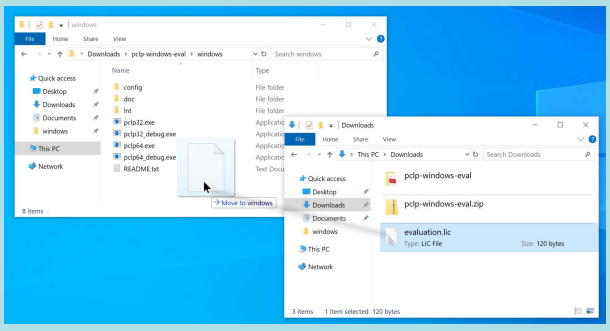
PC-lint Plus Quick Start for Visual Studio



Locate the downloaded evaluation zip file.



Extract the zip file.

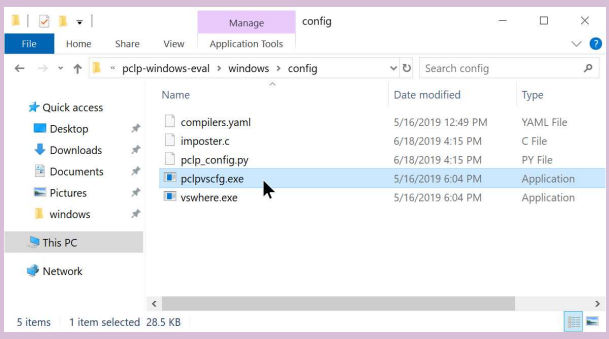


Move the license file downloaded from the email attachment to the directory containing the PC-lint Plus executables.

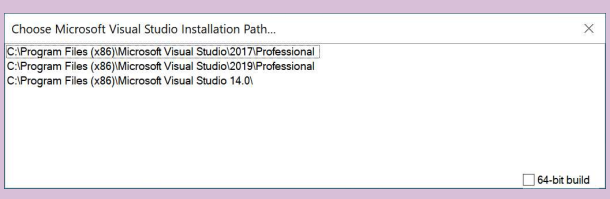
The evaluation license is now ready for use. **Follow the next steps to create a configuration for your compiler and project.**

Configure PC-lint Plus for a Visual Studio project or solution:

The configuration utility requires Python and the `regex` and `pyyaml` Python modules to be installed which can be accomplished easily by following the directions provided in the subsections "Installing python" and "Installing required modules" in section 2.3 of the Reference Manual.



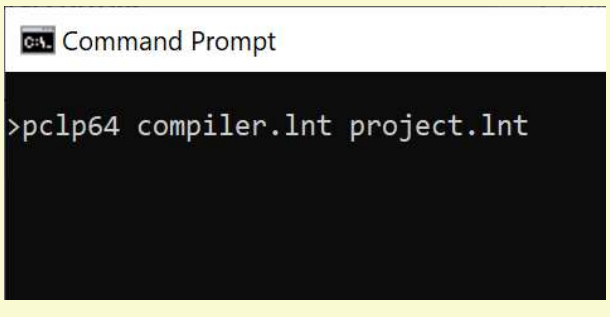
Launch `pclpvscfg.exe` from the config directory.



Select the version of Visual Studio you are using from the list. Check the "64-bit build" box if appropriate. Double click the selected Visual Studio version to continue, then follow the prompts.

From a Command Prompt or Powershell window, **run the batch file** created by `pclpvscfg.exe`. The batch file will generate the compiler and project configuration files.

Run PC-lint Plus with a compiler and project configuration:



Run PC-lint Plus from a command prompt, providing the compiler and project configuration files as arguments in that order. **To integrate PC-lint Plus into the Visual Studio IDE, proceed to Step 2.**

Getting Started using PC-lint Plus with Visual Studio

Introduction

PC-lint Plus is a powerful static analysis tool that can find bugs and potential issues in C and C++ source code, help detect violations of safety-related coding guidelines such as MISRA and AUTOSAR, and improve the quality of a project by revealing dubious constructs, undefined behavior, and other issues that make the code unnecessarily difficult to understand. Before you can start using PC-lint Plus on your project, it needs to be correctly configured. The PC-lint Plus Reference Manual that is distributed with the product provides details about all features of PC-lint Plus but this guide will help you to quickly configure PC-lint Plus and provide practial pointers for using PC-lint Plus.

Please contact us at `support@pclintplus.com` with any questions or feedback regarding your evaluation experience.

Prerequisites

Apply evaluation license file

Download the evaluation license file sent as an email attachment. Place the evaluation license file in the directory containing the PC-lint Plus executables extracted from the zip file.

Ensure `pclp_config` dependencies are installed

`pclp_config` requires Python and the `regex` and `pyyaml` Python modules to be installed which can be accomplished by following the directions provided in the subsections "Installing python" and "Installing requird modules" in section 2.3 of the Reference Manual.

Step 1 - Create a compiler configuration

While C and C++ are standardized languages, every compiler provides its own predefined macros, non-standard keywords, compiler-specific behavior, etc. that PC-lint Plus needs to know about in order to correctly analyze your source code. PC-lint Plus also needs to know the sizes of types like `int` and `float` and where your compiler looks for system headers. The included `pclp_config` tool will automatically extract the necessary information from your compiler.

Use `pclp_config` to generate a compiler configuration

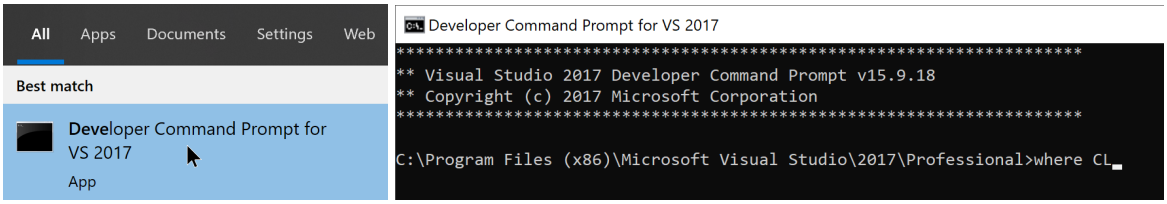
Note: Compiler and project configuration files can be generated automatically for a Visual Studio project or solution using the `pclpvscfg.exe` GUI utility distributed with PC-lint Plus. See the first page of this guide for a quick start using this utility. The following section describes how to use the command line utility `pclp_config.py` to create a Visual Studio compiler configuration. The Reference Manual also describes this process in the subsection "Creating a compiler configuration for Microsoft C/C++ compilers" of section 2.3 "Configuring with `pclp_config`".

First, you need to identify the full path of your compiler. Open a "Developer Command Prompt" (*not* a Powershell) for the version of Visual Studio you want to create a configuration for. You can find installed Developer Command Prompts by pressing the Windows key to open the Start menu or Start screen and typing `"dev"` in the search field. Select the appropriate Developer Command Prompt to launch.

From the Developer Command Prompt, run:

```
where CL
```

This command will print the path to the Visual Studio compiler, `cl.exe`. If you do not see a path and the message `INFO: Could not find files for the given pattern(s).` appears then please ensure the command was executed from a *Developer* Command Prompt, not a regular Command Prompt.



You can now use `pclp_config.py` to generate a compiler configuration. This can be performed from a regular Command Prompt in a convenient working directory to collect output files. Substitute and run the following command to generate compiler configuration files:

```
python pclp_config.py
    --compiler=vs2019
    --compiler-bin=C:\path\to\cl.exe
    --config-output-lnt-file=co-vs2019.lnt
    --config-output-header-file=co-vs2019.h
    --generate-compiler-config
```

- Replace the `vs2019` argument to the `--compiler` option with the appropriate Visual Studio version (`vs2017`, `vs2015`, etc.). Note that `vs2019` refers to the 32-bit compiler. Use `vs2019_64` (or `vs2017_64`, etc.) for 64-bit projects. The filenames in the two `--config-output` options should be updated to match the compiler.
- Replace the `C:\path\to\cl.exe` argument to `--compiler-bin` with the full compiler path produced by the `where CL` command in the previous step.
- This command is presented on multiple lines for readability but must be entered at the Command Prompt on a single line.
- Additional compiler options can be provided using the `--compiler-options` option. For example, if you build your Visual Studio project using the `CL.exe` option `/std:c++17` then this should be provided to `pclp_config.py` by adding the option `--compiler-options="/std:c++17"`.

Test the compiler configuration

The compiler configuration generated by `pclp_config` will consist of a `.lnt` file and a `.h` file with the names you provide (the name should typically correspond to the compiler, e.g. `co-gcc.lnt` and `co-gcc.h` for GCC). These files should be kept together (in the same directory). PC-lint Plus can now be used to analyze source modules like so:

```
pclp co.lnt source-files
```

where `pclp` refers to the PC-lint Plus executable (`pclp64.exe` for Windows, `pclp64_linux` for Linux, and `pclp64_macos` for mac OS), `co.lnt` should be replaced with the `.lnt` file generated above, and *`source-files`* is a list of one or more C and/or C++ source files to analyze.

If `co.lnt` or your source files are not in the directory from which PC-lint Plus is executed, you need to use the `-i` option to specify the directories that should be searched for these files, e.g. `-iC:/pclp/lnt/`.

You should confirm that PC-lint Plus is able to successfully parse a non-trivial source file with the generated compiler configuration. This can be done using:

```
pclp co.lnt -w1 +e900 source-files
```

Use a valid C or C++ source file that does not contain syntax errors.

The `-w1` option will suppress all messages except errors which indicate a configuration problem and the `+e900` option will enable message 900 which is emitted when PC-lint Plus successfully finishes processing the provided source file(s). The result should look something like this:

```
PC-lint Plus 2.0, Copyright Vector Informatik GmbH 1985-2022
```

```
--- Module:   a.c (C)
--- Module Wrap-up
--- Global Wrap-up

--- Module:   a.c (C)
note 900: execution completed producing 0 primary and 0 supplemental messages
    (0 total) after processing 1 module
```

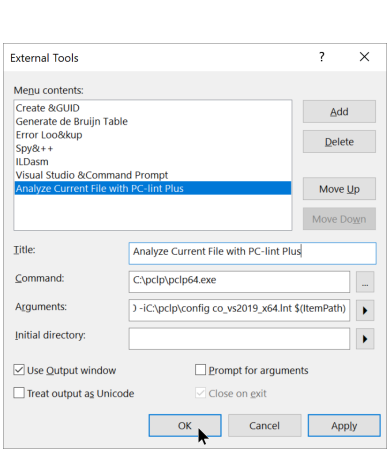
If you receive error messages here, that is indicative of an incomplete configuration file (see Troubleshooting FAQ below).

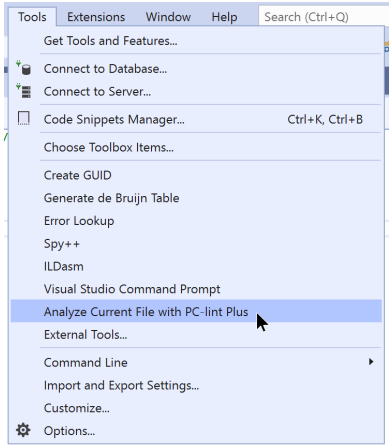
Step 2 - Integrate with your IDE (optional)

PC-lint Plus is a command line application which allows it to be integrated easily with other tools including your build process, continuous integrations, and many IDEs. If you execute your build process within an IDE, you will probably want to be able to run PC-lint Plus from within your IDE.

PC-lint Plus can be integrated into Visual Studio as an External Tool available in the Tools menu. Output from PC-lint Plus can be displayed in the Output window with support for double-click navigation to the relevant source code line.

1. Open the "Tools" menu and select "External Tools...".
2. Press the "Add" button on the right.
3. In the "Title" field, name the command "Analyze Current File with PC-lint Plus"
4. Press the "..." button to the right of the "Command" field and locate the PC-lint Plus executable you want to use. We recommend that you use `pclp64.exe` unless you have a specific reason to select a 32-bit or debug executable.
5. In the Arguments field, enter the following (including quotes and dashes) with spaces separating each line:
   - `-"format=%(%F(%l):%) error %n: (%t -- %m)"`
   - `-hF2 -width(0) -t4 +e900`
   - `-iC:\directory\of\configuration` *where* `C:\directory\of\configuration` is the directory where you saved the compiler configuration produced in the previous step or by the `pclpvscfg.exe` GUI
   - The name of the compiler configuration file you generated that exists in the directory specified in the previous item, e.g. `co-vs2019.lnt`
   - `$(ItemPath)`
6. Select the "Use Output window" checkbox.
7. Press OK





Test the new External Tool entry by selecting "Tools", "Analyze Current File with PC-lint Plus" while a C or C++ source file (`.c` or `.cpp`) is open. Headers included by the analyzed source file will be analyzed as well. There is

generally no need to directly pass a `.h` file as a module argument to PC-lint Plus.

Step 3 - Generate a project configuration

The *compiler* configuration tells PC-lint Plus how to process source code written for a particular compiler. A *project* configuration tells PC-lint Plus how to analyze a given project and typically consists of:

- The list of source files to analyze as part of the project
- Project-specific macros defined in your project's build process
- Locations for project-specific headers
- Message suppressions used in the project

A configuration file is a plain text file with a `.lnt` extension that consists of PC-lint Plus options. While these same options can be provided on the command line, it is usually easier to maintain your configuration when these options are all contained in a single file. This file can then be referenced on the command line to invoke the options contained within it.

The simplest project configuration contains just the names of the source modules to analyze. If you have project-specific macros (macros defined in your compiler invocation when building the project), these should be included in this file as well. Macros can be defined using the `-d`, `+d`, or `++d` options, e.g. `-dFOO` defines the macro `FOO` without a value, `-dBAR=1` defines the macro `BAR` and causes it to expand to `1`, and `-dfoo(x)=x` defines a function-like macro. See detailed documentation for these options in section 4.4.2 of the Reference Manual.

As you use PC-lint Plus, you may want to enable messages that are not enabled by default or suppress some messages. Messages can be *disabled* globally using `-e#` (e.g. `-e888` will suppress message 888) and *enabled* globally using `+e#` (e.g. `+e888` to enable message 888). PC-lint Plus provides options to enable or suppress messages within specified functions, files, or macro expansions and enable or suppress messages that reference specific symbols, types, or strings. Additionally, using source code comments, messages can be suppressed on particular lines and within specified expressions, statements, and braced regions. See section 4.3.1 (Error Inhibition) for descriptions of all the suppression mechanisms that PC-lint Plus provides.

Note: Using the `pclpvscfg.exe` GUI utility distributed with PC-lint Plus, a compiler and project configuration can be generated automatically for a Visual Studio project or solution. See the first page of this guide for a quick start using this utility.

1. Follow the instructions from Step 1 to open a Developer Command Prompt and locate the `cl.exe` binary.
2. From the `config` directory included with PC-lint Plus, execute `cl.exe imposter.c` to compile the `imposter` utility.
3. Locate the Visual Studio project or solution file for your project. Solution files have a `.sln` extension.
4. In the Developer Command Prompt, set the `IMPOSTER_LOG` environment variable to the full path where compiler invocations should be logged:

   ```
   set IMPOSTER_LOG=/path/to/imposter-log
   ```

   The file name is not important. If the file already exists, it should be truncated before continuing as `imposter.exe` appends entries to this file without truncating it.
5. Run `msbuild` on your project file using `imposter.exe` as the compiler by executing the following commands in the same Developer Command Prompt:

   ```
   msbuild project.sln /t:clean
   msbuild project.sln /p:CLToolExe=imposter.exe /p:CLToolPath=C:\path\to\imposter
   ```

   Note that the name without a path is provided to the `/p:CLToolExe` option and the path, without the file name, is provided to the `/p:CLToolPath option`.

Note: If your project fails to properly build using `imposter.exe` as the compiler you may need to have `imposter.exe` run the compiler during the build process. This can be accomplished by running the following command and then running the two above commands in the same Developer Command Prompt:

```
set IMPOSTER_COMPILER=/path/to/cl.exe
```

6. Run `pclp_config.py` to process the output of the imposter log and generate a project configuration:

```
python pclp_config.py
    --compiler=vs2015
    --imposter-file=/path/to/imposter-log
    --config-output-lnt-file=project.lnt
    --generate-project-config
```

Replacing `vs2015` with the appropriate compiler name, `/path/to/imposter-log` with the same value that `IMPOSTER_LOG` was set to above and `project.lnt` with the desired value.

Step 4 - Analyze your project

Once you have a project configuration containing at least the names of your source files, you can simply run PC-lint Plus as:

`pclp` *`co.lnt project.lnt`*

where *`co.lnt`* is your compiler configuration and *`project.lnt`* is your project configuration.

Again, there should be no *error* messages (those introduced with `error`; warnings and infos are expected). If you encounter error messages at this point, there is likely a configuration issue that needs to be addressed before continuing.

Next Steps

- **Further Reading**
  The Reference Manual found in the `doc/` directory of the PC-lint Plus distribution documents all aspects of PC-lint Plus including many features that are not described here. Of particular importance when first becoming aquanted with PC-lint Plus are:
  - Chapters 1 and 3 of the Reference Manual comprise fewer than 5 pages but provide useful insight into the workings of PC-lint Plus.
  - Chapter 2 describes how to configure PC-lint Plus for other compilers and IDEs not mentioned above.
  - Chapter 4 documents the many options supported by PC-lint Plus. All configuration of PC-lint Plus takes place through the use of options, whether appearing on the command line, a `.lnt` file, or a source code annotation.

- **MISRA, AUTOSAR, and CERT C guideline checking**
  PC-lint Plus can check your source code for violations of supported MISRA, AUTOSAR, and CERT C coding guidelines. These checks can be enabled by simply referencing the appropriate configuration files supplied in the `lnt/` directory of the PC-lint Plus distribution (e.g. `au-misra3.lnt` for MISRA C 2012, `au-autosar.lnt` for AUTOSAR, etc.). A few quick pointers about these files:
  - Make sure to reference the configuration file(s) *before* your source files or they won't take effect until after your modules are processed.
  - Each of these configuration files enables the language mode associated with that standard; if you are using a newer C or C++ version than the corresponding guidelines were written for, you will need to add your own `-std` option after referencing the file. For example, MISRA C++ requires the use of C++03 so the `au-misra-cpp.lnt` file contains the option `-std=c++03` which will result in unwarranted syntax

errors if you are using a newer language version. In that case you will need to provide the option `-std=c++11`, `-std=c++14`, `-std=c++17`, or `-std=c++20` after referencing the MISRA configuration file.

- The configuration files enable all supported checks for both *library* code (e.g. system headers) and *non-library* code. To disable such checking within library code, add the options `-wlib(4) -wlib(1)` immediately after referencing the corresponding configuration file(s).
- The PC-lint Plus Reference Manual contains a chapter corresponding to each guideline family: "MISRA Standards Checking", "AUTOSAR Standard Checking", and "CERT C Standard Checking". Each of these chapters provides additional useful information about how PC-lint Plus supports the corresponding standard and includes tables that detail the level of support for each guideline.

- **Introducing PC-lint Plus to an established project**
  When analyzing a project with PC-lint Plus for the first time, the volume of feedback can often be daunting. PC-lint Plus categorizes all diagnostics as *errors*, *warnings*, *infos*, and *notes*. The default warning level of 3 results in errors, warnings, and infos being enabled. If the output produced by PC-lint Plus is overwhelming, it can be helpful to start by changing the warning level to two using the option `-w2` which will limit output to errors and warnings. Each diagnostic also has its own message number which can be used to suppress the message globally using `-e#` (where # is the message number) if you are not interested in a particular diagnostic or do not feel its issuance is useful. Such an approach should make it easier to process the output of PC-lint Plus by reporting only the most egregious issues detected.

Troubleshooting FAQ

- **How can I skip processing of headers?**
  PC-lint Plus needs to correctly process all included headers in order to understand and properly analyze your source code. If the desire is to suppress violations of coding guidelines in library headers, use the options `-wlib(4) -wlib(1)` to suppress all messages except errors which indicate a configuration problem in library code as explained above. If errors are being reported from header files, see the next two items to resolve this issue.

- **PC-lint Plus cannot find my header**
  PC-lint Plus has no innate knowledge about the location of headers; this information must be provided explicitly using `-i` options. A compiler configuration generated with `pclp_config` will contain `-i` options that correspond to system include directories extracted from the compiler but any other include directories used by the analyzed project will need to be specified manually in your project configuration. If PC-lint Plus issues error 322 (unable to open include file), use the `-i` option to specify the directory containing the header. For example, if the include directive is `#include "dir3/test.h"` and the full path of this file is `C:/dir1/dir2/dir3/test.h`, the appropriate `-i` option would be `-iC:/dir1/dir2` (as this is the directory that contains `dir3/test.h`).

- **Syntax errors from headers**
  Syntax errors often represent a configuration issue and generally should not be suppressed as doing so only hides issues and compromises analysis. Errors emitted within headers are often related to missing macro definitions that cause an undesired conditional inclusion path to be followed while processing the header. A review of the code surrounding the location provided in the error message emitted by PC-lint Plus will generally reveal the macros involved which can then be defined appropriately with the `-d` option.

- **PC-lint Plus produces too many messages**
  When PC-lint Plus is initially introduced to a mature project, the volume of messages emitted with default settings can be overwhelming. See "Introducing PC-lint Plus to an established project" above for some tips on managing the results of PC-lint Plus in such situations.

- **How can I tell PC-lint Plus to write analysis output to a file?**
  The output of PC-lint Plus can be directed to a file using the `-os(`*`filename`*`)` option. Since options in PC-lint Plus are processed in the order in which they appear, make sure that this option appears early in your invocation to ensure the desired output is redirected.

- **Does PC-lint Plus provide output summaries or reports?**
  The `-summary` option will generate a summary of all messages that were issued by PC-lint Plus. PC-lint Plus does not produce any general *reports* but rather provides a flexible diagnostic format that can be used by a post-processing tool to generate custom reports. See section 4.3.3 of the Reference Manual for information on message formatting options.
- **How can I configure PC-lint Plus to produce HTML/XML output?**
  PC-lint Plus provides several format options that can be used to produce robust HTML or XML output. See the `env-html.lnt` and `env-xml.lnt` configuration files for examples of how to produce HTML and XML output, respectively. These configuration files can be used as is by referencing them in your PC-lint Plus invocation or can be modified to meet your specific needs.
- **How can I make PC-lint Plus run faster?**
  There are a variety of factors that influence the performance of PC-lint Plus including the size of the project being analyzed, the options being used, the available hardware of the host machine, and other processes running on the machine. The following factors may artificially degrade the performance of PC-lint Plus:
  - Accessing data over a network. The PC-lint Plus executable, any configuration files, and source files (including headers) being analyzed should be locally available on the machine executing PC-lint Plus. If files are accessed over a network (e.g. a network drive), this will introduce a bottleneck that may significantly impact performance.
  - Virus scanners. Inappropriately configured virus scanners may result in substantial slow downs during analysis.
  - Insufficient memory. The memory used by PC-lint Plus will vary considerably depending on the project and use case. If physical memory is being exhausted during analysis, this can degrade performance due to thrashing.
  - Running in a virtualized or emulated environment. Virtualized environments generally have access to only a subset of the host machine's resources and emulation layers may incur performance penalties, both of which may limit the performance of PC-lint Plus.

  The performance of PC-lint Plus may be improved by:
  - Using parallel analysis. By default, PC-lint Plus will use a single thread to perform analysis. If the host machine has multiple processors or cores, PC-lint Plus can analyze multiple modules in parallel by performing analysis across multiple threads, typically providing a significant reduction in runtime. This needs to be explicitly enabled by using the `-max_thread=`n option. For example, to perform analysis with 4 threads, use `-max_threads=4`. See section 16.9 "Parallel Analysis" in the PC-lint Plus Reference Manual for details.
  - Using precompiled headers. PC-lint Plus supports the use of precompiled headers which can often appreciably improve performance of C++ projects. See Chapter 6 (Precompiled Headers) in the PC-lint Plus Reference Manual for more information.