Getting Started using PC-lint Plus with Code Composer 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.

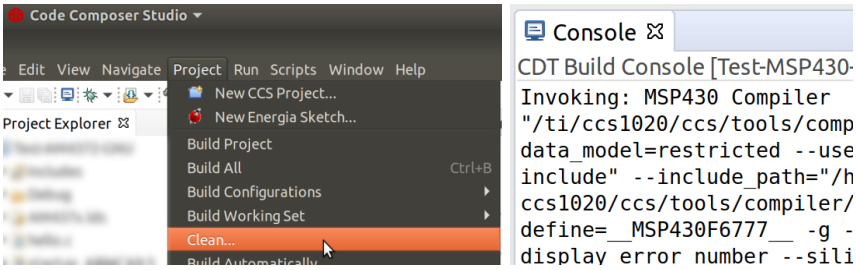
Ensure `pclp_config` dependencies are installed

`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



1. Open the project in Code Composer Studio. In the "Project" menu, select "Clean...".
2. Uncheck both "Clean all projects" and "Start a build immediately".
3. Ensure that the correct project to clean is selected.
4. Press "Clean".
5. In the "Project" menu, select "Build Project".
6. In the "Console" pane, look for an invocation of `armcl`, `cl430`, `cl2000`, or `cl6x` similar to:
   `"/ti/ccs1020/ccs/tools/compiler/compiler-family/cl430" --define=family -i"directory"`

`--include_path="directory"`
Relevant compiler options such as macro definitions and include options should be passed to `pclp_config.py` using the `--compiler-options` option.

If you instead find an invocation of `gcc`, see section 2.3.3 (Creating a compiler configuration for GCC or Clang) in the Reference Manual (`doc/manual.pdf`).

7. Open a PowerShell or Terminal in the directory where you want to store the generated configuration files. Note that `pclp_config.py` is included with PC-lint Plus in the `config` directory.
   - Note: The configuration command includes a single-quoted string with inner double quotes. The Windows Command Prompt does not recognize single quotes, requiring that they are replaced with double quotes and each inner double quote escaped with a backslash. It is recommended on Windows to instead execute the command using the single quote support available in PowerShell.
8. The configuration for this example would be generated with the command:
```
python pclp_config.py
  --compiler=ti_cl430
  --compiler-bin=/ti/ccs1020/ccs/tools/compiler/compiler-family/cl430
  --config-output-lnt-file=co-msp430.lnt
  --config-output-header-file=co-msp430.h
  --compiler-options='--define=family -i"directory" --include_path="directory"'
  --generate-compiler-config
```
to produce a compiler configuration.

Texas Instruments compilers available for use with `pclp_config.py`:

Value for `--compiler`	Description
`ti_cl430`	Texas Instruments `cl430` for MSP430
`ti_cl2000`	Texas Instruments `cl2000` for C2000
`ti_cl6x`	Texas Instruments `cl6x` for C6000
`ti_armcl`	Texas Instruments `armcl` for ARM

Note: `cl6x` vector extensions are not supported.

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:

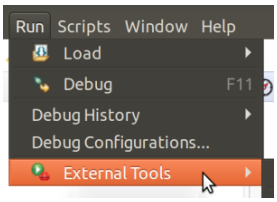
```
PC-lint Plus 2.0, Copyright Vector Informatik GmbH 1985-2022

--- 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 used from within Code Composter Studio by adding it as an External Tool.



1. Open the IDE.
2. Select "Run", "External Tools", "External Tools Configurations...".
   - Note: If "External Tools" is not listed in the "Run" menu:
     (a) Select "Window", "Perspective", "Customize Perspective..."
     (b) Select the "Menu Visibility" tab.
     (c) Expand "Run", scroll down, and check the box next to "External Tools". Resume from step 2.
       – Note: If "External Tools" is not listed inside the "Run" tree in the "Menu Visibility" tab, then open the "Window" menu, select "Preferences", expand "General", "Capabilities", and ensure "ANT Tools" is checked. Resume from step (a).
3. Select "Program" in the left sidebar, then press the "New" icon above the list view.
4. In the "Name" field, enter "Run PC-lint Plus on Selected File"
5. For the "Location" field, select "Browse File System..." and locate your PC-lint Plus executable.
6. In the "Working Directory" field, enter `${container_loc}`
7. In the "Arguments" field, add a new line containing `-i` followed immediately (without an intervening space) by the full path to the directory containing your compiler configuration. E.g. `-i/home/example/pclp/config`. The IDE allows spaces in arguments that are surrounded by double quotes.
8. In the "Arguments" field, add a new line containing the file name of your compiler configuration, e.g. `co-example.lnt`. This file should exist in the directory provided in the previous step.

9. In the "Arguments" field, add a new line and enter `${resource_loc}`
10. Select the "Common" tab and ensure that the "External Tools" checkbox in "Display in favorites menu" is checked.
11. Select "Apply", and then select "Close".
12. Select a source file (not a header file) in the "Project Explorer" pane.
13. Select "Run", "External Tools", "Run PC-lint Plus on Selected File". Output will appear in the "Console" pane.

To analyze a project rather than a single file, follow the above steps but substitute `${resource_loc}/*.c` for `${resource_loc}` when entering the compiler arguments. To run with this external tool configuration, the project directory containing the source files must be selected in the "Project Explorer" pane instead of a source file.

Note: If you receive a variable expansion error from the IDE when running the external tool in the last step, then ensure that the most recent action within the IDE prior to opening the "Run" menu was to select and focus the target in the "Project Explorer" pane.

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.