CCT Generator User Guide

by

Sam Bell
02/06/2015

Reviewed by

Table of Contents

1. Change History

Version	Date	Author	Change Log
D0.1	02/06/1015	Sam Bell	Initial Draft

2. Introduction

This user guide is for the PRQA CCT Generator. This tool wraps the CPG and PRLGCC allowing inspection of a users build to help achieve a CCT compatible with their system. It is designed to work autonomously but is fully configurable to allow users to overcome any issues.

3. Build inspection

The Users build will be inspected for command line options, include paths and defines. This achieved by running the Users build using the PRQA CCT Generator.

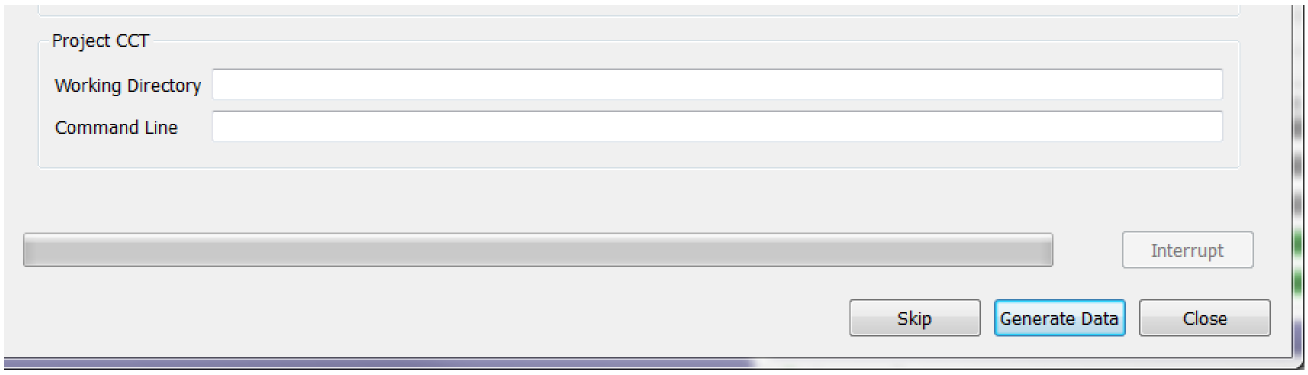


Figure 1

3.1 Generate Data

Figure 1 shows the *Command Line* edit box, in here the User can enter the command necessary to start their build (i.e. using make or a shell or bash script). You should ensure that the build will compile at least one file or no information will be collected. If the command has to be run from a certain location then the working directory can be used to ensure that the make command is run from the correct directory.
Once the correct data has been entered the User should press the *Generate Data* button, this will run the command and inspect the compile lines for data required.
Should the User wish to stop a build (this might be useful for a long build where compile data doesn't change from file to file) the *Interrupt* button can be pressed.

3.2 Skip

If the User has previously generated data for their build they can skip the build step and use the existing data, pressing the *Skip* button will cause the data generation to be bypassed and the next screen will be displayed.

4. Advanced Settings

If the build inspection does not pick up all necessary information then the User may have to use the advanced settings (shown in Figure 2). Selecting the *Advanced Settings* tick box will allow the user to change the inspection configuration.

4.1 File Extensions

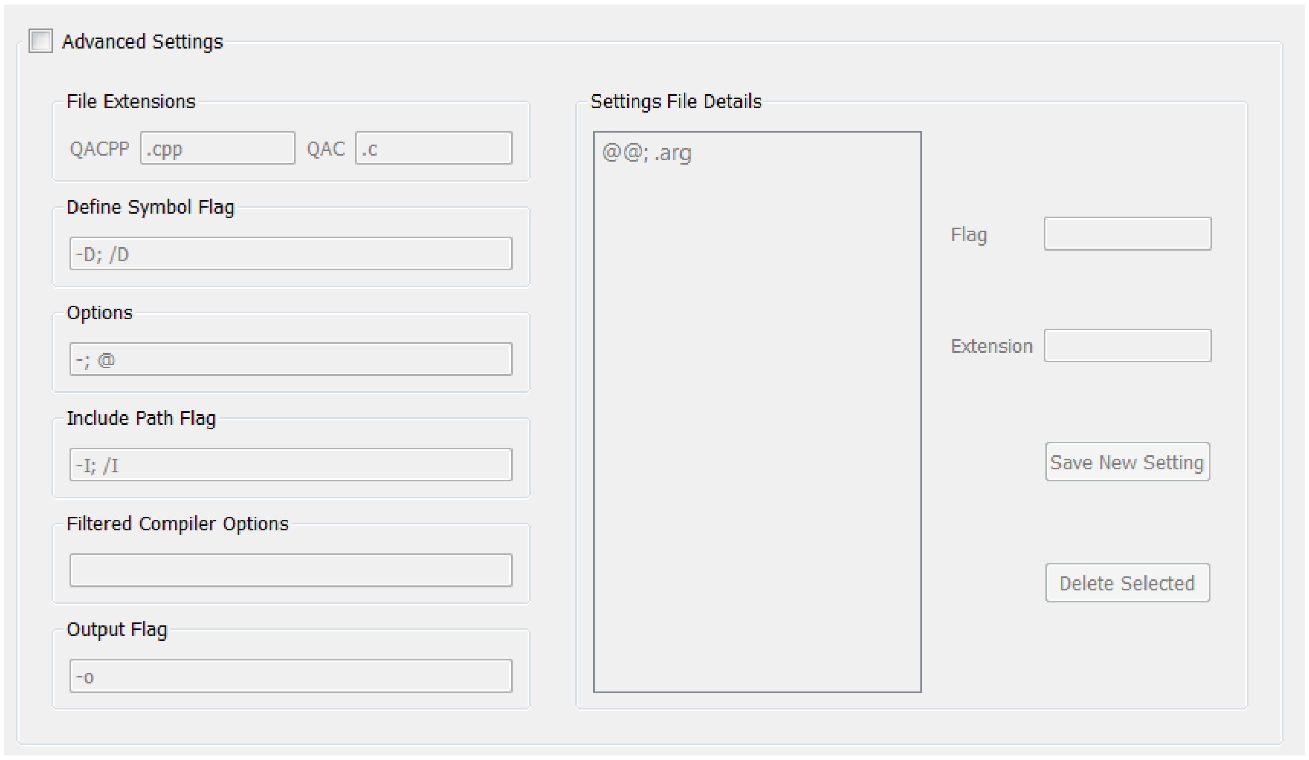
The file extensions tell the system the source extensions to find when inspecting compile lines. They will be associated with either the QACPP CCT or the QAC CCT.

4.2 Define Symbol Flag

This should be the command line flag used by the Users compiler when specifying a macro.

4.3 Options

This should be the delimiting character which defines a command line option for the users compiler.


Advanced Settings

File Extensions

QACPP .cpp QAC .c

Define Symbol Flag

-D; /D

Options

-; @

Include Path Flag

-I; /I

Filtered Compiler Options

Output Flag

-o

Settings File Details

@@; .arg

Flag

Extension

Figure 2

4.4 Include Path Flag

This should be the command line flag used by the Users compiler when specifying an include path.

4.5 Filtered Compiler Options

Should the generated data contain compiler options for which it is known that they are not useful or needed then the compiler option can be specified here and the option will be ignored.

4.6 Output Flag

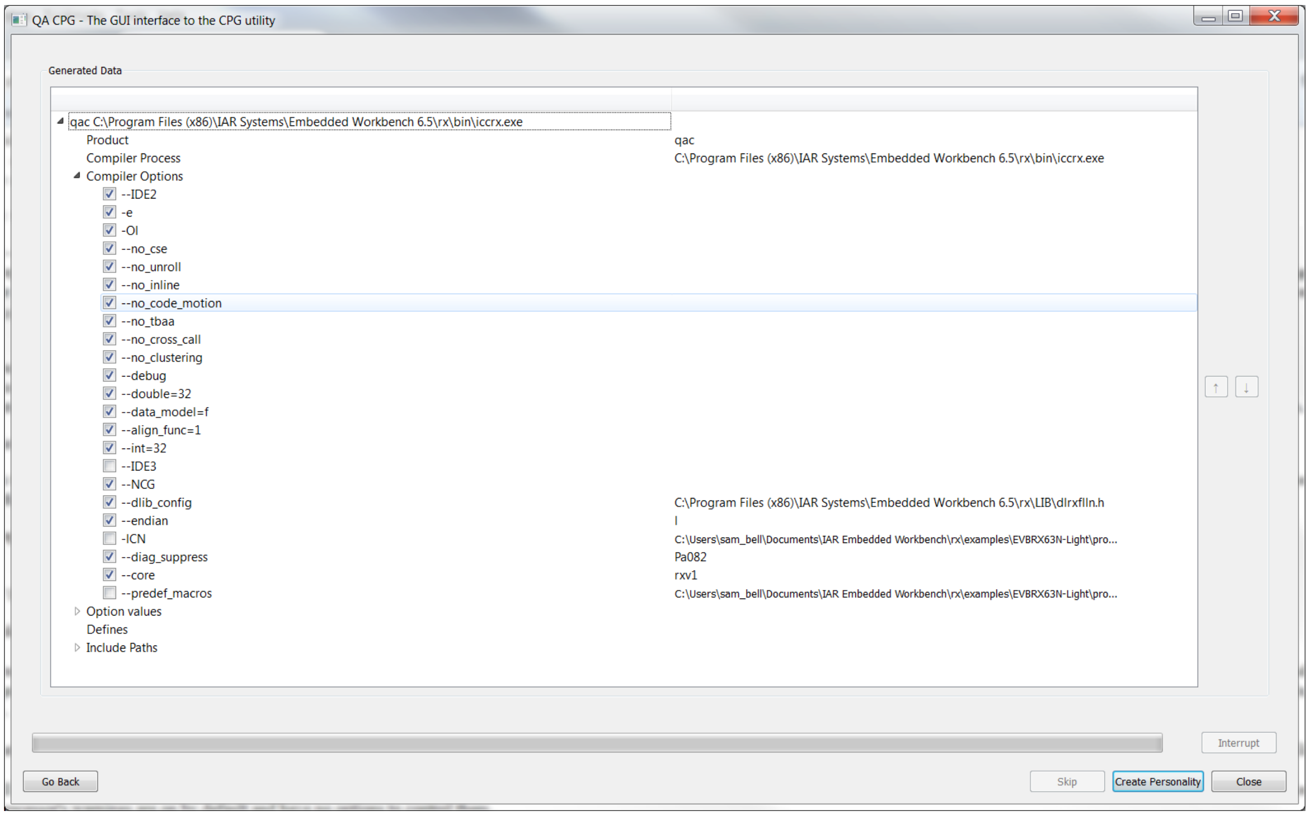
While inspecting the compile line the system will try to work out the correct output flag and object extension. Should the user know the correct output flag it can be entered here.

4.7 Settings File Details

Some compilers, particularly on windows, put their compile line options inside a file. If the users compiler does this the flag and expected extension can be entered here. The file will then be read during inspection and compile options inspected.

5. Personality Creation

The personality creation page will be shown once the data generation has completed (figure 3).



qac C:\Program Files (x86)\IAR Systems\Embedded Workbench 6.5\rx\bin\iccrx.exe	
Product	qac
Compiler Process	C:\Program Files (x86)\IAR Systems\Embedded Workbench 6.5\rx\bin\iccrx.exe
Compiler Options	
✓ --IDE2	
✓ -e	
✓ -Ol	
✓ --no_cse	
✓ --no_unroll	
✓ --no_inline	
✓ --no_code_motion	
✓ --no_tbaa	
✓ --no_cross_call	
✓ --no_clustering	
✓ --debug	
✓ --double=32	
✓ --data_model=f	
✓ --align_func=1	
✓ --int=32	
☐ --IDE3	
✓ --NCG	
✓ --dlib_config	C:\Program Files (x86)\IAR Systems\Embedded Workbench 6.5\rx\LIB\dlrxflln.h
✓ --endian	l
☐ -ICN	C:\Users\sam_bell\Documents\IAR Embedded Workbench\rx\examples\EVBRX63N-Light\pro...
✓ --diag_suppress	Pa082
✓ --core	rxv1
☐ --predef_macros	C:\Users\sam_bell\Documents\IAR Embedded Workbench\rx\examples\EVBRX63N-Light\pro...
Option values	
Defines	
Include Paths	

Figure 3

Depending on the build there could be multiple compiler processes for each product (QAC,QACPP). Each set will show the *Product* for which a CCT will be created and the *Compiler Process.*
The data collected will be displayed under

- Compiler Options
- Option Values
- Defines
- Include Paths

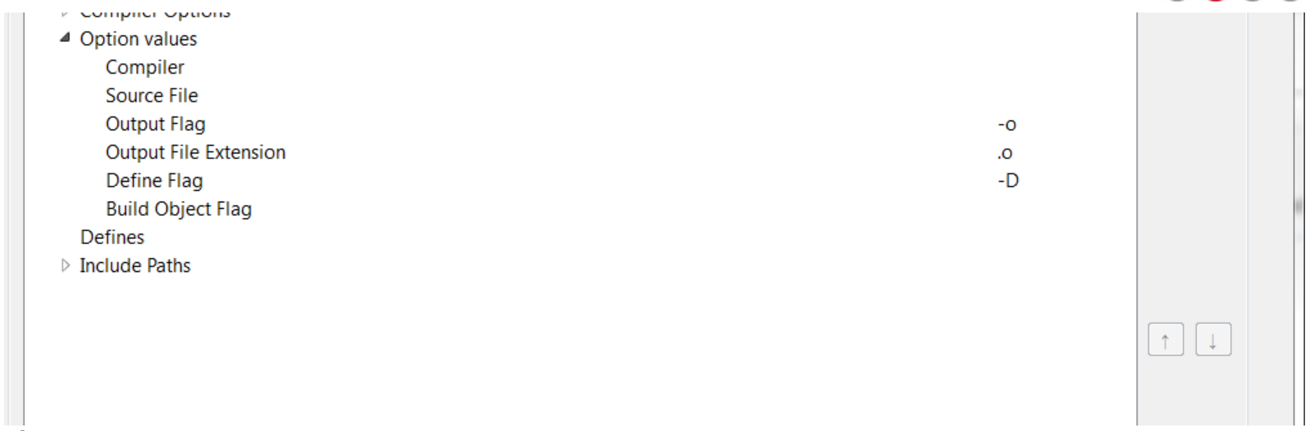
To use the data press the *Create Personality* button, this will start the generation and open an output window which will show progress.
If the user has previously generated a personality then this stage may be skipped using the *Skip* button.

5.1 Compiler Options

All compiler switches and flag/value pairs will be displayed here. Options to be passed on to the personality generator will be ticked.

5.2 Option Values

The Option values will only be used when the CCT is being generated for a non GCC based compiler. For non GCC based compilers the CPG needs information on how the command line works so it can create compiler command lines which run correctly.
Figure 4 shows the options, these can be moved up and down using the arrows in the margin.


Option values

Compiler	
Source File	
Output Flag	-o
Output File Extension	.o
Define Flag	-D
Build Object Flag	

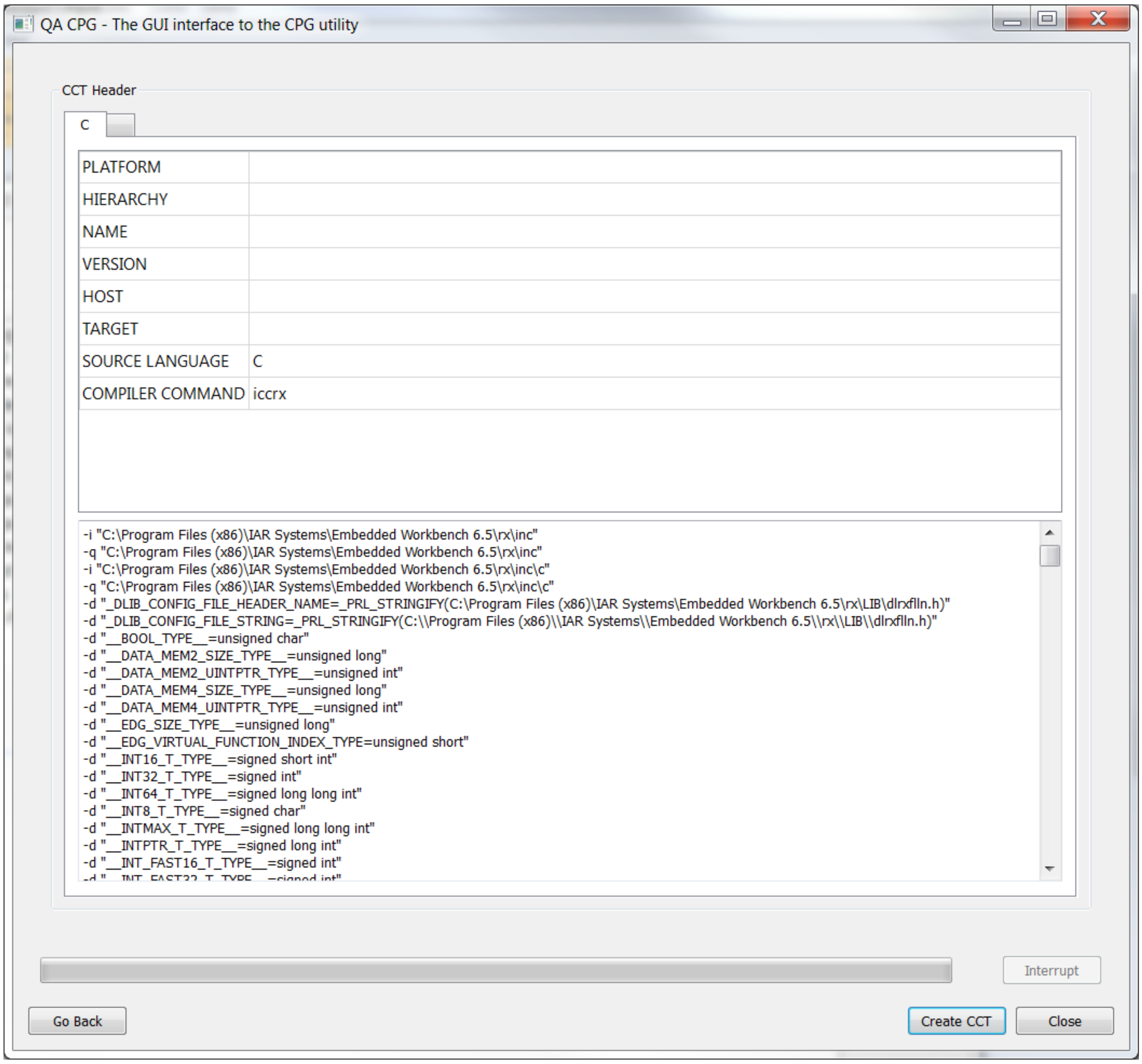
Defines

Include Paths

Figure 4

6. CCT Generation

The CCT Generation page allows the User to create a CCT from the generated personality. Figure 5 shows the Generation page. It will be possible to switch between language types if there are multiple CCTs to create. The top pane contains information regarding the CCT. *Source Language* and *Compiler Command* will always be filled in. Any empty values can be filled in with the relevant details by the user, this will lead to a more descriptive CCT name and more fields on which to filter when choosing a CCT in PRQA Framework.



PLATFORM	
HIERARCHY	
NAME	
VERSION	
HOST	
TARGET	
SOURCE LANGUAGE	C
COMPILER COMMAND	iccrx

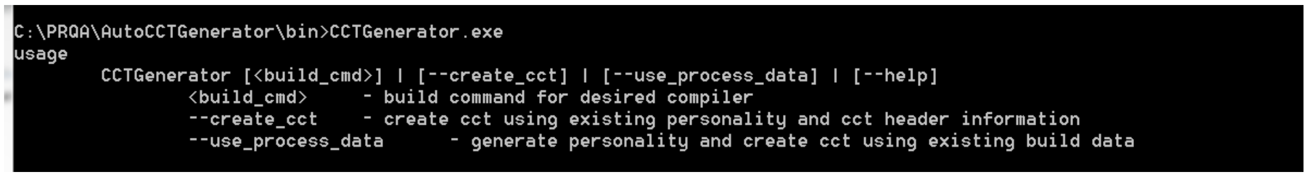
```
-i "C:\Program Files (x86)\IAR Systems\Embedded Workbench 6.5\rx\inc"
-q "C:\Program Files (x86)\IAR Systems\Embedded Workbench 6.5\rx\inc"
-i "C:\Program Files (x86)\IAR Systems\Embedded Workbench 6.5\rx\inc\c"
-q "C:\Program Files (x86)\IAR Systems\Embedded Workbench 6.5\rx\inc\c"
-d "_DLIB_CONFIG_FILE_HEADER_NAME=_PRL_STRINGIFY(C:\Program Files (x86)\IAR Systems\Embedded Workbench 6.5\rx\LIB\dlrxflln.h)"
-d "_DLIB_CONFIG_FILE_STRING=_PRL_STRINGIFY(C:\\Program Files (x86)\\IAR Systems\\Embedded Workbench 6.5\\rx\\LIB\\dlrxflln.h)"
-d "__BOOL_TYPE__=unsigned char"
-d "__DATA_MEM2_SIZE_TYPE__=unsigned long"
-d "__DATA_MEM2_UINTPTR_TYPE__=unsigned int"
-d "__DATA_MEM4_SIZE_TYPE__=unsigned long"
-d "__DATA_MEM4_UINTPTR_TYPE__=unsigned int"
-d "__EDG_SIZE_TYPE__=unsigned long"
-d "__EDG_VIRTUAL_FUNCTION_INDEX_TYPE=unsigned short"
-d "__INT16_T_TYPE__=signed short int"
-d "__INT32_T_TYPE__=signed int"
-d "__INT64_T_TYPE__=signed long long int"
-d "__INT8_T_TYPE__=signed char"
-d "__INTMAX_T_TYPE__=signed long long int"
-d "__INTPTR_T_TYPE__=signed long int"
-d "__INT_FAST16_T_TYPE__=signed int"
-d "__INT_FAST32_T_TYPE__=signed int"
```

Figure 5

The bottom pane shows the settings which will be used when analyzing, this can be checked to see that sensible options have been set.

7. Command Line Interface

The command line interface is provided by the CCTGenerator executable. Figure 6 show the available help.



```
C:\PRQA\AutoCCTGenerator\bin>CCTGenerator.exe
usage
        CCTGenerator [<build_cmd>] | [--create_cct] | [--use_process_data] | [--help]
                <build_cmd>     - build command for desired compiler
                --create_cct    - create cct using existing personality and cct header information
                --use_process_data      - generate personality and create cct using existing build data
```

Figure 6

To run the full process call CCTGenerator with the build command. This will then attempt to run through all processes and create a valid CCT.
If the User wishes to skip the build step then they can run
CCT Generator --use_process_data
This will use the existing data from the GeneratedData.xml file which is located in the bin folder.
If the User wishes to create a CCT from an existing personality the can run
CCTGenerator --create_cct
This will use existing personalities in the GeneratedPersonalities folder.

To change configuration the GUI interface can be used as described above, or for advanced users the xml files can be edited directly.

Appendix A: Glossary

- CCT – Compiler Compatibility Template.
- CPG – Compiler Personality Generator.
- PRLGCC – Compiler Personality Generator for gcc based compilers.
- QAC – C code static analyzer.
- QACPP C++ code static analyzer.

Contact Us

PRQA has offices globally and offers worldwide customer support. Visit our website to find details of your local representative.

Email: info@programmingresearch.com
Web: www.programmingresearch.com