

## **IMPORTANT NOTICES**

### **DISCLAIMER OF WARRANTY**

*The staff of Programming Research Ltd have taken due care in preparing this document which is believed to be accurate at the time of printing. However, no liability can be accepted for errors or omissions nor should this document be considered as an expressed or implied warranty that the products described perform as specified within.*

### **COPYRIGHT NOTICE**

*This document is copyrighted and may not, in whole or in part, be copied, reproduced, disclosed, transferred, translated, or reduced to any form, including electronic medium or machine-readable form, or transmitted by any means, electronic or otherwise, unless Programming Research Ltd consents in writing in advance.*

### **TRADEMARKS**

*PRQA, the PRQA logo, QA C and QA C++ are registered trademarks of Programming Research Ltd.*

### **CONTACTING PROGRAMMING RESEARCH LTD**

*For technical support, contact your nearest Programming Research Ltd authorized distributor. You can contact Programming Research Ltd's head office:*

by telephone on +44 (0) 1 932 888 080  
by fax on +44 (0) 1 932 888 081  
or by e-mail on [support@programmingresearch.com](mailto:support@programmingresearch.com)



# Table of Contents

Introduction .....	5
SCOPE OF CPG USER'S GUIDE .....	5
PRQA PRODUCTS .....	5
WHAT TO EXPECT FROM CPG .....	5
CPG OVERVIEW.....	5
Installing CPG .....	7
WINDOWS .....	7
UNIX .....	7
Starting the wizard .....	8
PRELIMINARY FACTORS .....	8
<i>Knowing your compiler</i> .....	8
<i>Setting the PATH</i> .....	8
NOW FOR THE WIZARD! .....	9
<i>Unix</i> .....	9
<i>Windows</i> .....	9
CPG Wizard: Product & Output .....	10
CPG Wizard: Compiler Command.....	11
THE COMPILER EXECUTABLE.....	11
<i>Cross Compiler and Compiler Flags</i> .....	12
COMPILE COMMAND ORDER.....	13
REDUCE FILE SIZE.....	13
CPG Wizard: Search Directories .....	14
CPG Wizard: Defines & Switches.....	15
CPG Wizard: Review .....	16
"Personality Generated" ...? .....	17
SAVE COMPILER DETAILS .....	18
SAVE LOG .....	18
What next?.....	19
RESULTING PERSONALITY EDITS .....	19
TRY THE PERSONALITY IN A SMALL PROJECT .....	19
Appendices .....	21
APPENDIX A: TROUBLESHOOTING / FAQs .....	21
APPENDIX B: CPG FROM THE COMMAND LINE .....	28
<i>Unix</i> .....	28
<i>Windows</i> .....	28
<i>Sample command line usage</i> .....	29
<i>Sample script file format</i> .....	30
<i>Sample batch file format</i> .....	30



# Introduction

## Scope of CPG User's Guide

CPG stands for Compiler Personality Generator, which is a wizard-driven application that can be used for the PRQA tools QA C and QA C++.

The purpose of this document is to provide guidance on how to use CPG to generate a Compiler Personality for you to use with a PRQA static analyser.

## PRQA Products

Both QA C and QA C++ are highly configurable. One of the configuration facilities is known as the Compiler Personality, which is used to specify options that reflect the configuration of your compiler. Each compiler that you use must have a Compiler Personality.

You can set up the Compiler Personality yourself, by following the detailed instructions given in the appropriate User Guide. Alternatively, you may prefer to use our Compiler Personality Generator tool (CPG), which partially automates the process.

## What to expect from CPG

The CPG tool asks you some questions and extracts information from your environment in order to generate a compiler personality. You will still need to review the generated personality yourself.

## CPG Overview

CPG has two executable files, `AutoCmpPersonGen` and `AutoCmpGen`.

`AutoCmpPersonGen` is an interactive interface between the user and the complex command-line setup for the personality generator itself, `AutoCmpGen`.

We do not recommend running `AutoCmpGen` directly from the command line, especially for inexperienced users. However, there is

a help file `AutoCmpPersonGen.htm` in the help subdirectory of the CPG installation area.

The general operation of the personality generator is to scan certain directories, creating a list of all the tokens found in the files, whether text-based or binary-based. CPG then tests its list of tokens against its test code to determine its type and how it is used, using the information thus gained to create a personality.

# Installing CPG

## Windows

The Compiler Personality Generator for Windows is packaged as an InstallShield executable. Run the executable to install the components, and you will be provided with a Start menu shortcut to the Compiler Personality Generator program.

You have the option of selecting where the installation is placed.

## Unix

The Compiler Personality Generator for Unix is packaged as a tar archive file containing the relevant executables. Untar this archive into the directory where you have installed QA C/ QA C++: for example `/usr/local/prqa`

# Starting the wizard

## Preliminary factors

Before starting the wizard, there are some important preliminary factors for you to take into consideration.

## Knowing your compiler

In order to use CPG, you'll need to know how to run your compiler successfully from the command line on the machine that you are using. You may need to refer to your compiler documentation for this, as you may need to set environment variables, or to run a configuring batch/script file, before the compiler is ready to be used directly from the command line.

## Setting the PATH

Before running the CPG utility, you should ensure that your compiler is on your system PATH, so that CPG can invoke the compiler internally.

## Unix

On Unix systems, set the PATH variable to the `bin` directory of your compiler.

## Windows

On Windows, set the system PATH environment variable in the system properties.

For example, on Windows XP:

Control Panel > System > Advanced > Environment Variables  
and Edit the Path, by appending a semi-colon and the path to  
wherever you have located the compiler executable.

## Now for the wizard!

At the top of each page in the wizard, there is some text explaining the required user input fields.

### Unix

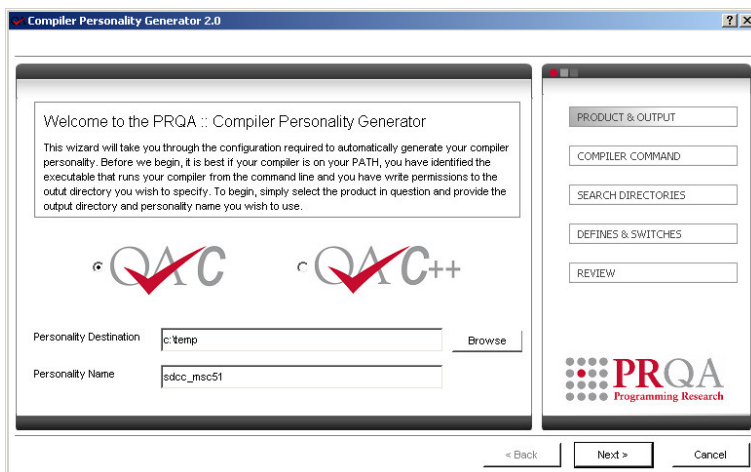
To launch the wizard, run `AutoCmpPersonGen` from the installation bin directory.

### Windows

To launch the wizard, go to

Start > Programs > PRQA > Compiler Personality Generator  
which will run `AutoCmpPersonGen.exe` for you.

## CPG Wizard: Product & Output



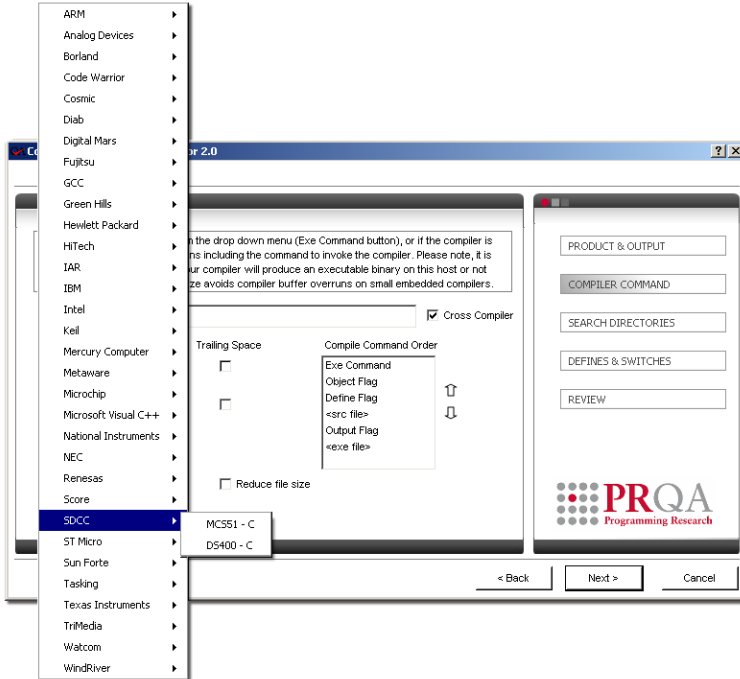
Select either QA C or QA C++ as the product for which you are generating the current personality. Even if your compiler supports both languages, you must generate a personality for each one.

Specify the output directory. This is where the the generated personality will be saved, as will the optional force include file / substitute headers (*see Appendix A for more details*). The directory must already exist: if your entry is displayed in **red**, then it doesn't exist.

Name this personality. Hint: it tends to be helpful if you indicate meaningful factors such as the compiler and version number as part of the name.

Click the "Next" button.

# CPG Wizard: Compiler Command



## The Compiler Executable

CPG must be able to run your compiler successfully, as CPG tests the compiler by creating and compiling test code files. This means that CPG must be able to **find** your compiler executable.

Select the “Exe Command” button, and select your compiler and version in the drop-down list. Note that the list is filtered, based on the product selection that you made on the Product & Output screen.

If your exact compiler isn't listed, select the closest possible. Some of the compiler options may need manual modification. If your compiler isn't in the list at all, you will have to enter all the compiler options manually. Refer to your compiler documentation if necessary.

If you select a compiler from the list, then the options for it will be entered automatically, and you can see them appear on the display.

If the compiler command displays in **red** then CPG hasn't been able to find that compiler executable. *Refer to Appendix A if you need help.*

## Cross Compiler and Compiler Flags

If the "Cross Compiler" checkbox is checked, then all test results will be on the basis of compilation only.

If you wish CPG to produce a program which can execute on the computer on which you are running CPG, leave the "Cross Compiler" checkbox blank. Many tests will be carried out, trying to compile, link and execute a file on your machine.

What goes in the other boxes depends on the idiosyncracies of your chosen compiler. You may be able to leave the other boxes blank.

## Output Flag

If the "Cross Compiler" checkbox is checked, this entry will not be used, as it specifies the flag used by your compiler to denote output to a file.

## Object Flag

CPG always needs to know if your compiler has been able to produce an object file, regardless of whether you have specified a cross-compiler or not.

Some compilers will delete an object file unless they have been instructed otherwise. If your compiler does not delete any object file that it may have been able to produce, you can leave this box blank. Otherwise, use the appropriate flag for your compiler.

## Define Flag

This specifies the flag or switch that your compiler uses for the defines, e.g. `-d -D` etc.

## Object File Extension

This specifies the extension of the object file(s) produced by your compiler. Note that the leading '.' character is required.

## Trailing Space boxes

Some compilers expect a space between the compiler flag and the flag text entries on the command line; some will not allow that; others don't care.

e.g.

```
-D CPU=XXXX <trailing space after -D>
```

```
-DCPU=XXXX <no trailing space after -D>
```

## Compile Command Order

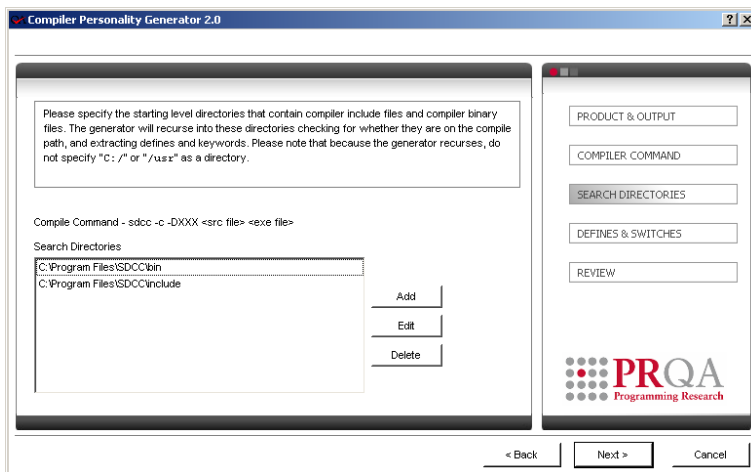
Some compilers must have the command line information presented to them in a particular order. If necessary, you can adjust the order here in order to match your compiler's expectations.

## Reduce File Size

In most cases, it is unnecessary to make use of this option. If you know your compiler has difficulty handling large files, switch this option on, and CPG will refrain from creating test source files that contain a large number of pre-processor conditionals.

Click the "Next" button when you're ready to move on to the next screen. If nothing happens when you do, this means that there is a problem relating to the Exe Command dialog box.

## CPG Wizard: Search Directories



Here, you specify the starting level directories in which CPG will look for tokens. CPG needs to know the toplevel directory for the compiler system header files, and the toplevel directory for the compiler binaries.

CPG will scan any sub-directories below the specified starting directory, so there is no need to specify each sub-directory, only the toplevel directory.

It follows that, precisely because CPG does scan any files in and below the specified starting directory, if there are many irrelevant files in that hierarchy, it is going to take a long time (perhaps hours). Where feasible, you should be using dedicated areas, so that this issue does not arise.

Click "Next" when you're ready to move on to the next screen.

# CPG Wizard: Defines & Switches

If, with your chosen compiler, you need to specify external switches or defines to ensure successful compilation, or to enforce a particular route through the pre-processing of system header files, then you can do so here.

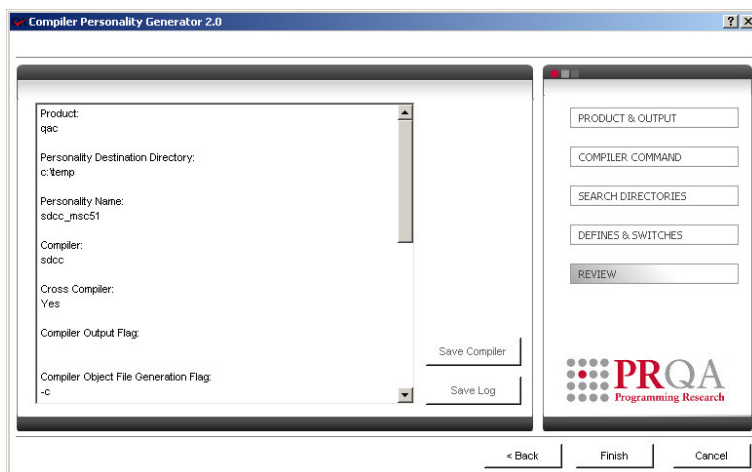
Specify any compiler switches or defines on separate lines, or by comma-separated values.

In general, when an executable file is being created on the machine being used, these options are not required, and can be left blank. There may be occasions when debugging or optimization switches are added.

On cross-compilers, producing no executable, a target chip is often required. If your compiler has a specific switch for the target, it should be added here.

Click “Next” when you’re ready to move on to the next screen.

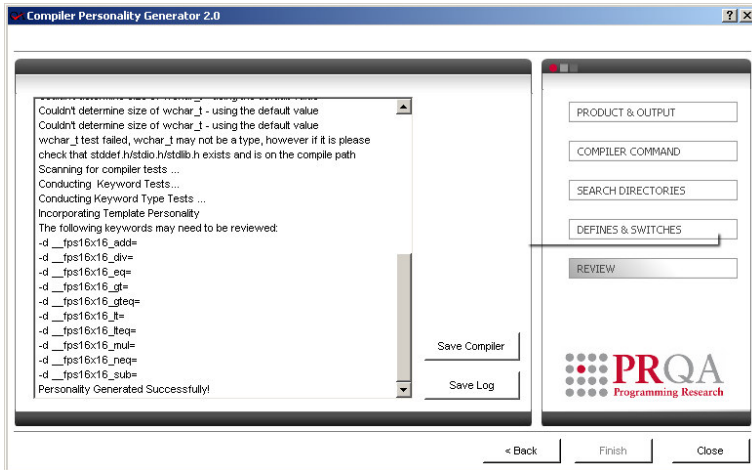
## CPG Wizard: Review



The Review screen provides a summary of all the settings you have specified.

When you've satisfied yourself that these are as you would expect, press "Finish" to end the wizard and let CPG run.

## “Personality Generated” ...?



While CPG is trying to create the personality for you, you'll be able to see occasional messages in the panel on the Review screen.

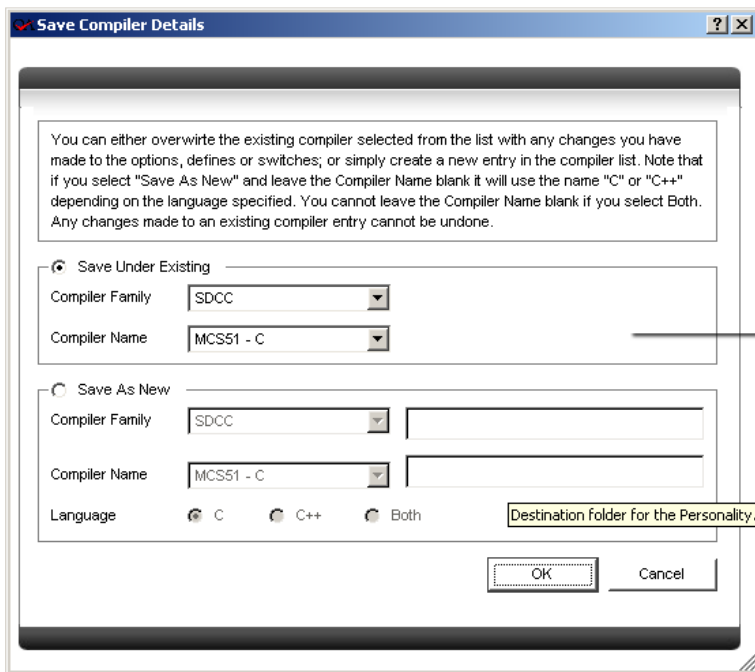
These messages constitute the Log for this run, and, when CPG has finished, the “Save Log” button will be enabled.

In due course, you should see a message as in this screenshot, stating “Personality Generated Successfully!”.

If CPG returns “Personality File Generation Failed”, then you need to review the details on the Review screen. Ensure that the compile command is formatted correctly, with the correct options.

If you cannot discover the reason for a failure, then take some screenshots of the options you have used, and send these and the log to [support@programmingresearch.com](mailto:support@programmingresearch.com)

## Save Compiler Details



The “Save Compiler” button will be enabled once CPG has finished running. If the compiler that you used was one that you selected from the menu, with no additional changes, then there is no need to “Save Compiler”.

If this is a new compiler, or you have changed the settings, then it is better to save the data.

## Save Log

Saving this log is optional, but it is generally recommended.

## What next?

After you see the message that the personality has been generated successfully, what do you do with it?

You can load the generated personality file into QA C or QA C++ by going to the section in which, when creating a new project, you are prompted to select personalities. However, ....

## Resulting Personality Edits

...it is recommended that you open the personality file with a text editor and check over the entries, before loading the personality file into a QA C / QA C++ project.

The log may note particular items – such as keywords, data paths, include items – which it suggests may need reviewing.

## Try the personality in a small project

Load the generated personality file into a small QA C / QA C++ project. If there are “syntax errors” reported during analysis carried out using the generated personality file, then can you determine the reason for them?

If there are syntax errors, and after reviewing everything as indicated above, you cannot determine the reason for them, then send the CPG log, the personality file, and the pre-processed source file from the analysis to PRQA at [support@programmingresearch.com](mailto:support@programmingresearch.com), for further investigation.

This procedure is intended to give you confidence in your generated personality, before using it in a fullsize project.



# **Appendices**

## **Appendix A: Troubleshooting / FAQs**

### **Write permissions**

Do you have write permissions for any necessary directories? such as the one where you'd like the generated personality saved.

### **Button greyed out**

There's something wrong on the current screen – either an error or an omission. Is any entry displayed in red?

### **Red entry in Personality Destination dialog box**

This means that the directory doesn't exist. Have you made a spelling mistake, perhaps?

### **Red entry in Exe Command dialog box**

This means that CPG cannot find the compiler executable. The compiler executable must reside on the computer on which you are running CPG.

Can you run the compiler executable from the command line? If so, it may be that the PATH does not include the way to the compiler executable.

It may be that the entry in the Exe Command dialog box contains spaces e.g. C:\Program Files. If so, try enclosing the string in quotes.

### **I'm stuck with a red compiler – what next?**

"When I select a compiler from the menu or when I try to enter the name directly, its name appears in red, and I am unable to proceed to the next step."

CPG tries to verify the compiler location upon a menu selection or data entry by searching the PATH settings. In the event that the compiler is not found, its name will appear in red.

CPG's tests include running the target compiler, which must be installed on the computer where CPG is being run, and the PATH settings must be such that CPG can find the compiler executable.

## ***I have checked my settings but ...***

"The CPG error window says to check my settings. I have checked my settings but I still get errors, so what do I do next?"

The error window is usually the result of incorrect compiler settings. CPG calls the compiler with a test file and then it looks for the created object file. If the object file has not been created, or its extension is incorrect, the warning window appears.

The reasons for the error vary. Here are some steps that you could try:

- Inspect the compiler switch settings again, especially if they been entered by hand.
- Create a small target source file that contains an empty `main` function and then run the compiler in a DOS or terminal window with the correct switches, trying to compile this file.
- If the compiler does not run correctly, the error information should appear in the terminal window. Common problems include: no valid licence, no target chip settings (cross-compiler), missing defines.
- When the compiler runs correctly with this file, verify that the object extension is correct. Cross-compilers very often do not use the familiar ".obj" or ".o" extensions.

## What about intrinsic register and port variables?

“Will CPG handle intrinsic register and port variables correctly?”

No. The reason is that these tokens rarely appear in the compiler's header files or in the compiler binary file. In the event that the program source contains these variables, they should be declared in a `forceinclude` header file.

```
extern int _EAX;
```

## What is a “forceinclude” file?

“I see references to a `forceinclude` file. Do I have to write this, or does my compiler supply it?”

A `forceinclude` header file is a header file that only the static analyser uses. It is not seen by the compiler.

A `forceinclude` file typically contains function prototypes for compiler intrinsic functions, and special “defines” required for QA C or QA C++. It is also possible that, if register or port variables are used, their declarations will need to be placed in this file.

A `forceinclude` file may already exist for the target compiler. If it does exist, simply add any required declarations or defines to it. If it does not exist, it may be necessary to include further details in a `forceinclude` file for compilers that are not in the list (of supported compilers), or that do not already have entries.

## How do I create a “forceinclude” file?

In CPG's “ForceIncludeFiles” subdirectory, create a header file and place in it any required declarations. The name of the header file is then added to `compilers.txt`, which is located in CPG's bin directory. In the block of options for the target compiler there should be a line that looks like:

```
<ForceIncludeFile></ForceIncludeFile>
```

Insert your header name here

```
<ForceIncludeFile>my_force_include.h</ForceIncludeFile>
```

Run CPG again by re-selecting the target compiler from the menu. The new forceinclude header will now be placed in the proper location with respect to the target personalities.

It is also possible to add the forceinclude file directly to the previously-created personality. Place an entry at the bottom of the personality in C:\Program Files\PRQA\QACxxxxx\personalities.

It will look similar to the below entry, but with the correct data.

```
-q "C:\ ... \prlforceinclude"  
-fi "C:\ ... \prlforceinclude\generic.h"
```

## **Where are the compiler include files?**

"I have successfully created a personality, but the compiler include files do not appear in it. What is wrong?"

There are occasions when, even though the header locations were added to CPG's search path, the header information does not appear in the created personality. An indication that this is going to happen can be seen in the output window while CPG is running. Normal operation is shown below.

```
Scanning All Directories ...  
Scanning for compiler include directories ...  
/usr/local/share/sdcc/include/  
/usr/local/share/sdcc/include//mcs51  
The order of these header directories may need to be  
reviewed.  
Scanning for compiler defines ...
```

As you can see, the include paths are shown in the output. In the event that you see this:

```
Scanning All Directories ...  
Scanning for compiler include directories ...  
Scanning for compiler defines ...
```

It means that the include paths are missing, and the resultant personality will also be missing the include information.

It appears that the compiler is having trouble finding the include files. Check the compiler documentation and ensure that any required environment variables have been set.

With some compiler installations, it has been found necessary to add the include path to the compiler command line. This is done in the compiler dialog window, the one containing the switch information. Add a “-i” or “-I” entry to the compiler executable dialog entry. Add quote characters if the path contains spaces.

```
sdcc -I“c:\my sdcc compiler\include“
```

## **How have I created a dual personality?**

“I successfully created one personality by selecting it from the CPG menu. Then, for a second compiler, I entered the information directly. The new personality contains information from the previous compiler. What is wrong?”

When CPG is first run, it saves a file called `settings.txt` in the directory where the program executable is located. This file contains all the settings details. When CPG is run again, it uses the settings in this file.

The workaround is to delete or modify `settings.txt` before re-running CPG.

## **Why am I having difficulty entering compiler details manually?**

The settings file also contains the last-used-template file, force-include file and the substitute header information. This information cannot be entered via the wizard, as it is normally retrieved from the `compilers.txt` menu file.

"I've edited and saved `compilers.txt` but CPG is not seeing my changes."

## Why does CPG take so long?

"I've included the location of the compiler header files and the binary file in the search path, but CPG takes **hours** to complete."

CPG can scan binary files, in addition to scanning the usual header files. This increases CPG's ability to detect additional tokens and "defines" that may not be in the compiler's header files. CPG will scan every binary file in the target directory. Clearly, the more files there are in the directory, the more time will be required to scan and process the tokens. In situations where the compiler binary is in a common directory with many other programs, such as the `/usr/bin` directory of Unix operating systems, there may be thousands of files.

Hints:

Do not include the compiler binary location if it has many files unrelated to the compiler.

Do not include the compiler binary location if CPG is generating many tokens that appear unrelated to the selected language (C or C++)

If it is desirable to include the compiler binary files, inspect the directory system for the compiler. Quite often embedded compilers will split the locations. The "starter" program for the compiler may appear in a `\bin` directory but the preprocessor and actual compiler may be located in a different location. Both locations should be included in the CPG search path.

## It's changed my personality!

"I used CPG to create a personality, and then modified it in the QA C / QA C++ GUI, and it's all been changed around."

Don't worry about this – it is only the appearance that is different. Technically, the personality is what you would expect



## **Appendix B: CPG from the command line**

It is much easier to use the CPG wizard than to run CPG from the command line.

However, it is possible that you might want to run CPG from the command line, for instance when you want to create a number of personalities of the same compiler, with only minor changes in the settings.

For both Unix and Windows, run AutoCmpGen (see below) to obtain the usage information, and worked out what usage options you should be specifying. Then re-run AutoCmpGen with your chosen options specified.

### **Unix**

To run CPG from the command line on a Unix machine,

```
cd <CPG install location>/bin
```

and run the executable

```
./AutoCmpGen
```

This will provide the necessary usage information about the required command line arguments.

### **Windows**

To run CPG from the command line on a Windows machine,

```
cd <CPG install location>/bin
```

and run the executable

```
AutoCmpGen
```

This will provide the necessary usage information about the required command line arguments.

Creating a command line to run the "AutoCmpGen" program can be quite difficult, due to the length of the command and all the options that are required.

The most common problems are listed below:

There are two kinds of switches, required and optional. In some cases a required switch will actually be empty in the GUI dialog entry. In these instances a placeholder is required for the switch in command line. A pair of double quotes "" is acceptable.

The <compile command order> argument requires double quotes around the string.

Double quotes are also recommended around other arguments that contain path information. If the path contains spaces, double quotes are essential.

The <compile command order> string requires the double % characters.

## **Sample command line usage**

Shown below are sample scripts for the command line call to AutoCmpGen for an embedded compiler that also requires substitute headers.

## Sample script file format

This should be a single line in the script file.

```
AutoCmpGen qac ccmp sdcc "" -c -D .rel "%c %j %d%s  
%o%e" -i "/usr/local/share/sdcc/include/"  
-op "/usr/prqa/qac-7.0/personalities/cpg-auto-test/auto-  
sdcc.p_c"  
-xs "-mmcs51" -tpl "../TemplatePersonalities/sdcc.tpl" -  
fi "../ForceIncludeFiles/generic.h" -sub  
"../SubstituteHeaders/sdcc"
```

## Sample batch file format

This should be a single line in the batch file.

```
AutoCmpGen qac ccmp sdcc "" -c -D .rel "%c %j %d%s  
%o%e" -i "C:\Program Files\SDCC\include"  
-op "C:\prqa\QAC-7.0\personalities\auto-sdcc.p_c" -xs "-  
mmcs51" -tpl "..\TemplatePersonalities\sdc.tpl"  
-fi "..\ForceIncludeFiles\generic.h" -sub  
"..\SubstituteHeaders\sdc"
```