



Programming Research Limited

Compiler Wrapper for QAC/QAC++

Version 2.6

Author: C. Waddingham
Version: 2.6
Status: Release
Date Created: 6th February, 2004
Last Modified: 16th June 2008

1. Table Of Contents

1. TABLE OF CONTENTS INTRODUCTION	3
1. INTRODUCTION	4
2. OVERVIEW	5
3. USAGE	6
4. CONFIGURATION FILE	7
4.1 Mandatory Configuration Options	7
4.1.1 COMPILER_SETTINGS_FILE	7
4.1.2 ANALYSER	7
4.1.3 ANALYSER_BASE	7
4.2 Optional Configuration Options	7
4.2.1 FILELIST	7
4.2.2 POST_ANALYSIS	8
4.2.3 INTEGRATION_TYPE	8
4.2.4 ANALYSER_FLAGS	8
4.2.5 ERRDSP_FLAGS	8
4.2.6 LOGFILE	8
4.2.7 DEBUG	9
4.2.8 STOP_ON_FAIL	9
4.2.9 SLEEP_AFTER_ANALYSIS	9
4.2.10 OUTPUT_DIR	9
4.2.11 DEFINE_ON_OPTION	10
4.2.12 INCLUDE_OPTION	10
4.3 Environment Variables in Configuration Options	10
5. CHANGE HISTORY	12
5.1 Version 2.4	12
5.1.1 Change Requests	12
5.2 Version 2.5	12
5.2.1 Change Requests	12
5.3 Version 2.6	12
5.3.1 Change Requests	12

2. Introduction

This document describes the operation of the compiler wrapper for QAC and QAC++.

The wrapper can be used with command line versions of either QAC or QAC++ on either Windows or Unix.

3. Overview

The wrapper is a utility that can be used alongside your compiler in order to perform the function of both analysis and compilation. Analysis is performed by either QAC or QAC++, and compilation is performed by the compiler as before. The command line passed to the wrapper is passed on to the compiler when invoking compilation.

Let us consider the use of the wrapper in a simple makefile.

This makefile is used to compile a “hello world” C program:-

```
.c.o:
    $(CC) -c $<

hello_world: hello_world.o
    $(CC) -o $@ $<
```

On Unix systems, we could use the wrapper by executing

```
make CC="wrapper.pl -wcf wrapper.cfg gcc"
```

This will replace the original compilation with a two step process of analysis and compilation. When the wrapper invokes the compiler, it extracts from the command line parameters any `-D` and `-I` options that need to be passed on to the analyser (QAC or QAC++).

On Windows, use `wrapper.exe` instead of `wrapper.pl`.

The behaviour of the wrapper is configured by configuration files which are passed to the wrapper script using the `-wcf` flag.

4. Usage

The wrapper expects the following options

```
wrapper.pl -wcf file1.cfg -wcf file2.cfg -wcf file3.cfg ....  
<compiler> <compiler command line>
```

The contents of each config file is described in section 4.

The wrapper is versatile enough to be used in both IDE integrations and makefile integrations.

Typical usage might be.

```
wrapper.pl -wcf qac_5.0.1.cfg -wcf gcc3.2.3.cfg -wcf sniff.cfg -  
wcf projectX.cfg gcc -c test.c
```

In this example, a separate wrapper configuration file has been created to describe settings prevalent to the analyser, the compiler, the type of integration and project specific coding standards.

The wrapper can also accept values for any of the configuration options which might be set in the configuration file. This is achieved by the `-set` flag, for example:

```
wrapper.pl -set "ANALYSER_FLAGS=-fi force.h" -wcf test.cfg gcc -  
c test.c
```

This would set the `ANALYSER_FLAGS` configuration option to `"-fi force.h"` before the first configuration file `test.cfg` was examined. This is very useful if you wish to pass values which are determined during the make process to the wrapper.

The wrapper will accept spaces between `-wcf` and `-set` flags and their arguments, but these are optional and can be omitted if your build system will not allow multiple instances of the same flag to be passed to a compiler-like process.

5. Configuration File

An example configuration file for the wrapper can be found in wrapper.cfg.

The wrapper has 3 mandatory settings as well as 9 optional settings.

When multiple configuration files are specified to the wrapper, the last applied settings will override previous settings with the exception of ANALYSER_FLAGS and ERRDSP_FLAGS.

If the wrapper is not supplied with the 3 mandatory options it requires to run, it shall complain about missing settings.

5.1 Mandatory Configuration Options

5.1.1 COMPILER_SETTINGS_FILE

The COMPILER_SETTINGS_FILE contains the path to the compiler personality for the specified compiler using the specified ANALYSER. On Windows use forward slashes for the path.

5.1.2 ANALYSER

The ANALYSER should be set to either QAC or QACPP depending on whether the analyser that you are using is QAC or QAC++.

5.1.3 ANALYSER_BASE

The ANALYSER_BASE setting provides the directory in which the analyser you wish to use is installed. Do not place a trailing slash on the path. On Windows use forward slashes for the path. Note it is possible to apply a current environment variable e.g.

```
ANALYSER_BASE=$(QACPATH)
```

Or

```
ANALYSER_BASE=$(QACPPPATH)
```

5.2 Optional Configuration Options

5.2.1 FILELIST

This setting controls whether an output file containing the names of each analysed file (along with the location of the output directory) is produced. The filelist should be the name of the desired filelist file.

It will always be created in the output directory, and contain only the names of analysed files whose output will be stored in that directory. For Makefiles with a number of output directories, a number of these filelist files will be created.

The filelist files are used to launch the QAC message browser, or any project wide analysis and reporting, such as prjdsp.

If omitted, no filelist file will be created.

5.2.2 POST_ANALYSIS

This setting controls the execution of a post-analysis program, which may be run after each analysis. At present, it only supports the operation of MISRA post-analysis as supplied in the Programming Research MISRA Compliance Module for QAC.

If set, the post-analysis program will be run after each analysis, with the output directory and the name of the file to be analysed set by the wrapper.

If omitted, no post-analysis will take place.

5.2.3 INTEGRATION_TYPE

This setting controls how the results for an analysis are displayed. The values for this are as follows

- Unset – if INTEGRATION_TYPE is unset then the wrapper will just produce error file and met files, but won't generate any output. This is the default value.
- **STDERR** – produce output to standard error (used for IDE integrations)
- **TEXT** – produce output to a text file in the output directory. It shall be named the same as the source file with a .txt suffix.
- **HTML** - produce output to a html file in the output directory. It shall be named the same as the source file with a .html suffix.
- You can also combine any of the strings STDERR, TEXT and HTML if you want more than one output, so **STDERR_TEXT_HTML** would produce all three.

The output produced can be tailored by altering **ERRDSP_FLAGS**. For example, the `-format` option to `errdsp` can be used to produce the required output format for an IDE such as Sniff or visual studio.

5.2.4 ANALYSER_FLAGS

ANALYSER_FLAGS can be used for specifying additional command line flags that you wish to pass to the analyser. For example, you may wish to produce pre-processed output for further debugging of hard errors.

It is possible to accumulate ANALYSER_FLAGS from multiple configuration files using the syntax

```
ANALYSER_FLAGS=$ANALYSER_FLAGS -ppl+ -ppf+
```

5.2.5 ERRDSP_FLAGS

ERRDSP_FLAGS can be used for specifying additional command line flags that you wish to pass to `errdsp`. For example, you may wish to add in a message personality as your coding standard using the `-via` option, or specify a message format string for a particular IDE.

It is possible to accumulate ERRDSP_FLAGS from multiple configuration files using the syntax

```
ERRDSP_FLAGS=$ERRDSP_FLAGS -ppl+ -ppf+
```

5.2.6 LOGFILE

LOGFILE if set should contain a path to a log file that will store the status of the operation of the wrapper. On Windows use forward slashes for the path. If LOGFILE is not set, then messages are sent to stdout.

5.2.7 DEBUG

If DEBUG is set to 1 then additional debugging information is sent to the LOGFILE or stdout.

5.2.8 STOP_ON_FAIL

STOP_ON_FAIL controls the behaviour of the wrapper if the initial analysis fails. The values for this are either 0 or 1.

If STOP_ON_FAIL is set to 0 (the default) then compilation will continue after analysis regardless of the outcome of the analysis.

If STOP_ON_FAIL is set to 1 then the wrapper will stop if an analysis is not successful. This can be used if you want to make sure that you can analyse all your files using the wrapper. By setting STOP_ON_FAIL, a makefile will stop building when it encounters a file that will not analyse.

5.2.9 SLEEP_AFTER_ANALYSIS

SLEEP_AFTER_ANALYSIS pauses the execution of wrapper after analysis for a specified time in seconds. This can be used in situations where continual analysis causes overheating of the CPU.

Allowable values for this are any positive floating point number. For example:

```
SLEEP_AFTER_ANALYSIS=0.25
```

will cause wrapper to pause for a quarter of a second after analysis. By default the setting is 0 i.e. no pausing after analysis

5.2.10 OUTPUT_DIR

Typically a Makefile gets the compiler to output the object files in the current directory alongside the source files, or in a separate 'object' directory. This option allows the QAC/QAC++ analysis output to be sent to a different directory. This can be useful if the output of different compilations need to be stored, for example a single threaded and multi-threaded build. The OUTPUT_DIR option specifies a name for the output directory: note this option only works if you specify an output dir in the compile command. The parameter to OUTPUT_DIR is the name of the directory to put the analysis output in.

OUTPUT_DIR may be supplied as an absolute path or a relative path.

When supplied as an absolute path, the absolute path provided prefixes the -o path passed to the compiler by Make. For example:

Supply setting:

```
OUTPUT_DIR=/home/new_output_root
```

Makefile at /home/build/Makefile does:

```
$(CC) -c file.cpp -o output/file.o
```

Then QAC/QAC++ output will appear at:

```
/home/new_output_root/output/file.err etc.
```

When OUTPUT_DIR is supplied as a relative path it is added as a suffix to the -o path passed to the compiler by Make. For example:

Supply setting:

```
OUTPUT_DIR=./new_output
```

Makefile at /home/build/Makefile does:

```
$(CC) -c file.cpp -o output/file.o
```

Then QAC/QAC++ output will appear at:

```
/home/build/output/./new_output/file.err etc.
```

5.2.11 DEFINE_ON_OPTION

This allows specification of defines to be passed to analysis based on compiler options. For example if you specify -mt (multi-threaded) to a compiler it may implicitly define _REENTRANT. This define needs to be passed to QAC/QAC++ to make sure the code being analysed is the same as the code being compiled. DEFINE_ON_OPTION is a space separated list of compiler options and define pairs. The option and define are separated by a colon. For example:

```
DEFINE_ON_OPTION=mt:_REENTRANT
```

If the compile line has the -mt option, then -D _REENTRANT will be added to the analyser flags and passed to QAC/QAC++.

5.2.12 INCLUDE_OPTION

By default wrapper assumes that both -i and -I specify include paths. Some compilers do not support -i. Some compilers have options beginning with -i. Where options begin -i (-instances=global for example), wrapper extracts the text after -i as an include path, which can cause problems. The INCLUDE_OPTION sets a regular expression to accept as the option specifying an include path. The default is '[Ii]', allowing an upper or lower case i. If your compiler only supports -I, then:

```
INCLUDE_OPTION=I
```

will make wrapper accept only -I<path> as the way to specify a search directory.

Another example is:

```
INCLUDE_OPTION=(i|I|include)
```

This allows -I, -I and -include to act as the include option, but nothing else.

Note a space is allowed between option and path.

5.3 Environment Variables in Configuration Options

The values which you assign to any of the options can contain any number of environment variables, which must be of the form \$(ENV_VARIABLE). These should be set at the point at which the wrapper is executed; if they are not set, the wrapper will produce a message.

For example, if you set the environment variable PRQADIR to be the directory path in which QAC is installed, you would be free to put a line:

ANALYSER_BASE=\$(PRQADIR)

in your configuration file. Environment variables can also be used in configuration options passed by the `--set` option.

6. Change History

6.1 Version 2.4

6.1.1 Change Requests

CR12032 - Wrapper on Windows does not accept absolute include paths. This fix also correctly maps symbolically linked UNIX style directories to the correct Windows path

CR12033 - Empty -I option to wrapper causes next argument to be ignored. This fix has also been applied to the -D option.

CR12034 - Wrapper needs an option to delay between analysing files to prevent CPU overheating.

CR12036 - Wrapper fails to handle quoted -D options correctly. Macro definitions containing quotes are interpreted correctly by wrapper for both analysis and passing to the compile command.

6.2 Version 2.5

6.2.1 Change Requests

CR12501 – Allow analysis output files to go to a different directory to compilation output. Pass compiler option configurable defines to analysis. Allow specification of the compiler include directory option.

6.3 Version 2.6

6.3.1 Change Requests

CR12501 – Reworked to allow absolute prefix or relative suffix for OUTPUT_DIR.

CR12840 – Extend wrapper.pl to allow writing of .wqp (wrapper QAC/PP parameter) files and a log file of all .wqp files written.