



## **Compiler Wrapper for QAC/QAC++**

### **Version 3.1**

Author:	Jason Masters
Version:	3.1
Created Date:	15 March 2010
Last Modified:	17 September 2010
Number of Pages:	21

# Contents

<b>1</b>	<b>INTRODUCTION.....</b>	<b>3</b>
<b>2</b>	<b>INSTALLATION .....</b>	<b>3</b>
<b>3</b>	<b>OVERVIEW .....</b>	<b>3</b>
<b>4</b>	<b>USAGE .....</b>	<b>4</b>
4.1	COMPILER WRAPPING.....	4
4.2	PROJECT CREATION .....	5
4.3	ANALYSIS ON A REMOTE MACHINE.....	6
<b>5</b>	<b>CONFIGURATION FILE.....</b>	<b>7</b>
5.1	MANDATORY CONFIGURATION OPTIONS.....	8
5.2	OPTIONAL CONFIGURATION OPTIONS.....	8
5.3	ENVIRONMENT VARIABLES IN CONFIGURATION OPTIONS .....	16
<b>6</b>	<b>EXAMPLE WRAPPER CONFIGURATION FILE .....</b>	<b>16</b>
<b>7</b>	<b>NOTES ON USAGE OF WRAPPER ON WINDOWS .....</b>	<b>18</b>
<b>8</b>	<b>CHANGE HISTORY .....</b>	<b>18</b>
8.1	VERSION 2.4 .....	18
8.2	VERSION 2.5 .....	19
8.3	VERSION 2.6 .....	19
8.4	VERSION 2.7.1 .....	19
8.5	VERSION 2.8-BETA .....	19
8.6	VERSION 3.0 .....	20
8.7	VERSION 3.1 .....	20

## 1 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.

## 2 Installation

Wrapper does not have an installation program or script: simply untar or unzip the package. The Windows and UNIX packages contain some different files: the following table describes the files within the package.

File	Package	Purpose
CompilerWrapper.pdf	Both	Documentation
wrapper.pl	Both	Wrapper script
wrapper.exe	Windows	Wrapper executable
iso.c iso.cpp	Both	ISO standard test files that can be used to check Wrapper is working
wrapper_pc.cfg	Windows	Sample Wrapper Configuration file
wrapper_unix.cfg	UNIX	Sample Wrapper Configuration file
create_project_back_end.exe	Windows	Project creator program
create_project.options.personalities qaw.options qaw.options.properties	Windows	Contain information on all analysis options, how they can be used and which personality they apply to. Used by Project creator to assemble the analyser personality. These files must be saved in the same directory as create_project_back_end.exe

The Wrapper script and executables may be saved anywhere.

## 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. Wrapper can also be used to create a Windows GUI project from a set of compilation instructions. 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

### 4.1 Compiler Wrapping

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 5.

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.

## 4.2 Project Creation

Project Creation is currently only available on the Windows Platform.

Project Creation is a two stage process. The first stage is to run the Makefile (and possibly compilation) to gather the settings that would be used to analyse the files. The second stage is to process these results to create a GUI project file and appropriate Analyser Personality settings for all the files that are compiled.

### 4.2.1 Stage one – Running the Makefile

#### 4.2.1.1 Setting the Project Configuration mode

This is set with the ‘MODE’ configuration option in the Wrapper Configuration file.

There are two modes of Project Creation, one where compilation is not performed and one where compilation is performed.

#### 4.2.1.2 PROJECT\_CREATOR\_MODE

In this mode Wrapper does not launch the compiler after the analysis options have been recorded. Using this mode is quickest as no compilation or analysis is performed, the Makefile can be processed quickly.

#### 4.2.1.3 PROJECT\_CREATOR\_AND\_MAKE\_MODE

In this mode Wrapper will record the analysis options and then launch the compiler. This may be needed for some Makefiles to complete – for example they may manually copy the object files or build libraries as part of the build process which must exist before continuing the Makefile.

#### 4.2.1.4 WQP\_OUTPUT\_DIR

This must be the full path to an existing directory. On Windows this must be a Windows path, even when running under Cygwin. Wrapper will place the project creation output files in this directory. These are then used in stage two. Once stage two has been completed, the files are no longer required and can be deleted.

#### 4.2.1.5 Running Wrapper

Once the creator configuration option has been set Wrapper can be run in the normal way. Once it has completed there will be a file in the WQP\_OUTPUT\_DIR called master.wqp and a number of other .wqp files, based on the source file names.

## 4.2.2 Stage Two – Creating the Project

This is done by another program: `create_project_back_end.exe`. This reads the `.wqp` files created in stage one and extracts the settings to create an Analyser Personality for each folder. If files in the same directory have different settings, then a different Analyser Personality will be created and the files will be added in a sub folder in the Project File. All the files that would have been analysed by Wrapper are added to the Project File in appropriate folders. Note that the Project File does not use relative paths – this can be done by the GUI later.

The project creation needs to be supplied a Compiler Personality so that the files can be analysed from the GUI. A Message Personality can also be supplied.

If using a compliance module, this can be specified in the Wrapper Configuration file in the same way as when compiler wrapping. This results in the Analyser Personalities that are produced having the correct settings for the Compliance Module. Other Post and Secondary Analysis settings are **not** transferred to the GUI project.

The command for creating the project is:

```
create_project_back_end.exe -wqp <path for file listing wqp file locations> -  
prjf <path for writing project file> -p_s <path for message personality to be  
used> -p_c <path for compiler personality to be used>
```

The `-wqp` option is the full path the master.wqp file created in stage one (i.e. the `WQP_OUTPUT_DIR` setting, a path separation character and 'master.wqp').

The `-prjf` file is the full path to the name of the project file to create.

`-p_s` and `-p_c` specify the Message and Compiler personality that all the folders in the project will have.

For example:

```
create_project_back_end.exe -wqp c:/tmp/projx/master.wqp -prjf  
c:/work/projx/projx.prj -p_s "C:/Program Files/PRQA/QAC-7.2-  
R/personalities/critical.p_s" -p_c "C:/Program Files/PRQA/QAC-7.2-  
R/personalities/gcc_v3.0_qac4.5.p_c"
```

Once this has finished 'projx.prj' can be opened in the QAC GUI.

Note that currently during the creation of the Analyser Personalities no checking is made to see if files in different directories have the same settings – and could therefore use the same Analysis Personalities. In some cases it may be possible for all the files in a project (even though in different folders) to use just the one Analyser Personality.

## 4.3 Analysis on a Remote Machine

This feature is only available for Wrapper running on UNIX platforms.

This feature can be used when development is done on UNIX platform that cannot run QAC or QAC++ - for example AIX or Free BSD. The Makefile can be run on this development machine but the analysis commands are run via SSH on a remote machine that can run QAC and QAC++. For this to work correctly, both machines must have exactly the same view of the code – which is easily achieved using NFS.

In this section, the ‘development machine’ refers to the machine where make and Wrapper are being run (the one that cannot run QAC/QAC++). The machine that the analysis commands are run on is called the ‘remote machine’ or ‘analysis machine’.

For remote analysis using QAC it is strongly recommended to use QAC version 6.2 or later. If using previous versions the STOP\_ON\_FAIL setting will be ignored and analysis will proceed even if there are any hard errors.

### **4.3.1 SSH**

The remote commands are sent over SSH – but there are many different implementations of SSH. Wrapper has been tested with OpenSSH - additional configuration may be required for different implementations. Since Wrapper sends lots of remote commands (at least one per source file) it is best to set up SSH to work with out entering a password while running the analysis.

Additionally X11 forwarding should be set up to allow the Message Browser to be run on the remote machine but be viewed on the development machine.

### **4.3.2 Wrapper Configuration**

The configuration script must be visible on both the development and analysis machine. Some of the settings now relate to the development machine and some to the analysis machine: the following list indicates these:

ANALYSER\_BASE: must be set to the install location of QAC/QAC++ on the analysis machine

COMPILER\_SETTINGS\_FILE: this is set to the compiler personality file on the analysis machine: it does not have to be visible to the development machine. However, the settings within this file must apply to the compiler on the development machine. This will probably require copying of compiler header files to the analysis machine and configuring the personality to search these paths for compiler headers.

LOGFILE: this is a file on the development machine

The only other setting required is REMOTE\_HOST: this should be set to the IP address or hostname of the analysis machine. Additionally REMOTE\_VIEWER\_SCRIPT and REMOTE\_MESSAGE\_PERSONALITY can be set to make viewing the analysis results easier.

### **4.3.3 Operation**

Wrapper is run on the development machine in the normal way. The Makefile executes on the development machine, but when it comes to running the QAC or QAC++ analysis, the command is executed on the remote machine. This includes primary and any secondary analysis.

### **4.3.4 Troubleshooting**

There are two failure modes with running remote analysis: the analysis could fail or the remote command itself could fail. Wrapper will log both types of errors and for analysis errors the output of QAC or QAC++ will be logged.

## **5 Configuration File**

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

The wrapper has three mandatory settings and twenty 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, ERRDSP\_FLAGS and SECONDARY\_ANALYSIS.

If the wrapper is not supplied with the 3 mandatory options it requires to run, it will issue an error message about missing settings.

## 5.1 Mandatory Configuration Options

Wrapper must be provided with these settings whether it is being used for compiler wrapping or project creation.

### 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 CMA or errsum.

If omitted, no filelist file will be created.

### 5.2.2 POST\_ANALYSIS



This setting has been deprecated in favour of the `SECONDARY_ANALYSIS` option.

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.

Note that the use of `STDERR`, `TEXT` and `HTML` are subject to additional license control and may not be available to all users.

### 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+
```

All options in `ANALYSER_FLAGS` will appear before options from the `compile` command and the Compiler Personality. This `ANALYSER_FLAGS` can be used to specify a substitute header directory.

If the `COMPLIANCE_MODULE` option is used then the Analyser Personality for the Compliance Module `ANALYSER_FLAGS` will automatically be added as a `-via` file to `ANALYSER_FLAGS`.

### 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+
```

Note that the use errdsp is subject to additional license control and may not be available to all users.

### 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 '[Ili]', allowing an upper or lower case i. If you 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=(iIlinclude)

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

Note a space is allowed between option and path.

### 5.2.13 COMPLIANCE\_MODULE

The setting will cause Wrapper to analyse the code to one of the specified Programming Research Compliance Modules. This is done in two parts: the Analyser Personality of the Compliance Module is added as a `-via` file to the `ANALYSER_FLAGS` setting and the Secondary Analysis task for the Compliance Module is configured to run as if it were set in the `SECONDARY_ANALYSIS` option. Note that when the analysis results are viewed (or generated with `errdsp`) then the Compliance Module Message Personality must be used to give the correct message-to-rule mappings. When using the Message Browser this means adding a `-via` to the Message Personality, when using `errdsp` this means setting the Message Personality as a `-via` in `ERRDSP_FLAGS`.

The following settings are allowed:

Compliance Module	Product	Setting
MISRA C 1998	QAC	mcm
MISRA C:2004	QAC	m2cm
High Integrity C++	QAC++	hicppcm
MISRA C++	QAC++	mcppcm
JSF AV C++	QAC++	jcm

By default Wrapper expects to find the Compliance Module installed in the product installation directory (i.e. `$QACPATH` or `$QACPPPATH`). If the compliance module is installed in a different location then the `COMPLIANCE_MODULE_BASE` setting can be used to tell Wrapper where it is installed. Alternatively a Compliance Module can still be used but configured manually in the `ANALYSER_FLAGS` and `SECONDARY_ANALYSIS` options)

For example:

```
COMPLIANCE_MODULE=m2cm
```

This will analyse the code with the MISRA C:2004 analysis options.

This setting must not appear more than once.

### 5.2.14 COMPLIANCE\_MODULE\_BASE

If the selected Compliance Module set in `COMPLIANCE_MODULE` is not installed in the product installation directory then this setting is used to tell Wrapper where it is installed. It must be a full path to the top level directory of the Compliance Module installation.

For example:

```
COMPLIANCE_MODULE_BASE=/opt/cm/m2cm
```

The `/opt/cm/m2cm` directory contains the following subdirectories: `bin`, `doc`, `message`, `personalities`, `projects`.

Note that Compliance Modules are usually installed against specific versions of QAC or QAC++ and so are typically installed in the product installation directory, hence this setting should not normally be needed.

### 5.2.15 SECONDARY\_ANALYSIS

This setting supersedes the POST\_ANALYSIS setting. It specifies either a shortcut to PRQA process or the full path to a custom process to run after the initial analysis by QAC or QAC++. Options that the processes requires can be added to the setting: Wrapper will automatically add the output option and directory and source filename to the command line when launching the process. On Windows when specifying paths it is preferable to use forward slashes and if the path includes spaces then it must be enclosed in double quotes.

There are two built-in tasks available:

Setting	Task	Required Options
name_check	Name checker – to enforce a naming convention	-nrf <full path to name rule file>
baseline	Apply the Baseline – run the patch transform part of the Baseline process. A Baseline must have been previously generated	-sf <full path to baseline suppression> -sop <full path to working source root directory>

The product name (QAC or QACPP) is automatically added for these built-in tasks only.

Any task can be specified by using the full path to it, plus any additional parameters: Wrapper will always add the output path and filename.

This option may appear more than once to specify a sequence of analysis tasks. These will be performed in the order they appear in the configuration file – this is important as the Baseline process should be performed last.

For example:

```
SECONDARY_ANALYSIS=name_check -nrf C:/work/rules/name.nrf
SECONDARY_ANALYSIS=C:/progs/my_check.exe -full
SECONDARY_ANALYSIS=baseline -sf C:/work/baseline/projx/projx.sup -sop
c:/work/projx
```

This will run the Name Checker using the rules in C:/work/rules/name.nrf and then launch the custom analysis task like this:

```
C:/progs/my_check.exe -full -op <output path> <source file>
```

and then apply the baseline in C:/work/baseline/projx/projx.sup using the copy-source method.

### 5.2.16 FILENAME\_EXTENSIONS

By default Wrapper will only analyse files with a known source file extension to prevent analysis of object files (where the compiler is used to invoke the linker) or other non-compilable files (e.g. header files). The default extension list is:

"c", "C", "cc", "CC", "cC", "Cc", "cxx", "CXX", "Cxx", "cXX", "cxX", "cpp", "Cpp", "cPP", "cpP"

Note that Wrapper itself does not differentiate between C and C++ files.

This setting is used to set a new list of extensions that Wrapper considers to be source files. Note that the original default list is replaced rather than appended to.

The setting consists of a list of extensions (with or without the period) separated by semi-colons. Whitespace is permissible around the semi-colons. For example:

FILENAME\_EXTENSIONS=.c; .cc

Wrapper will only launch analysis for files that have a .c or .cc extension. Note that any case differences are handled as the underlying operating system would. In this example on Windows files with extensions .C and .CC would be analysed as well.

## 5.2.17 NON\_COMPILE\_OPTIONS

Compilers can be used for purposes other than creating an object file. For example, GCC can produce a dependency file for a source file or produce pre-processed source. Neither of these actions result in an object file. Some Makefiles use the compile variable to invoke these additional processes: normally this would also make Wrapper launch analysis – this could be incorrect due to incomplete options or cause duplicated analysis.

This setting allows specification of options that the compiler takes which do not produce an object file. The options must be separated by a space and must not include the character that specifies it as an option (either a '-' or a '/').

The default setting is 'MM'. Using this setting will override this default.

For example, using the GCC compiler:

NON\_COMPILE\_OPTIONS=M E

If a compile line invokes Wrapper with either '-M' or '-E' as one of the options then Wrapper would skip analysis and just run the compiler.

## 5.2.18 LOCK\_FILES

This setting can be used to make Wrapper lock any files that may be written when multiple instances of Wrapper are running concurrently. This can happen when make is given the -j2 option, or on sophisticated build systems using multiple processors/machines. In these cases the file list specified in FILELIST and the master wqp file in WQP\_OUTPUT\_DIR could be accessed by different processes at the same time, leading to corrupt or missing data.

Setting LOCK\_FILES=1 will cause Wrapper to exclusively lock these files while writing to them. Other processes have to wait until the lock is released before they can write to the file.

The default setting is 0 – i.e. no locking.

## 5.2.19 Windows Specific Settings

### 5.2.19.1 MODE

This option sets how Wrapper will operate – in either analysis or project creation mode. The mode may be set to one of the three following settings:

Setting	Action
MAKEFILE_INTEGRATION_MODE	Wrapper will launch analysis and compilation. This is the default mode.
PROJECT_CREATOR_MODE	Wrapper will create .wqp files only. No analysis or compilation will take place. If this mode is set WQP_OUTPUT_DIR must also be set.
PROJECT_CREATOR_AND_MAKE_MODE	Wrapper will create .wqp files and launch compilation. No analysis will take place. If this mode is set WQP_OUTPUT_DIR must also be set.

See section 4.2 for further information on using the project creator modes.

### 5.2.19.2 WQP\_OUTPUT\_DIR

This option sets the directory where Wrapper will write the .wqp files during the project creation stage. The full path must be used and the directory must exist.

## 5.2.20 Remote Operation (UNIX only ) settings

### 5.2.20.1 REMOTE\_HOST

This option enables running the analysis commands on a remote machine. Either the IP address or hostname of the analysis machine can be used.

### 5.2.20.2 REMOTE\_VIEWER\_SCRIPT

This setting will cause Wrapper to write a file that contains the command line to run the Message Browser on the remote machine in each directory where files are analysed. This removes the need for working out what the correct remote command should be. This option should be set to a the name of the file to be produced, for example:

```
REMOTE_VIEWER_SCRIPT=show_viewer.sh
```

If the file already exists it will be overwritten. If REMOTE\_HOST is not set, this option is ignored.

### 5.2.20.3 REMOTE\_MESSAGE\_PERSONALITY

This option can be set to the Message Personality to be used when opening the Message Browser from the remote script. If it is not set then no Message Personality is used, so all

messages will be shown. If set, it should contain the full path to the Message Personality on the analysis machine. For example

REMOTE\_MESSAGE\_PERSONALITY=/usr/local/qac-7.2R/personalities/critical.p\_s

If REMOTE\_VIEWER\_SCRIPT is not set, this option is ignored.

## 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 Example Wrapper Configuration file

Below is an annotated Wrapper Configuration File showing a possible setup for compiler wrapping using a Compliance Module with extra checks and using Baseline.

```
## Mandatory Settings
# Set compiler personality
COMPILER_SETTINGS_FILE=C:/Progra~1/PRQA/QACPP-2.5/personalities/VC++2005.p_c
# Set the product to use
ANALYSER=QACPP
# Set the product location: run qacppconf.bat to set the environment
ANALYSER_BASE=$(QACPPPATH)
## End Mandatory Settings
## Optional Settings
# Use the MISRA C++ Compliance Module
COMPLIANCE_MODULE=mcppm
# Also run the namin convention rules
SECONDARY_ANALYSIS=name_check -nrf C:/work/config/name.nrf
# And finally apply the baseline using a VCS
SECONDARY_ANALYSIS=baseline -sf C:/work/projx/projx.sup -sop c:/work/projx -sdt
"prqavcs -diff %BT %BF"
# Set thresholds for cyclomatic complexity and static path count...
ANALYSER_FLAGS=-thresh STCYC>10 -thresh STPTH>200
# ...and enforce the source code layout - this setting is cumulative
ANALYSER_FLAGS=-lf C:\work\config\extended.layout
# log extra information to file ...
DEBUG=1
# ... specified here
LOGFILE=C:/logs/wrapper.log
# Halt the analysis if we generate a level 9 message
STOP_ON_FAIL=1
# Record the files analysed in a file list
FILELIST=cpp_files
# Our Makefile uses G++ to generate preprocessed source
```



```
NON_COMPILE_OPTIONS=E
```

Assuming this file was saved as `c:/work/config/qacpp_wrap.cfg` and `wrapper.pl` is on the path, Wrapper can be run with the following command in a Cygwin shell:

```
$ make clean
$ make CXX="wrapper.pl -wcf c:/work/config/qacpp_wrap.cfg g++"
```

If the Makefile and analysis completes without failure the following commands can be used to create a master file list, run Cross Module Analysis and open the Message Browser with the MISRA C++ Message Personality (note that in this case this should be configured to display the metric threshold message and appropriate layout messages):

```
$ find . -name cpp_files | xargs cat >> all_cpp_files
$ echo "-cmf c:/work/projx/projx" >> all_cpp_files
$ pal QACPP -list all_cpp_files
$ viewer QACPP -via "C:/Program Files/PRQA/QACPP-2.5/mcpp/personalities/mcpp.p_s" -list all_cpp_files
```

Note that usually the `-cmf` file option would be directed to the output folder. The master file list can be used as input for other PRQA tools, for example reports, the generation of a baseline or for uploading to MIS.

Below is a sample Configuration file for creating a GUI project for the same make project.

```
## Mandatory Settings
# Set compiler personality
COMPILER_SETTINGS_FILE=C:/Progra~1/PRQA/QACPP-2.5/personalities/VC++2005.p_c
# Set the product to use
ANALYSER=QACPP
# Set the product location: run qacppconf.bat to set the environment
ANALYSER_BASE=$(QACPPPATH)
## End Mandatory Settings
## Optional Settings
# Use the MISRA C++ Compliance Module
COMPLIANCE_MODULE=mcppm
# Set thresholds for cyclomatic complexity and static path count...
ANALYSER_FLAGS=-thresh STCYC>10 -thresh STPTH>200
# ...and enforce the source code layout - this setting is cumulative
ANALYSER_FLAGS=-lf C:\work\config\extended.layout
# Our Makefile uses G++ to generate preprocessed source
NON_COMPILE_OPTIONS=E
# Set the mode to just extract the settings
MODE=PROJECT_CREATOR_MODE
# Put all the settings files here
WQP_OUTPUT_DIR=C:/config/proj_create
```

Note that running the Name Checker and Baseline must now be configured from the GUI.

Wrapper can be invoked the same way as for the compiler wrapping example:

```
$ make clean
$ make CXX="wrapper.pl -wcf c:/work/config/qacpp_wrap.cfg g++"
```

This will complete more quickly as there will be no analysis or compilation. The project can then be created with the following command:

```
$ create_project_back_end.pl -wqp c:/config/proj_create/master.wqp -prjf  
c:/work/projx/projx.prj -p_s "C:/Program Files/PRQA/QACPP-  
2.5/mcpp/personalities/mcpp.p_s" -p_c "C:/Program Files/PRQA/QACPP-  
2.5/personalities/VC++2005.p_c"
```

When the resulting project file `c:/work/projx/projx.prj` is opened in the QAC++ GUI al; the files compiled in the normal build process will be in the folder structure. The project can be analysed as normal (see the QAC++ User Guide for information on configuring the Name Checker and Baselineing).

## 7 Notes on Usage of Wrapper on Windows

In a pure Windows environment `wrapper.exe` should be used. Where there is a UNIX like environment under Windows (e.g. Cygwin) then either the PERL script `wrapper.pl` or the executable can be used; the PERL script is preferable.

When specifying Window paths, forward slashes are preferable to backslashes (as each backslash needs to be doubled up to escape it).

Where spaces are used in path names, the whole path name must be enclosed in double quotes. Using the short path name can sometimes be easier (though be aware that short path names can still contain spaces).

Windows paths must always be used when configuring options: even if you are running `wrapper.pl` in Cygwin, QAC and QAC++ are Windows programs and cannot understand UNIX paths. Additionally QAC and QAC++ cannot follow symbolic links so paths must be reachable under Windows and executables like the compiler must be able to be run from Windows.

## 8 Change History

### 8.1 Version 2.4

#### 8.1.1 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.

## **8.2 Version 2.5**

### **8.2.1 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.

## **8.3 Version 2.6**

### **8.3.1 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.

## **8.4 Version 2.7.1**

### **8.4.1 Change Requests**

CR 13338 – Quoted include files passed to Wrapper from an executable are now treated correctly. Files using backslash directory separators are treated correctly on Windows when using Cygwin.

CR13404 – Wrapper fails to handle DOS paths correctly in @inline files.

## **8.5 Version 2.8-beta**

### **8.5.1 Change Requests**

CR 13483 Insert the ANALYSER\_FLAGS before all other command line options (from the Makefile and Compiler Personality). This means that a substitute header directory can be specified in the Wrapper Configuration File.

## 8.6 Version 3.0

### 8.6.1 Change Requests

CR 13479 POST\_ANALYSIS programs fail when supplied with options when running Wrapper as a Native PERL script. Programs are now launched correctly from Wrapper.

CR 12134 Convert file names to Windows paths on Windows. File names in the file list will now appear as Windows paths when running Wrapper (either the script or the executable) on Windows to allow the Message Browser and other PRQA tools to read them correctly.

CR 13486 Extend to handle multiple source files per compilation and CR 13223 Handle more than one source file at a time. Wrapper will now process (i.e. analyse or include in project creation) all the appropriate files on a compile command.

CR 13224 Write .wqp files in location(s) away from source files. When in project creation mode Wrapper will put all the temporary .wqp files into a specified directory rather than the output directory.

CR 13227 Start Multiple Secondary Analysis Tasks in Wrapper. Wrapper can now run more than one Secondary Analysis task. Secondary Analysis tasks for PRQA Compliance Modules and processes (i.e. Name Checker and Baseline) can be specified easily.

CR 12192 Pass product to secondary analysis task. This is applicable only to PRQA Compliance Modules, the Name Checker and Baseline. Custom Secondary Analysis tasks (that require it) must have this specified in the SECONDARY\_ANALYSIS option.

CR 13296 New feature: in PROJECT\_CREATOR\_MODE, have option to run compilation. PROJECT\_CREATOR\_AND\_MAKE\_MODE will launch compilation after processing the compiler arguments. This is because some Makefiles need compilation to succeed before progressing. Note that analysis is not performed.

CR 13309 Require File::Spec::Win32 qualification needed.

CR 13485 Wrapper should be able to analyse files of arbitrary extension. Analysable files extensions can be specified in a list

CR 13487 Wrapper does not identify gcc's-M as a non-compile option. Options that do not result in the compiler producing an object file can be specified in a list.

CR 13551 Remove Wrappers dependency on errdsp for QAC. For QAC versions 6.2 and later the call to errdsp to verify the analysis success has been removed as QAC correctly reports this in it's exit code. For previous versions, errdsp is still called. The call to errdsp to produce the 'hard error' list on analysis failure has been removed.

CR 13226 Multiple concurrent analysis fails. This was due to concurrent access to the file list. Optional file locking has been added for this file and the master wqp file in Project Creation mode.

## 8.7 Version 3.1

### 8.7.1 Change Requests

CR 13666 Add ability to run analysis on a remote host to Wrapper. This is for use on UNIX development systems where there is no port of QAC or QAC++, for example AIX. The make

command is run on the development system but the analysis is done on a remote UNIX system that can run QAC and QAC++ (typically Linux).

CR 13689 Wrapper assumes that the qac executable is in the PATH when trying to get version number. Wrapper now uses the full path to the executable.