
QAW User Guide

Overview

The QAW program is used as a command line interface to PRQA tools except for the Windows and Unix GUIs, and Integrations. Note that simple command line use of QAC and QACPP only performs the primary (major) phase of analysis, Secondary-Analysis is not invoked, whereas QAW will invoke Secondary-Analysis if it is specified. Other capabilities of QAW include analysing project files directly and the ability to handle many files and projects on the same command line.

QAW can be used to analyse with QAC or QACPP, the choice is controlled by the 'qac' | 'qacpp' keyword following the qaw program name. Throughout this guide 'qac' is used, when using QACPP substitute 'qacpp' throughout.

For example to analyse a QAC project and view the results, use:

```
qaw qac c_examples.prj -view
```

To analyse a QACPP project and display the results as annotated source on standard output, use:

```
qaw qacpp cpp_examples.prj -disp
```

Summary of QAW Options

Option	Description
-afe	Specify Analysis Filename Extensions
-alias	Specify 'environment variable' local to QAW options
-canal	Invoke Configured project wide Analysis
-cargs	Specify start of compiler arguments section of command line, where compiler is specified - used within Makefile Integrations
-crep	Invoke Configured Report
-dafe	Display Analysis Filename Errors
-disp	Request display of annotated source on analysing each file
-ehtml	Ensure .html output generated
-etxt	Ensure .txt output generated
-exec	Specify Secondary-Analysis program
-fdisp	Request annotated source output to file on analysis of each file
-glob	Specify project wide analysis program and display program
-glog	Request logging of analysis progress to GUI window (tbd)
-help	Request information on options
-logerrs	Specify output destinations for error output
-logmsgs	Specify output destinations for message output
-maseq	Specify a sequence of project wide analysis programs
-mode	Control which files are to be analysed
-pdsp	Request warning listing display on analysing each set of files
-pdspd	Request warning listing display on analysing each set of files. Display with default browser.
-plog	Request logging of analysis progress to standard output
-sapf	Request analysis parameters report (Show Analysis Parameters File)
-saseq	Specify a sequence of secondary analysis programs
-sat	Set Argument Translation for use with -cargs
-sfba	Suppress File Based Analysis - Suppress primary and secondary analysis, perform project wide analysis only
-stoponfail	Terminate on encountering an analysis error
-targs	Specify start of compiler arguments section of command line, where no compiler is specified. Used within Makefile Integrations
-trace	Request trace of QAW and any Secondary-Analysis programs that accept -trace parameter
-version	Request version of QAW
-view	Request message browsing on analysing each set of files

QAW Features

QAW is intended to act as a wrapper for QAC and as such will handle all the options normally passed to QAC. In fact it does pass many options to QAC without filtering them at all. The extra features introduced by QAW are:

- To enable command line usage of QAC to invoke any number of Secondary-Analysis programs.
- To enable multiple files to be passed for analysis per invocation - either by simply specifying a list of file names on the command line, possibly with wildcards, or by using the `-list` option, or both.
- To enable the user to directly analyse any number of entire projects or specific folders within projects simply by supplying project file(s) on the command line, also possibly with wildcards for file names.
- To enable embedding of environment variables in `.prj` files and within personality settings. QAW allows definition of local aliases within these files. QAW also supports the relative path and environment variable features of the Windows GUI.
- To allow project wide analysis to be invoked on completion of analysing all or part of a project or an explicitly supplied set of files, or both, and to allow viewing of the results.
- To provide the user with audit information detailing all settings used in the analysis of each file.
- To provide a simple mechanism for launching configured reports.
- To provide a simple mechanism for launching configured project-wide analysis.
- To provide a mechanism for launching analysis and compilation as needed by Makefile Integrations. This includes specification of how compiler arguments are translated to QAC/PP options.
- To display annotated source (`errdsp`) on completion of analysis of each file.
- To display the Warning Listing (`prjdsp`) on completion of analysis of a set of files.
- To invoke the Message Browser (`viewer`) on completion of analysing a set of files.
- To provide improved help on options.
- To allow the user to control (`-mode` option) the analysis of
 - all files
 - only those files for which existing analysis results are out of date
 - or to perform no Primary or Secondary-Analysis of the files whatsoever and simply view the current results.

Passing Files on the Command-Line

The number of files that can be passed to QAW on the command line is only limited by the maximum length of a command line in the shell of the operating system being used. Limitations on the maximum length of the command line allowed by the shell can be overcome by specifying any number of `-list` options on the command line. For example to analyse the following files (and not display analysis results) `file1.c`, `file2.x`, `file3.c`, all files listed in `lof1` and `lof2`, use:

```
qaw qac -via..\person\mcm.p_s file1.c file2.x -list lof1 -list lof2 file3.c
```

Syntax of the `-list` option is:

```
-list <full file name>
```

Wildcards may also be used for filenames e.g.

```
qaw qac *.prj *.c
```

QAW is sensitive to the extension of file names supplied, whether supplied on the command line, using `-list`, or within project files. By default the following extensions are allowed, any other extension (except `.prj`) will result in the warning message "Ignoring unrecognised file ..."

```
".c.C.cc.CC.cpp.cxx.c++.h.H.hpp.hxx.h++.i.I.ipp.ixx.i++"
```

If you wish to use files with other extensions you should use the `-afe` option, e.g.

```
qaw qac -op . -afe "c75.C*" a.c75 b.C c.Cxx99
```

- will allow analysis of files `a.c75`, `b.C` and `c.Cxx99`.

Analysing Projects

To analyse an entire project simply include the project file name on the command line in the same way as specifying a simple file for analysis. e.g.

```
qaw qac mcm_examples.prj
```

QAW treats files with the `.prj` extension as project files.

To analyse an individual folder within a project folder use the following:

```
qaw qac "mcm_examples.prj#Folder Name"
```

or, to analyse 2 separate folders and display the results in the message browser:

```
qaw qac mcm_examples.prj#folder1 mcm_examples.prj#folder2 -view
```

or, to analyse an individual file within a project folder use the following:

```
qaw qac "mcm_examples.prj#Folder Name:File Name" -view
```

Note the ':' syntax which introduces the 'filename within folder', i.e. 'File Name' within folder 'Folder Name'.

Note that the full syntax for 'filename within folder' specification is:

```
qaw qac <project file>#<folder>:<filespec>
```

Here <filespec> is normally a simple (base) filename, however, if this is not sufficiently accurate to specify the required file, <filespec> can be given as an absolute (full) path.

To analyse a number of files, within a project folder, using a wildcard specification use, for example, the following:

```
qaw qac mcm_examples.prj#folder_name:rule_12*
```

or, to analyse a set of files using the personality settings of a project folder use:

```
qaw qac <project file>#<folder>#<filespec>
```

e.g.

```
qaw qac mcm*.prj#folder_1#*.c
```

Note that here filespec specifies the list of file(s) to be analysed using the project personalities. Here the file(s) specified by filespec are relative to the current working directory (i.e. pwd for Unix etc), and not relative to the folder directory, which is the case when using the ':' syntax. That is, the filespec may be any files and can only specify files within the project if filespec explicitly gives the paths of file(s) within the project.

For example the following commands analyse the same files:

```
qaw qac $MCM\mcm*.prj#folder_1#$MCM\src\*.c
qaw qac $MCM\mcm*.prj#folder_1:*.c
```

When **analysing** projects QAW ignores all command line settings except the following:

- -afe
- -canal
- -crep
- -dafe
- -disp
- -ehhtml
- -etxt
- -fdisp
- -glog
- -glog
- -logerrs
- -logmsgs
- -maseq
- -mode
- -pdsp
- -plog
- -sapf
- -sfba
- -stoponfail
- -trace
- -view

Project analysis results reflect the settings within the project itself unless these are overridden on the command line. For example, -exec specified on the command line affects individual and -list specified files, as well as any projects listed on the same command line.

When analysing and/or displaying project analysis output, all command line settings including the above 'QAW' settings plus all display settings within the project itself, are used. For example:

```
qaw qac $MCM\mcm*.prj -html+ -hdr- -disp
```

Embedding Variables in Personality and Project Files

Unix style environment variable embedding may be used on Windows and Unix within project files, files that are included with -via, file names included by -list, and simple command line arguments. For example consider the following QAC Windows project file:

```

VersionTag45
StartProjectMarker
FolderName=global_checks
SourcePath=$QACPATH\mcm\projects\global_checks\src\
OutputPath=$QACPATH\mcm\projects\global_checks\output\
SubsPers=$QACPATH\mcm\projects\global_checks\person\mcm.p_s
AnalPers=$QACPATH\mcm\projects\global_checks\person\mcm.p_a
CompPers=$QACPATH\mcm\projects\global_checks\person\mcm.p_c
${QACPATH}\mcm\projects\global_checks\src\file1.c
${QACPATH}\mcm\projects\global_checks\src\file2.c
EndContainedFilesMarker

```

When this project file is evaluated \$QACPATH is replaced with the current environment setting of environment variable QACPATH. Note that \${QACPATH} is also acceptable.

Environment variables can be used anywhere in the QAW input, i.e. within project files, within personality files, within list of files files and on the command line itself.

A further feature of QAW is locally declared variables, this is done using the -alias option to define a variable.

Syntax is:

```
-alias <var name>=<value>
```

Note that -alias always takes exactly one parameter which should be quoted if it contains any space characters. Quote characters can be embedded by escaping with \ (backslash).

A variable declared with -alias is dereferenced in the same way as an environment variable. Hence a project file can have its own local values e.g. (n.b. this is a QAC Windows project file):

```

VersionTag45
-alias PRJ_HOME=$QACPATH\mcm\projects\global_checks
-alias PRJ_PERSON=${PRJ_HOME}\person
StartProjectMarker
FolderName=global_checks
SourcePath=$PRJ_HOME\src\
OutputPath=$PRJ_HOME\output\
SubsPers=$PRJ_PERSON\mcm.p_s
AnalPers=$PRJ_PERSON\mcm.p_a
CompPers=$PRJ_PERSON\mcm.p_c
$PRJ_HOME\src\file1.c
$PRJ_HOME\src\file2.c
EndContainedFilesMarker

```

Windows GUI Relative Paths and Environment Variables

As well as allowing the Unix style environment variable embedding described above, QAW supports the Windows GUI method of defining Relative Paths and Environment Variables. This is on Windows only (on Unix Windows style environment variables e.g. %QACBIN% are not allowed.)

On Windows, QAW will handle all projects created in the Windows GUI. The follows rules apply:

- %EnvironmentVariable% is only allowed once per option, or -via file line, or project file line.
- %EnvironmentVariable% may only be embedded in options that have a path parameter. e.g. it is not possible to use %EnvironmentVariable% in a -format string.
- Relative Path prefixing is not applied to options on the QAW command line.
- It is an error if a %EnvironmentVariable% is not defined in the current environment.
- It is an error if a path is not absolute after an embedded %EnvironmentVariable% is expanded.
- It is an error if a path is not a real path after an embedded %EnvironmentVariable% is expanded.
- It is an error if %SOURCEPATH% is found in a personality file.
- It is an error if %SOURCEPATH% replacement does not result in a real path.
- It is not possible to write a parameter which mixes a wildcard with an environment variable or relative path.

Secondary-Analysis

QAW allows the standard QAC message personality based method of specifying Secondary-Analysis via -rem statements. Extra flexibility is provided by allowing any number of message personality files to be specified - each message personality file is allowed to provide at most one Secondary-Analysis specification. An alternative method for specifying Secondary-Analysis, which has no limit on where it is used, or the number of times it is used, is the [-exec option](#) , for example:

```
-exec "C:\Program Files\prqa\QAC-5.0-P3\mcm\bin\qacuser_mcm.exe###F"
```

The [-saseq option](#) also allows specification of Secondary-Analysis, the difference being that -saseq specifies a Secondary-Analyser **sequence**, for example:

```
-saseq "qacuser_mcm %F+#C:\QAC\ Files\misrauser_mcm %F+"
```

Note that `-saseq` merges the specification of the analyser program and its parameters. It is important to note that in the `-exec` program specification, spaces in the program path are written without any formatting. In `-saseq` options spaces in the program path must be escaped with backslash, for example:

```
"C:\Program Files\X"
```

is escaped as:

```
"C:\Program\ Files\X"
```

Note that `-saseq` is cumulative as is `-exec`, `-rem`, `-glob` and `-maseq`. Where these options occur they are always added to the currently specified Secondary-Analyser or Project-Wide_Analyser settings.

Project Wide Analysis

QAW has the capability to run any number of project wide analysis programs which it does on completion of analysis of the supplied files or on the completion of analysis of each project - provided (for `-mode all` analysis only) there are no errors during analysis of the project or fileset. To do this the [-glob option](#) or the [-maseq option](#) is used (`-maseq` allows specification of a project wide analyser **sequence**), for example:

```
-glob "C:\Program Files\prqa\QAC-5.0\mcm\mcmprj#-list %L -html %R+ %P+#Iexplore"
```

```
-maseq "mcmprj -list %L -html %R+ %P+#pal %Q %L+ %P+"
```

Note that `-maseq` merges the specification of the analyser program and its parameters. It is important to note that in the `-glob` program specification, spaces in the program path are written without any formatting. In `-maseq` options spaces in the program path must be escaped with backslash, for example:

```
"C:\Program Files\X"
```

is escaped as:

```
"C:\Program\ Files\X"
```

Note that [-canal option](#) and [-crep option](#) provide shortcut methods of invoking project wide analysis.

Analysis with Dataflow

To analyse with the dataflow capability introduced in QAC-8.0, simply append the `-ed` option to the command line. e.g.

```
qaw qac -op . my project.prj file.c -ed+ -view
```

To ensure that dataflow is switched off do

```
qaw qac -op . my project.prj file.c -ed- -view
```

Analysis control

QAW allows the user to control analysis of the supplied files or projects. This is done with the `-mode` and `-sfba` options. Syntax of `-mode` is:

```
-mode analysis_mode
```

Where `analysis_mode` can take one of the following 3 values

all	This is the default. In this mode all files are analysed.
dep or depend	Analyse only those files for which the current results are or may be out of date.
none	Skip analysis of files and simply execute display options.

Note that if project wide analysis is specified (`-canal`, `-crep`, `-glob` or `-maseq` option), then project wide analysis and display is always performed irrespective of the setting of `-mode`. QAC analysis of individual files is governed by `-mode`.

If the `"-mode depend"` option is supplied QAW only analyses files for which the [analysis](#) options have changed (e.g. different `-D` settings, `-I` settings, `-SI`, `-SD`, `-EN` etc) or if any file contributing to the translation unit is newer than the existing `.err` file, or if any of the `.err`, `.opt` and `.met` files are missing, or if the Primary Analyser or any of the Secondary-Analyser programs have a different version number. Should other output files be required the analysis must be invoked with the `"-mode all"` option in order to guarantee the output files are up to date. In other words `"-mode depend"` only guarantees `.err`, `.met` and `.opt` files are up to date.

Note that when using QAC-8.0 or later, the `-O` and `-N` options are treated as [analysis](#) options and hence will trigger 're-analysis' if -

mode depend is used and the only change to the QAW settings is a modification of the -O and/or -N settings.

QAW uses the same algorithm for looking up header files as the parsers, this is based on -I, -SI specifications.

The -sfba option suppresses primary and secondary analysis, it is intended for project wide analysis usage in QAW. For example, a number of files may be analysed with given personality settings, normally the project wide analysis stage would be performed with the same settings. By using a command like:

```
qaw qac mcm.prj -maseq "pal %Q %P+ %L+" -sfba -view
```

- the project wide analysis is performed without repeating the primary and secondary analysis of mcm.prj.

The project wide analysis could also be performed with different settings, e.g.

```
qaw qac -list lof -via person -view
qaw qac -list lof -via other -maseq "pal %Q %P+ %L+" -sfba -view
```

- will result in primary and secondary analysis being performed using 'person' and the project wide analysis performed using 'other', without repeating primary and secondary analysis of lof.

Compiler Wrapper Usage

QAW can be used to perform both analysis and compilation functions, i.e. it can be used as a compiler wrapper for applications like Makefile Integrations. When QAC is used as a compiler wrapper analysis is performed by QAC and compilation is performed by a conventional compiler. In this mode the command line passed to the QAW wrapper is forwarded to the compiler in order to invoke compilation.

QAW supersedes the PRL wrapper.pl utility.

QAW is versatile enough to be used in both IDE integrations and makefile integrations. Considering use of QAW as a wrapper in a simple makefile, the makefile used to compile a C "hello world" program might be:

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

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

On Unix systems QAW can be used as a wrapper as:

```
make "CC=qaw qac -sat I=-I -sat D=-D -cargs gcc"
```

This CC definition causes the makefile to execute with a two step process of (QAC) analysis followed by compilation.

A typical QAW wrapper usage might be:

```
qaw qac -via gcc3.2.3_compiler_person.p_c -cargs gcc -c -DWIN alg.c main.c
```

QAW invokes QAC with the QAW options specified on the left of the -cargs marker token, translated options after the -cargs marker, and files listed after -cargs. QAW then invokes the compiler specified by the token after -cargs (gcc in this example) with the parameters that follow the compiler token. Note that QAW options (including all QAC options) can be placed before the -cargs marker in the same way as when a simple QAW analysis is performed.

For the above example, QAW execution steps are equivalent to:

QAC analysis step:

```
qaw qac -via gcc3.2.3_compiler_person.p_c -DWIN alg.c main.c
```

Where gcc3.2.3_compiler_person.p_c contains:

```
-sat D=-D
```

Compilation step:

```
gcc -c -DWIN alg.c main.c
```

Secondary-analysis is invoked in the normal way. i.e. by providing a .p_s file with -rem statements detailing the secondary-analysis, or by providing QAW -exec or -saseq commands.

-stoponfail can be used to terminate QAW execution on a QAC analysis error.

Output of Annotated Source

On completion of analysis of each file, both for individual files and files within projects, QAW will call `errdsp` to output the annotated source for that file to Standard Output if the `-disp` option is specified on the command line.

Output of Warning Listing

On completion of analysis of the supplied files or on the completion of analysis of each project, QAW can invoke `prjdsp` for the file set. To do this the `-pdsp` / `-pdspd` option is used, for example:

```
-pdsp "C:\Program Files\Internet Explorer\IExplore"
```

Here the syntax is:

```
-pdsp <renderer program>
```

Where *renderer program* is the program specification ([path]name) necessary to invoke an html browser like Netscape, IExplore, Chrome etc .

Alternatively you can use the `-pdspd` command which uses the Environment Variable `PRQA_DEFAULT_BROWSER` as the renderer. Hence syntax is simply:

```
-pdspd
```

Output Using the Message Browser

On completion of analysis of the supplied files or on the completion of analysis of each project, QAW can invoke the Message Browser (viewer) for the file set. To do this the `-view` option is used. For example:

```
qaw qac $MCM\mcm*.prj -view
```

Help

The `-help` option has 2 modes.

1. If the user invokes QAW with `-help` and no other options, then help for all options is displayed.
2. If the user supplies any additional options, help is displayed for those options only. When looking up the options to give help for, QAW returns help on all options that start with the supplied option name.

For example:

```
qaw qac -help -d
```

Results in:

```
-D ident -D ident=value  
Define macros for QA C in the same way as you would  
for your compiler.
```

```
-DISP  
Display annotated source on analysis.
```

Analysis Parameters Reporting

QAW always generates a `.opt` file alongside `.err` and `.met` on completion of analysis. This file contains all parser parameter settings used for the analysis, i.e. the contents of `settings.via` as passed to QAC.

The purpose of `.opt` files is to capture analysis parameters in order to be able to retrieve those parameters allowing the user to justify analysis output with respect to the parameters used to perform the analysis. `.opt` files are also used by the `-mode` option to determine whether significant analysis options have been changed when deciding if the file should be reanalysed. `.opt` files can also be used directly as QAW input parameter files should the user want to repeat analysis with previous analysis parameters.

The QAW option `-sapf` (Show Analysis Parameters File) outputs the `.opt` file to standard output immediately after outputting the annotated source if `-disp` is specified, or otherwise immediately after completing analysis.

QAW Usage

This section explains some of the detailed aspects of using QAW and gives examples of usage

Use of Quoted Parameters

Any parameter to a QAW option that includes spaces must be quoted with double quotes. Note that the *parameters* section of `-exec` and `-glob` and the entire parameter of `-maseq` and `-saseq` are an exception to this. For example:

```
-alias "ANSI=C:\Program Files\prqa\QAC-5.0\include\ansi"
```

or

```
-alias"ANSI=C:\Program Files\prqa\QAC-5.0\include\ansi"
```

or

```
"-aliasANSI=C:\Program Files\prqa\QAC-5.0\include\ansi"
```

Other uses of quotes are possible but not recommended. All spaces within quotes are significant, hence the following will probably be an error as the user would not intend the path to be prefixed with spaces:

```
"-i C:\Program Files\somedir"
```

Spaces and quotes are treated in exactly the same way whether the parameter is within command line options, projects files, `-list` files or files included by `-via`.

For options: `-exec`, `-glob`, `-maseq` and `-saseq`; significant embedded space characters may be escaped (Unix style) with `\` (backslash). For example:

```
-exec "\user dir\qacuser## -xxx \user\ dir\other\xxx -file %R"
```

```
-saseq "C:\dir\ name\prog %F+#C:\x\ y\w\ z\prog %F+"
```

Note that file name entries in a list of files specified by the `-list` option should be quoted if the full file name has spaces.

Analysis File Sets

QAW analyses files in sets. Each project or folder of a project that is submitted is analysed as one set. All other (non-project) files are analysed in one set. That is, all file names specified on the command line, and all files specified by the `-list` option, are analysed as one set with all command line options applied. Files specified by a *project* based analysis are analysed as a distinct set (for each project spec on the command line). So files are specified in one of the following ways,

1. outside of projects
2. within projects
3. using hybrid method

Files specified outside of projects are all the files specified on the command line that are not project files. This set is always analysed after all projects have been analysed, i.e. last of all.

The hybrid method of specifying files uses the personalities of the supplied project file (a folder name is optional) to analyse specific files. Files to be analysed are specified in one of two ways:

1. Relative to the source directory of the specified project folder, or
2. Relative to the current working directory.

To specify files relative to a project folder, use syntax:

```
qaw qac <project file>#<folder>:<filespec>
```

To specify files relative to the current working directory use syntax:

```
qaw qac <project file>#<folder>#<filespec>
```

All *options* on the command line are global to all analyses. Personality and other details held within a project are only applied to that project when it is analysed, along with all command line settings.

Specification of OUTPUTPATH

The `-outputpath` (`-op`) option has unique behaviour among PRQA options. For analysis of files on the command line, `-op` must be specified as follows:

`-op` should be specified before the first file on the command line, if this is not the case a warning is issued and the `QACOUTPUTPATH` environment variable is used.

Trailing -op specification(s) are not allowed, this results in termination.

Examples of QAW Usage

QAW requires that the QAC environment variables are set before it is run. To do this QAC\bin\qacconf.bat may be used on Windows. For Unix .profile etc should be sourced.

The simplest way to use QAW is as a direct replacement for QAC, e.g.

```
qaw qac file.c
```

To avoid the QACOUTPATH warning, specify the outputpath setting as the first option on the command line. e.g.

```
qaw qac -op . file.c
```

Several files can be specified, e.g.

```
qaw qac -op . file1.c file2.c file3.c
```

or

```
qaw qac -op . *.c
```

A list of files can also be specified, e.g.

```
qaw qac -op . -list list_of_files
```

Projects can be specified as follows. Note that project files define outputpaths hence the -op option is not necessary here (and will be ignored even if it is set), e.g.

```
qaw qac diff.prj example.prj
```

Wildcards can be used, e.g.

```
qaw qac *.prj
```

Headings are inserted into -plog output if a folder/filespec is given in the project specification. This is also true if the folder/filespec names are null, for example the following causes headings to be inserted into the output log,

```
qaw qac *.prj#: -plog
```

A folder within a project can be specified, e.g.

```
qaw qac diff.prj#folder_name
```

or

```
qaw qac d*.prj#folder_name
```

A combination can be analysed, e.g.

```
qaw qac file1.c -list list_of_files diff.prj file2.c
```

All parser options may also be specified on the command line in the same way as when using the parser as a standalone program. To analyse a file with a personality file use the standard parser options e.g.

```
qaw qac file1.c -via..\person\mcm.p_s -via ..\person\mcm.p_c
```

or

```
qaw qac file1.c -via..\person\mcm.*
```

or use the equivalent form of personality specification via a project e.g.

```
qaw qac ..\projects\mcm_examples\mcm_examples.prj##file1.c
```

All the above examples invoke the parser which will write .err, .met and possibly other files. For each file analysed QAW writes an <output path>/<filename>.opt file which contains the options used for the analysis and the time of the analysis for the file being analysed.

Further options may be specified on the command line to control Secondary-Analysis, Global Analysis and viewing of the results of analysis.

To analyse files on the command line with Secondary-Analysis applied use the -exec option, e.g.

```
qaw qac file1.c -list files.lst \  
-exec "C:\Program Files\prqa\qac-5.0\mcm\bin\qacuser_mcm##%F"
```

Or a Secondary-Analyser sequence as specified with -saseq, e.g.

```
qaw qac file1.c -list files.lst \  
-saseq "qacuser_mcm %F#misrauser %F"
```

(Remember that % is interpreted by the DOS shell. To pass % in a batch file it must be doubled up.)

-exec and -saseq can be used in any personality file (including project personality files) or on the command line to specify Secondary-Analysis for all files. Analysis of projects is governed by the personality files referenced in the project file (.prj file), these settings are further qualified by command line settings.

Each Secondary-Analysis program specified is run immediately after successful analysis of each file by the parser. Global Analysis is invoked on completion of analysis of an entire file set. Any number of Secondary-Analysis and Global Analysis programs can be run.

Primary and Secondary analysis can be suppressed with the -sfba or -mode none options. -sfba will just suppress Primary and Secondary analysis, but will allow global analysis. -mode none will suppress all analysis.

Project wide analysis can be performed with different settings to those used for Primary and Secondary analysis, e.g.

```
qaw qac -list lof -via person -view  
qaw qac -list lof -via other -maseq "pal %Q %P+ %L+" -sfba -view
```

- will result in primary and secondary analysis being performed using 'person' and the project wide analysis performed using 'other', without repeating primary and secondary analysis of lof.

As well as the -exec and -saseq options, the QAC message personality mechanism for specifying Secondary-Analysis can be employed. That is, a mixture can be used. When using the message personality file to specify Secondary-Analysis only one Secondary-Analysis program can be specified per message personality file. However any number of message personalities can be included in the command set. Duplicate specification of a Secondary-Analysis program will result in duplicated Secondary-Analysis runs.

To analyse the same files with 2 Secondary-Analysis programs, and make use of the alias facility and the QACPATH environment variable, use

```
qaw qac file1.c -list files.lst -alias"BIN=$QACPATH\mcm\bin" \  
-exec $BIN\qacuser_mcm##%F -exec $BIN\misrauser##%F
```

Or use -saseq, e.g.

```
qaw qac file1.c -list files.lst -alias"BIN=$QACPATH\mcm\bin" \  
-saseq "$BIN\qacuser_mcm %F#$BIN\misrauser %F"
```

The \ (backslash) on the end of the first line is used here as a notation for continuation line.

To apply Global Analysis as well use

```
qaw qac file1.c -list files.lst \  
-alias"BIN=C:\project files\prqa\qac-5.0\mcm\bin" \  
-exec $BIN\qacuser_mcm##%F -exec $BIN\misrauser##%F \  
-alias "IE=C:\Program Files\internet explorer\iexplore.exe" \  
-glob "$BIN\mcmprj#%L+ %R+ -html %P+#$IE"
```

To view analysis results on standard output as analysis of each file completes use

```
qaw qac file1.c -disp
```

To view analysis results with the Message Browser on completion of analysis of each set of files use

```
qaw qac file1.c file2.c diff.prj -view
```

To view the warning listing output on completion of analysis of each set of files use

```
qaw qac file1.c file2.c diff.prj -pdspd
```

The -plog and -glog options cause QAW to display a log of progress information, as the analysis proceeds, of each file as it is analysed. For example, to invoke the progress log and, on completion, the Message Browser, use

```
qaw qac diff.prj -plog -view
```

The -afe and -dafe options both relate to filename extensions. -afe can be used to specify the list of extensions to be allowed for analysis. This can be made totally open by using a wildcard, e.g.

```
qaw qac -op . -list lof -via ..\per.via -afe ".*" -plog
```

- will result in all files in lof being analysed, irrespective of extension.

Normally QAW gives error messages if unrecognised file extensions are encountered. These messages may be suppressed using -dafe-, e.g.

```
qaw qac -op . -list lof -afe .c -plog -dafe-
```

- will result in analysis of all files in lof with extension .c, any others to be silently ignored.

There are no restrictions on combining options, any combination may be used. For overriding options, should an option be repeated with a different parameter, the last instance of the option in the command set will become the setting that is actually used. For cumulative options, all settings are significant.

Options are evaluated from left to right along the command line. When specifying Analyser and Compiler Personalities on the command line, the Analyser Personality should appear before the Compiler Personality. The implications of order are that files and projects will be analysed in the order they appear on the command line, and that secondary and project wide analysis will be executed in the order they are written, but note that all projects are analysed before any individual files.

QAW Commands Reference

-afe

Syntax:

```
-afe "<period separated list of extensions>"
```

Example:

```
-afe ".c.h.c++.any*"
```

Specifies the permissible set of file name extensions for analysis files.

Default is:

```
".c.C.cc.CC.cpp.cxx.c++.h.H.hpp.hxx.h++.i.I.ipp.ixx.i++"
```

Extension specifications are case sensitive and may be wildcarded with '?' or '*'.

Extension specifications are separated by '!'.

-alias

Syntax:

```
-alias "<lhs>=<rhs>"
```

Example:

```
-alias "ANSI=C:\Program Files\prqa\QAC-5.0\include\ansi"
```

Where *lhs* is the string to be replaced at each instance of *\$lhs* or */\${lhs}* throughout the QAW option parameters set and file specifications set with the string represented by *rhs*.

QAW attempts to expand -alias definitions prior to expanding environment variable instances.

QAW will not modify any instance of *\$string* or */\${string}* where no previous definition is found.

-canal

Syntax:

```
-canal <configured project wide analysis program name>
```

Example:

```
-canal $QACBIN\pal.exe
```

This is a shortcut for -glob where the program to be run is present in the Windows QAC.ini configuration file, or the Unix .xqacrc file. The above example is (normally) a shortcut for:

```
-glob "$QACBIN\pal.exe#%Q %L+ %P+"
```

-cargs

Syntax:

```
-cargs
```

Example:

```
qaw qac -via person.p_a -via person.p_c -cargs gcc -I\include_path -Ddef=val source.c
```

This option (Compilation ARGumentS) is provided for make operations where compilation is required as well as analysis. Should compilation not be required, i.e. all files in the Makefile are to be analysed only, then use option -targs instead.

QAW uses the token immediately after the -cargs marker token as the program name of the compiler to be used.

QAW performs analysis of files on the right of the -cargs marker (except the first, i.e. compiler path, token) using QAW options specified on the left of the -cargs token and options on the right of the -cargs token that have -sat translations specified.

QAW compiles the files specified on the right of the compiler name token immediately after the -cargs token using the supplied (unchanged) options on the right of the compiler name token.

QAW translates any option tokens in the -cargs section and uses those options for the (primary and secondary) analysis parameters of the files in the -cargs section. The translation of parameter tokens in the -cargs section is performed as defined by the -sat option(s).

QAW and -cargs is intended for use as a compiler wrapper when running Make files. Example usage is:

```
make "CC=qaw qac -via person.p_a -via person.p_c -cargs gcc"
```

This has the effect of replacing all 'make' invoked Makefile CC compilation operations. For example, applying the above CC definition to the following:

```
CC -c -I \usr\include a.c b.c c.c
```

Results in execution of:

```
qaw qac -via person.p_a -via person.p_c a.c b.c c.c
```

```
gcc -c -I \usr\include a.c b.c c.c
```

-cargs acts as a separator between QAW options and the 'make' provided parameters. Tokens to the left of the -cargs token are passed directly to QAC. Tokens to the right are passed to QAC if they are source files or if they are options for which -sat translation definitions are present. The first parameter after -cargs is the name of the compiler to be run (this is the compiler that is being 'wrapped'). All tokens to the right of -cargs are passed to the compiler.

The key to passing compiler *options* (like -I etc) to QAC, is the -sat option. (-sat is -SetArgumentTranslation.) -sat options must appear before the -cargs token on the command line (they really belong in the compiler personality). Each -sat introduces an option translating specification that can be thought of as a simple edit replace option (s/old/new/).

Note that options (like -I say) after the -cargs marker are not applied to QAC by default. If this is desired, specify the -sat definitions to perform the translations for the required options. For example if -I can appear to the right of -cargs, and this option is to be passed to QAC, then a -sat translation for that -I should be specified. Note, a -sat translation is required even if the translation does not change the option. This is because -sat options do two things:

1. They cause all instances of a given option on the right of -cargs to be passed to QAC.
2. They allow that option to be 'translated' into one or more QAC options which can differ textually.

An example CC definition using -sat is:

```
"CC=qaw qac -via person.p_c -sat I=-i -satst=-ST -cargs gcc"
```

-crep

Syntax:

```
-crep <configured report name>
```

Example:

```
-crep "Code Review"
```

This is a shortcut for -glob where the report to be run is present in the Windows QAC.ini configuration file, or the Unix .xqacrc file.

-dafe

Syntax:

```
-dafe+|-
```

Example:

-dafe-

Instructs QAW to display unrecognised analysis file name warnings when set as -dafe+ or -dafe.

When set to -dafe-, this option results in unrecognised analysis file name warnings being suppressed.

-disp

Syntax:

```
-disp
```

Example:

```
-disp
```

Instructs QAW to generate textual annotated source output for the currently analysed files. This text is directed to stdout and/or stderr and/or file, depending on the setting of -logmsgs.

-html

Syntax:

```
-html
```

Example:

```
-html
```

Ensures that current .html files exist for each file in the fileset. This is done prior to executing options -glob, -pdsp, -pdspd, -view.

-txt

Syntax:

```
-txt
```

Example:

```
-txt
```

Ensures that current .txt files exist for each file in the fileset. This is done prior to executing options -glob, -pdsp, -pdspd, -view.

-exec

Syntax:

```
-exec argument
```

Example:

```
-exec "C:\Program Files\prqa\QAC-5.0-P3\mcm\bin\qacuser_mcm.exe##%F"
```

Where *argument* = *Secondary-Analysis_executable_path* [*# script_path* [*# parameters*]]

argument should be enclosed in quotes if there are any spaces in any of its 3 components.

Quotes should not be embedded in the *-exec parameters*. Should paths exist in the *parameters* section with embedded spaces, escape each significant embedded space with \ (backslash).

```
-exec "$MCM\bin\qacuser_mcm##-param some\ thing"
```

acts as the delimiter of the 3 components. Null parameters are specified by simply omitting the text of the parameter.

Secondary-Analysis_executable_path is mandatory and should be absolute.

script_path is optional, if present it specifies the script that is executed by the interpreter which is *Secondary-Analysis_executable_path*.

parameters specifies the parameters to the program / script as a string with placeholders that are replaced as follows. Note that the Unix version of QAC differs from the Windows version.

Windows GUI:

%Q - QAC
%P - Personality file (settings.via)
%N - Naming rule configuration file
%F - Name of source file being analysed
%S - Location within the command string to insert the script_path component

"+" can follow %P, %N with following results:

%P+ => -via *settings.via*
%N+ => -nrf <naming rule file>.html

Unix GUI:
%Q - QAC
%M - met file for source file being analysed
%E - err file for source file being analysed
%P[+] - [-via] settings.via
%F[+] - [-op <output path>] Name of source file being analysed

-fdisp

Syntax:

```
-fdisp <file name>
```

Example:

```
-fdisp
```

QAW generates textual annotated source output for the currently analysed fileset. This text is written to *file name* regardless of the -logmsgs setting. The -fdisp output file is emptied at the start of each QAW command, it is appended to for each file analysed during the remainder of that QAW command.

-glob

Syntax:

```
-glob argument
```

Example:

```
-glob "C:\Program Files\prqa\QAC-5.0\mcm\mcmprj#-list %L -html %R+ %P+#Iexplore"
```

argument = *project_wide_analysis_executable_path* [# *parameters* [# *rendering_program*]]

rendering_program may be omitted, in which case the browser specified by the environment variable PRQA_DEFAULT_BROWSER is used.

argument should be enclosed in quotes if there are any spaces in any of its 3 components.

Quotes should not be embedded in the -glob *parameters*. Should paths exist in the *parameters* section with embedded spaces, escape each embedded space with \ (backslash).

```
-glob "C:\Program Files\prqa\QAC-5.0\mcm\mcmprj#%L+ some\ thing#netscape"
```

acts as the delimiter of the 3 components. Null parameters are specified by simply omitting the text of the parameter.

project_wide_analysis_executable_path is mandatory and should be absolute.

parameters specifies the parameters to the program as a string with placeholders that are replaced as follows.

%D - insert default source path of root folder selected
%J - insert project name
%L - insert name of file listing all files for this global analysis
%O - insert output path of root folder selected
%P - include settings.via
%Q - QAC
%R - placeholder for result file which is named <project wide analysis program name>.html by the GUI
%T - temp dir

"+" can follow %L, %P, %R with following results:

```
%L+    => -list filelist.lst
%P+    => -via settings.via
%R+    => -file <program>.html
```

Project wide analysis output is controlled within the parameters specification as follows.

In order for output to be displayed by the *rendering_program* the *parameters* section must contain a '**-file <filename>**' option. This may be specified explicitly by embedding '**-file filename**' into the *parameters* section, or by using the **%R+** option. Note that QAW will not allow a parameters section that expands to more than one '**-file**' option. Hence zero or one '**-file**' options should result on expansion.

A special case of *parameters* is where output is to be directed to Standard Output. In this case the User should include '**-file STDOUT**' in his *parameters* section. Hence **-file** is used in 3 ways:

1. No **-file** option
This means do not execute the *rendering_program*.
2. **-file STDOUT**
This means copy result of *project_wide_analysis_executable_path* to Standard Output.
3. **-file filename**
This means *project_wide_analysis_executable_path* creates filename and displays using *rendering_program*.

rendering_program is the program specification ([path]name) necessary to invoke an html browser and is optional. If omitted QAW will use the browser specified in the Environment Variable PRQA_DEFAULT_BROWSER.

-glog

Syntax:

```
-glog
```

Example:

```
-glog
```

Invokes QAW output to its a graphic window display a log of analysis activity. TBD.

-help -h

Syntax:

```
( -h | -help ) [ <list of options> ]
```

Example:

```
-help -i -x
```

When invoked with no parameters, QAW lists brief help information for each option.

When invoked with parameters, QAW lists brief help information for each option that matches the *list of options*. Where a match is made on any options in the full set of options available (ie QAW options and the QAC/PP toolset options), that commence with the characters supplied for each option id *list of options*.

-logerrs

Syntax:

```
-logerrs [STDOUT],[STDERR],[FILE]
```

Example:

```
-logerrs STDOUT,FILE
```

QAW writes all error output to the specified destination(s). Default is STDERR.

On specification of '**-logerrs FILE**', QAW writes all error output to the file qaw_err.log.

-logmsgs

Syntax:

```
-logmsgs [STDOUT],[STDERR],[FILE]
```

Example:

```
-logmsgs STDOUT,FILE
```

QAW writes all message output to the specified destination(s). Default is STDOUT.

On specification of '-logmsgs FILE', QAW writes all message output to the file qaw_msg.log.

-maseq

Syntax:

```
-maseq argument
```

Example:

```
-maseq "mcmprij -list %L -html %R+ %P+#pal %Q %L+ %P+"
```

Where *argument* = *program_spec* { # *program_spec* }

program_spec = *project_wide_analysis_executable_path* space *parameters*

argument should be enclosed in quotes if there are any spaces within it.

Quotes should not be embedded in any *program_spec* term. Should significant spaces exist in any part of *program_spec*, escape each embedded space with \ (backslash).

```
-maseq "C:\Prog\ Files\bin\prg %Q %L+ -param some\ thing"
```

acts as the delimiter of each program specified in the sequence.

project_wide_analysis_executable_path is mandatory and should be absolute unless the program specified is on the system PATH variable.

parameters specifies the parameters to the program as a string with placeholders that are replaced as follows.

%D	- insert default source path of root folder selected
%J	- insert project name
%L	- insert name of file listing all files for this global analysis
%O	- insert output path of root folder selected
%P	- include settings.via
%Q	- QAC
%R	- placeholder for result file which is named <project wide analysis program name>.html by the GUI
%T	- temp dir

"+" can follow %L, %P, %R with following results:

%L+	=> -list <i>filelist.lst</i>
%P+	=> -via <i>settings.via</i>
%R+	=> -file <program>.html

-mode

Syntax:

```
-mode ( ALL | DEPEND | DEP | NONE )
```

Example:

```
-mode depend
```

Where parameter is:

ALL	This is the default. In this mode all files are analysed.
DEP or DEPEND	Analyse only those files for which the current results are or may be out of date.
NONE	Skip analysis of files and simply execute display options.

-mode controls whether files in the fileset submitted to QAC are always analysed, analysed only if necessary, or not analysed. Files submitted with -mode none result in QAW performing the specified display options without running any analysis.

-pdsp

Syntax:

```
-pdsp <browser program spec>
```

Example:

```
-pdsp iexplore
```

Where *browser program spec* is the full path to the html browser to be used to display prjdsp output, or is the name of an html browser executable on the OS PATH.

Invokes the PRQA prjdsp component on each fileset and uses the supplied browser to display the result.

Automatically invokes -html to ensure that html files exist and are up to date for the Warning Listing generated.

-pdspd

Syntax:

```
-pdspd
```

Example:

```
-pdspd
```

Invokes the PRQA prjdsp component on each fileset and uses the browser specified by PRQA_DEFAULT_BROWSER to display the result.

Automatically invokes -html to ensure that html files exist and are up to date for the Warning Listing generated.

-plog

Syntax:

```
-plog
```

Example:

```
-plog
```

QAW displays to its logmsgs device(s) (stdout / stderr / qaw_msg.log) a log of analysis activity.

-sapf

Syntax:

```
-sapf
```

Example:

```
-sapf
```

QAW outputs, for each file analysed, a report comprising the options used for analysis and display.

-saseq

Syntax:

```
-saseq argument
```

Example:

```
-saseq "C:\Program\ Files\prqa\QAC-5.0-P3\mcm\bin\qacuser_mcm.exe %F"
```

Where *argument* = *program_spec* { # *program_spec* }

program_spec = *Secondary-Analysis_executable_path* space *parameters*

argument should be enclosed in quotes if there are any spaces within it.

Quotes should not be embedded any *program_spec* term. Should significant spaces exist in any part of *program_spec*, escape each embedded space with \ (backslash).

```
-saseq "C:\Prog\ Files\bin\prg %F -param some\ thing"
```

acts as the delimiter of each program specified in the sequence.

Secondary-Analysis_executable_path is mandatory and should be absolute unless the program specified is on the system PATH variable.

parameters specifies the parameters to the program as a string with placeholders that are replaced as follows. Note that the Unix version of QAC differs from the Windows version.

Windows GUI:

%Q - QAC

%P - Personality file (settings.via)

%N - Naming rule configuration file

%F - Name of source file being analysed

%S - Location within the command string to insert the script_path component

"+" can follow %P, %N with following results:

%P+ => -via *settings.via*

%N+ => -nrf <naming rule file>.html

Unix GUI:

%Q - QAC

%M - met file for source file being analysed

%E - err file for source file being analysed

%P[+] - [-via] *settings.via*

%F[+] - [-op <output path>] Name of source file being analysed

-sat -SetArgumentTranslation

Syntax:

```
-sat <lhs>=<rhs>
```

Or

```
-SetArgumentTranslation <lhs>=<rhs>
```

Example:

```
qaw qac -sat "MTd=-d _MT -d _DEBUG" -cargs cl /c /MTd file.c
```

Causes following analysis:

```
qaw qac -d _MT -d _DEBUG file.c
```

and executes:

```
cl /c /MTd file.c
```

Where *lhs* represents the characters of an option token to be replaced.

N.B. the argument is case sensitive.

N.B. the *lhs* is specified as the compiler option to be matched less its first character. This character is either '-' or '--'. Hence, normally there is no '-' or '--' prefix to *lhs*, however the replacing (QAC/PP) options in *rhs* ARE prefixed with '-'.

That is:

```
-sat D=-D -sat D=-d -sat "D=-DWIN -D"
```

are normal.

If the option to be modelled starts with more than one '-' character, for example --DEBUG, then specify -sat as:

```
-sat-DEBUG=...
```

This option specifies a replacement set of tokens (ie *rhs*) that are inserted at every instance of *-lhs* or *//hs* in the command line tokens following the -cargs or -targs option.

-sfba

Syntax:

```
-sfba
```

Example:

```
qaw qac mcm.prj -sfba -maseq "pal %Q %P+ %L+"
```

Suppresses primary and secondary analysis, allowing only project wide (global) analysis of the given fileset.

-stoponfail

Syntax:

```
-stoponfail
```

Example:

```
-stoponfail
```

QAW terminates on primary, secondary, or global analysis failure, with QAW returning the return code of the analysis program that failed.

-targs

Syntax:

```
-targs
```

Example:

```
qaw qac -via person.p_a -via person.p_c -targs -I\include_path -Ddef=val source.c
```

This option (Translation ARGumentS) is provided for Make based PRQA analysis where compilation options of a Makefile are to be translated into analysis options without compilation, i.e. all files in the Makefile shall be analysed only. In this mode no compilation takes place, should compilation be required the partner option `-cargs` should be used instead.

QAW performs analysis of files on the right of the `-targs` marker, using QAW options specified on the left of the `-targs` token and options on the right of the `-targs` token that have `-sat` translations specified.

QAW translates any option tokens in the `-targs` section and uses those options for the (primary and secondary) analysis parameters of the files in the `-targs` section. The translation of parameter tokens in the `-targs` section is performed as defined by the `-sat` option(s).

QAW `-targs` is intended for use as a compiler wrapper when running Make files. Example usage is:

```
make "CC=qaw qac -via person.p_a -via person.p_c -targs"
```

This has the effect of replacing all 'make' invoked Makefile CC compilation operations. For example, applying the above CC definition to the following:

```
CC -c -I \usr\include a.c b.c c.c
```

Results in execution of:

```
qaw qac -via person.p_a -via person.p_c a.c b.c c.c
```

`-targs` acts as a separator between QAW options and the 'make' provided parameters. Tokens to the left of the `-targs` token are passed directly to QAC. Tokens to the right are passed to QAC if they are source files or options for which `-sat` translations exist.

The key to passing compiler *options* (like `-I` etc) to QAC, is the `-sat` option. (`-sat` is `-SetArgumentTranslation`.) `-sat` options must appear before the `-targs` token on the command line (they really belong in the compiler personality). Each `-sat` introduces an option translating specification that can be thought of as a simple edit replace option (`s/old/new/`).

Note that options (like `-I` say) after the `-targs` marker are not applied to QAC by default. If this is desired, specify the `-sat` definitions to perform the translations for the required options. For example if `-I` can appear to the right of `-targs`, and this option shall be passed to QAC, then specify a `-sat` translation for that `-I`. A `-sat` translation is required even if the translation does not change the option. This is because `-sat` options do two things:

1. They cause all instances of a given option on the right of `-targs` to be passed to QAC.
2. They allow that option to be 'translated' into one or more QAC options which can differ textually.

An example CC definition using `-sat` is:

```
"CC=qaw qac -via person.p_c -sat I=-i -satst=-ST -targs"
```

-trace

Syntax:

```
-trace
```

Example:

```
-trace
```

The `-trace` option causes QAW to create a file named `qaw.trace` with the following content:

- i. List of entire set of options passed to QAW.
 - ii. Full text used to invoke each PRQA program.
-

-version -v -ver

Syntax:

```
-v | -ver | -version
```

Example:

```
-version
```

QAW displays the QAW program version string, e.g.

```
qaw version 2.0
```

-view

Syntax:

```
-view
```

Example:

```
-view
```

QAW executes the Message Browser with the current filesset.
