



# HelixQAC

---

Manual

Version: 2021.1

PERFORCE

[www.perforce.com](http://www.perforce.com)

© Perforce Software, Inc. All rights reserved.

## Important Notice

### Disclaimer of Warranty

This document should only be used in conjunction with Helix QAC.

Programming Research Ltd have taken due care in preparing this document which it has endeavored to ensure is accurate at the time of printing. However, no liability can be accepted for errors or omissions; nor should the document be considered as an expressed or implied warranty of accuracy or completeness, fitness for a particular purpose, or that the products described perform as specified within.

### Copyright Notice

All rights reserved. No part of this document may be reproduced, stored in a retrieval system of any nature, or transmitted in any form or by any means, including photocopying and recording, without the prior written permission of Programming Research Ltd., the copyright owner. If any unauthorized acts are carried out in relation to this copyrighted work, a civil claim for damages may be made and/or a criminal prosecution may result. Copyright ©Programming Research Ltd 2021.

### Trademarks

Helix QAC, the Helix QAC logo, Helix QAC for C, Helix QAC for C++ and High Integrity C++ (HIC++) are trademarks of Programming Research Ltd.

"MISRA", "MISRA C" and "MISRA C++" are registered trademarks of HORIBA MIRA Limited, held on behalf of the MISRA Consortium.

"AUTOSAR" is a registered trademark of AUTOSAR GBR, held on behalf of the AUTOSAR Development Partnership.

Yices is a registered trademark of SRI International.

Windows is a registered trademark of Microsoft Corporation.

## Contacting Programming Research Ltd

For technical support, contact your nearest Programming Research Ltd. authorized distributor or, alternatively, the Programming Research Ltd head office as follows

by telephone on +44 (0) 1932 888 080

by fax on +44 (0) 1932 888 081

or by webpage: <https://www.perforce.com/support/request-support>

# Contents

Important Notice .....	2
Disclaimer of Warranty .....	2
Copyright Notice .....	2
Trademarks .....	2
Contacting Programming Research Ltd .....	3
<b>1   Introduction .....</b>	<b>13</b>
Interfaces .....	13
Desktop versus Server-side Working .....	13
Diagnostics Database .....	14
<b>2   Getting Started .....</b>	<b>15</b>
Basic Concepts .....	15
Setting the User Data Store .....	16
Performance Optimization on the User Data Store .....	16
Creating a Helix QAC Project .....	16
Naming Your Project .....	17
Choosing Configuration Files .....	17
Setting the File Extensions for Your Sources .....	18
Setting the Root Directory of Your Sources .....	18
Extraction of Configuration Data .....	20
For Plug-in Users .....	20
For C and C++ Users .....	20
Compiler Wrapping .....	21
Manual Extraction .....	22
Scripted Extraction .....	23
Analyzing Your Project .....	24
QA·GUI .....	24
Analysis Dialog Box .....	24
Helix QAC Eclipse .....	24
Helix QAC Visual Studio .....	24
QA·CLI .....	24
<b>3   GUI Overview .....</b>	<b>27</b>
GUI Features .....	28
Opening a Project (from the CLI) .....	28

Opening a Project .....	28
Viewing the Diagnostics .....	29
Files Panel .....	29
Message Levels Panel .....	31
Rule Groups Panel .....	31
Viewing Diagnostic Messages In-Line .....	32
Viewing and Filtering the Diagnostics .....	33
Whole Project Analysis Hard Errors Panel .....	34
Menu-based Operations .....	35
The Project Menu .....	35
The Admin Menu .....	36
The Analyze Menu .....	38
The Report Menu .....	39
The Helix QAC Dashboard Menu .....	39
The View Menu .....	41
The Help Menu .....	41
<b>4   Projects .....</b>	<b>43</b>
Using the Project Creation Wizard .....	43
Page one: General Settings .....	43
Project Location .....	43
Project Name .....	44
Config Name .....	44
Analysis Configuration File .....	44
Rule Configuration File .....	44
Compiler Settings .....	44
Page two: Compiler Compatibility Templates .....	45
Page three: Unified Projects .....	46
C and C++ Compile Dependencies .....	46
Populating Your Project .....	47
Adding Files .....	47
Observing a Build .....	47
Project Portability and Sharing .....	48
Generic Project Portability .....	48
Project-Specific Portability .....	48

Compiler Selection .....	49
CCT Filtering .....	49
Selecting a CCT .....	49
Updating a Project CCT .....	49
CIP Settings .....	49
Baseline Diagnostics Suppression .....	50
Generating a Baseline .....	50
Baseline by Dashboard snapshot .....	50
Baseline by Version Control .....	51
Baseline by Local Copy .....	51
Projects with Multiple Configurations .....	51
Restrictions .....	51
Multiple Configurations with QA GUI .....	52
Selection of Alternative Project Configurations .....	52
Configuration Manager .....	52
Configuration Changes - Project Properties .....	52
Multiple Configurations with Helix QAC-CLI .....	53
Deleting a Project .....	53
Optimizing a Project .....	54
<b>5   Analysis Configuration .....</b>	<b>55</b>
The Analysis Tab .....	55
The Analysis Configuration Panel .....	55
Importing, Exporting and Clearing Configuration Settings .....	55
User Messages File .....	55
Creating User Messages .....	56
Selecting the Source Language Toolchain .....	56
Mixed Language Toolchain .....	56
Common C and C++ Components Toolchain .....	56
Selecting the Toolchain Components .....	57
<b>6   Rule Configuration .....</b>	<b>59</b>
The Default Rule Configuration Files .....	59
Rule Configuration Operations .....	59
Edit .....	59
Remove .....	60

Disable/Enable .....	60
New Rule .....	60
Messages .....	60
Associating Messages with Rules .....	60
Disassociating Messages from Rules .....	61
New Rule Configuration .....	61
<b>7   Project Synchronization .....</b>	<b>63</b>
Synchronization via Process Monitoring .....	63
Process Monitoring via QA·CLI .....	63
Process Monitoring via QA·GUI .....	64
Verifying Process Monitor Synchronization .....	65
Synchronizing a Visual Studio Project .....	65
Preparing the Helix QAC Visual Studio Plug-in prior to Synchronization .....	65
Performing the Visual Studio Synchronization using the GUI .....	66
Verifying the Outcome of the Synchronization .....	67
Manual Synchronization Methods .....	67
JSON Compilation Database .....	67
Build Log .....	68
Synchronization Options .....	68
Include Path Options .....	69
Define Symbol Options .....	69
Compiler Settings File Options .....	69
Exclude Processes .....	69
File Filter .....	70
Suppress Include Paths .....	70
<b>8   Version Control Configuration .....</b>	<b>71</b>
VCF File Testing .....	71
Using Helix QAC Without a Version Control System .....	72
<b>9   Working with Helix QAC Dashboard .....</b>	<b>73</b>
Connecting to Helix QAC Dashboard .....	73
Disconnecting from Helix QAC Dashboard .....	74
Centralizing Project Definitions .....	74
Creating a Unified Project .....	74
Downloading a Unified Project .....	76

Working on a Unified Project .....	77
Updating a Unified Project .....	77
Maintaining the Unified Ruleset .....	77
Creating a Unified Project as a New Project .....	77
Uploading Results to Helix QAC Dashboard .....	78
Uploading via QA GUI .....	78
Viewing a Snapshot .....	79
Downloading a Baseline .....	79
Downloading Suppressions .....	79
<b>10   Cross-Module Analysis (CMA) .....</b>	<b>81</b>
Single and Multi-Project CMA .....	81
Single-Project CMA Analysis .....	81
Multi-Project CMA Analysis .....	82
CMA Project Editor .....	82
Viewing CMA Data .....	84
Cleaning CMA Data .....	84
<b>11   Reports .....</b>	<b>85</b>
Introduction to reports .....	85
Standard Report Types .....	85
Components of Function Structure .....	86
Straight Code .....	86
If .....	87
If-Else .....	87
Switch .....	88
While Loop .....	89
For Loop .....	89
Nested Structures .....	90
Break in a Loop .....	90
Return in a Loop .....	91
Continue in a Loop .....	92
Unreachable Code .....	92
Custom Report Plug-ins .....	93
Report Generator Naming .....	95
Improving Python Performance .....	95



Ixml Installation .....	95
<b>12   QA·CLI .....</b>	<b>97</b>
Introduction to QA-CLI .....	97
Common Error Messages .....	98
< n > success and < m > failures .....	98
< file > --no results for this files in the database .....	98
< file > --zero diagnostics found .....	98
Progress Output .....	98
Filelists .....	99
Bash command line completion for QACLI .....	99
Admin .....	100
Installation Management .....	100
Helix QAC Project Management .....	101
Configuration File Management .....	103
CMA Project Management .....	104
Extraction of C/C++ Analysis Data .....	105
Adding Files Explicitly .....	105
Updating Folders .....	105
Optimization .....	106
Removing Files Explicitly .....	106
Removing Folders .....	106
Optimization .....	106
Specifying Helix QAC Dashboard Credentials .....	106
Working with Unified Projects .....	107
Upgrade .....	108
License Server Management .....	109
User Data Location Management .....	110
Administration of User Messages File .....	110
Source Extensions .....	111
Analyze .....	111
Analysis of Helix QAC Projects .....	111
Retry after analysis failure .....	113
Monitoring Progress .....	113
Analysis Timings .....	113

Simultaneous Synchronization and Analysis .....	114
Analysis of CMA Projects .....	114
Raw Source Analysis .....	115
Debugging Analysis .....	115
Baseline .....	115
Setting a Dashboard Baseline .....	115
Setting a Version Control Baseline .....	116
Setting a Local Baseline .....	116
Downloading Dashboard Suppressions .....	117
Export .....	117
Exporting CCTs .....	118
Example Usage .....	118
Help .....	119
Import .....	119
Importing CCTs .....	120
Example Usage .....	120
Log .....	121
Pprops (Project Properties) .....	122
Listing Components .....	122
Adding a component to a toolchain in a project .....	122
Upgrading a toolchain component in a project .....	123
Removing a toolchain component from a project .....	123
Component Options .....	123
Component Preset Groups .....	125
Sync Settings .....	126
CIP Settings .....	127
Miscellaneous Settings .....	128
Report .....	129
Sync .....	130
MONITOR .....	131
BUILD LOG .....	131
JSON .....	132
MSVS .....	132
Upload .....	133

Upload to Helix QAC Dashboard .....	133
Upload to Structure101 .....	134
Upload of Selected Files .....	134
Generate Dashboard Configuration .....	135
View .....	135
Annotated Source .....	138
SUMMARY .....	138
Viewing CMA Projects .....	139
Filtering Messages .....	139
Formatting the Diagnostic String .....	140
Table of suppression types .....	141
Conditional Formatting .....	141
Message Text Control .....	143
An Example Message Format .....	144
Default Format Strings .....	144
Formatting in Windows command files (.bat) .....	145
XML Output .....	145
XML Indent Size .....	145
XML File Format .....	145
XML Content .....	147
Specifier to XML mapping .....	147
XML Schema .....	148
XML Examples .....	148
Deprecated Commands .....	149
Table of deprecated commands .....	150
<b>13   Upgrading Helix QAC .....</b>	<b>151</b>
User Settings .....	151
Upgrading Projects .....	151
Missing Component Dialog Box .....	151
Updating the Analysis Toolchain .....	152
Installing the Missing Component .....	152
<b>14   Adding Translations to Helix QAC .....</b>	<b>153</b>
<b>15   Glossary .....</b>	<b>155</b>
Glossary of Terms .....	155

Table of Helix QAC Terms .....	155
Glossary of Acronyms .....	155
<b>A   QA-CLI Return Codes .....</b>	<b>157</b>
<b>B   Keyboard Shortcuts .....</b>	<b>159</b>
<b>C   Validation .....</b>	<b>161</b>
Filenames .....	161
Pathnames .....	162
<b>D   Environment Variables .....</b>	<b>163</b>

# 1 | Introduction

Helix QAC is a Source Code Analysis Management framework for managing source code analysis projects and analyzers. It is designed to provide you with an Analysis System that handles several programming languages, using selectable analysis components. In the case of C and C++, it supports mixed language projects that use both languages.

## Note:

Help on individual, associated components is available via the Helix QAC Welcome Page and the Help Menu. As regards the full documentation set, the PDF manuals and Help for the individual components can be found under the Helix QAC installation folder. From there, go to "components", then to the required folder (for example, "qac" for Helix QAC for C or "qacpp" for Helix QAC for C++), open the "doc" sub-folder for your language (for example, "doc-en\_US"), and then select "component\_manual" followed by "html" and the top .html file, or else "pdf" and the relevant .pdf file. You will need a PDF reader such as Acrobat to open this latter file.

The default setup uses the underlying Primary Analysis components such as the Helix QAC for C and Helix QAC for C++ parsers, but its strength lies in the flexibility of configuring an analysis chain, using other Secondary Analysis components.

The **Rule Configuration** utility allows you to customize user messages and rules to suit company standards.

## Interfaces

Helix QAC provides four interfaces through which all features are presented to the user:

### QA·GUI

A Graphical User Interface to Helix QAC.

### QA·CLI

A Command Line Interface to Helix QAC.

### QA·Visual Studio

A Microsoft Visual Studio Integration that provides an interface to Helix QAC.

### QA·Eclipse

An Eclipse Integration that provides an interface to Helix QAC.

The above interfaces are available from both the Linux and Windows Helix QAC packages, with the exception of QA·Visual Studio, which is only available on Windows.

## Desktop versus Server-side Working

The variety of interfaces provided by Helix QAC means that two main usage patterns are possible:

- Use on the desktop by a developer
- Use as part of an automated build-test-distribute workflow

Desktop usage allows rapid correction of problems by the developers, as the code is being written. It is good practice for developers to run an analysis before checking any code into the version control system, just as unit tests would normally be run. In most cases, the analysis can be run and results viewed directly from the developer's IDE. QA·GUI can be used if a non-supported IDE is being used for development.

Server-side usage would normally be run as part of a Continuous Integration workflow, or an overnight build process. Complete automation is possible, as QA·CLI commands can be called from scripts, which in turn can be called from CI tools. There is also a Jenkins plug-in available. Ideally, the results of the server-side processing should be uploaded into Dashboard for further analysis and distribution, but Helix QAC also has reports that can be run directly from QA·CLI.

Server-side processing options are known, and typically fixed, which means the results from Helix QAC are reliable and therefore can be easily used in an audit trail by customers.

These two usage patterns work well together. The Desktop usage helps to minimize the number of issues that reach the server-side processing. Likewise, having these results available from the server means that Desktop processing can skip these options, making it quicker to run.

## Diagnostics Database

Diagnostics are stored in the QAX database, using a process (called `qaxd`) which runs as a daemon, starting and stopping as required. There is one database per configuration in a project, located in the `prqa/configs/<config_name>/output` folder of the project, and called `prqa.db`. Unless directed by Support, you must not alter this database manually; it should only be modified automatically by Helix QAC and its components.

## 2 | Getting Started

This section is a guide to getting started with Helix QAC and introduces basic concepts and usage.

### Basic Concepts

The more information the analyzer knows about the compilation environment of a file, the more accurate and consistent results will be.

As an example, take the following piece of C code:

```
// file.h
#define MACRO 1
unsigned int i
; int foo(void);

// file.c
int foo(void)
{
    return i = MACRO + CLIMACRO;
}
```

which is compiled with the following command:

```
gcc -o file.o -DCLIMACRO=2 -I. file.c
```

Looking at file.c, we can see that there are several identifiers that are not resolved within that file itself: `foo`, `i`, `MACRO` and `CLIMACRO`. This poses a question for any designer of a static analyzer or compiler:

- What to do when an identifier is not resolved in the source file within which it was declared or used.

How this question is answered determines the robustness of analysis results. Programming Research's primary analysis components do not simply assume that identifiers and language constructs correctly resolve to something safe and legal: Helix QAC parsers resolve identifiers and determine the resolved values.

For this reason, the analysis components need to know the information that your compiler knows about your source files when they are compiled. This includes compiler specifics, such as:

- the compiler's built-in extensions to the language,
- the location of compiler include directories or references,
- the sizes of the various types known to your compiler,

and also build environment information, such as:

- the search paths that are used when compiling your source files to locate include files or references,
- environment variables that are present when compiling your source files,
- macros that are passed to your compiler on the command line while compiling.

This amounts to a great deal of data, data that may change from project to project, subproject to sub-project or, indeed, source file to source file. Fortunately, Helix QAC is equipped with features that automate the extraction of this data.

As the information that accompanies a source file when being compiled is intrinsic to the resolution of all, not some, types and identifiers, Helix QAC has not been designed as a tool that leads the user to simply point to a source file and expect coherent analysis results. The apparent complication of the setup process is the price to pay in order to have insightful and complete diagnostic data as an outcome from the analysis results.

[Creating a Helix QAC Project](#) will guide you through creating your first Helix QAC project. [Extraction of Configuration Data](#) describes the various data extraction features that are available so that you may choose one that suits your environment. [Analyzing Your Project](#) provides instructions on the various ways in which you can analyze your new project.

## Setting the User Data Store

In order to operate, Helix QAC stores user settings, sample projects, default files and help in the following default locations.

- **Windows:** %LOCALAPPDATA%\Perforce\Helix-QAC-2019.1
- **Linux:** \$HOME/.config/Perforce/Helix-QAC-2019.1

Customers are able to change this location via the CLI or the GUI.

For more information on changing the User Data Store, please refer to the following.

- **QA-CLI method:** please see [User Data Location Management](#)
- **QA-GUI method:** please see [The Admin Menu](#)

## Performance Optimization on the User Data Store

In versions prior to Helix QAC 2021.1, every time the CLI or GUI started it would compare every file in the *samples* and *config* directories in the User Data Store against the version in the installation directories and copy any missing files. To improve performance, this no longer happens. Now, on first use, the files in these directories are copied to the User Data Store and a temporary *hash* file is created in the root of each directory. This file has a *hash* value of several digits in its name, and ends in *.hash*, for example *1234666999.hash*. The *hash* value is based on the name and contents of the install directory. If this file is present in the User Data Store it allows the copying operations to be skipped where appropriate, with a resultant uptake in performance.

If you want to force the *samples* and/or *config* directory to be updated from the install directories, simply delete the *.hash* file(s) and the working files will be copied the next time that any Helix QAC command is issued or that the GUI is started.

## Creating a Helix QAC Project

The steps to creating your first Helix QAC project are as follows:

- Choose a name for your project, in accordance with Section [Naming Your Project](#)
- Choose the compulsory Helix QAC configuration files you wish to use for your project, in accordance with Section [Choosing Configuration Files](#).
- Choose the file extensions of your sources, in accordance with Section [Setting the File Extensions for Your Sources](#).
- Choose the optimum Helix QAC root directory for sources, in accordance with Section [Setting the Root Directory of Your Sources](#).



Once complete, a file named "prqaproject.xml" and a directory named "perforce", containing your chosen configuration files, are created in the project directory. This project does not, as yet, have any sources associated with it. It is termed a "headless Helix QAC project" and can be used as a project template to be used for more projects or to distribute to other machines where Helix QAC is installed (just copy the "prqaproject.xml" file and the "perforce" directory to any directory where you wish to create a project).

## Naming Your Project

Helix QAC is designed to integrate with code projects that already exist. For this reason, a Helix QAC Project Location is the path to the folder containing the Helix QAC project (refer to [Project Location](#)).

Generally, a code base is made up of many smaller sub-projects, components and/or modules that are buildable entities in themselves. Not all sub-projects require the exact same configuration and, just as organizing your code base into smaller cohesive subprojects helps to manage the code base, the same is true for your analysis projects. You are therefore advised to treat the root directory of each cohesive, buildable entity as a separate Helix QAC project, to which a different CCT (Compiler Compatibility Template) can be applied as required.

Naming a new project from QA·GUI:

- **Project menu : New Project** : Project Name:

Naming a new project from QA·CLI:

- `qaccli admin --qaf-project <path> --qaf-project-config --cct <file> --acf <file> --rcf <file>`

If you are using either QA·Visual Studio or QA·Eclipse, your project name will be automatically set to the location for your project or solution.

To create a project in QA·Visual Studio, access the following menu item:

- **Helix QAC menu : Admin** : *Create/Sync Project*

The same can be done in QA·Eclipse by accessing the right-click menu on your project of choice and clicking on the following menu item:

- **Helix QAC sub-menu** : **Convert to Helix QAC Project**

## Choosing Configuration Files

There are certain compulsory configuration files that must be chosen when creating a Helix QAC project. These are:

- at least one CCT (Compiler Compatibility Template)
- one ACF (Analysis Configuration File)
- one RCF (Rule Configuration File).

Helix QAC is shipped with support for numerous compilers. Choose one that is an exact match for the compiler with which you compile your source code. If you use both a C and C++ compiler to build your sources, choose a CCT for each one that matches your compilers. If you do not see a match for your compiler(s) in the provided list, please contact support.

Leave the default choices for the ACF and RCF for now. The ACF specifies a toolchain of analysis components for each source language and the RCF specifies coding standard rules and the messages that enforce each rule.

If you are using QA·Visual Studio, default configuration files will be chosen when your project is created. You can add, change and/or edit these configurations after project creation if you wish, through the Helix QAC **Project Properties** panel. This panel is launched automatically after project creation if the **Open Project Properties** option is checked.

## Setting the File Extensions for Your Sources

Helix QAC determines how to process a source file based on the language of the source file, for example C or C++. It does this using the file extension of the source file and the operating system. For each project, a set of file extensions is mapped to a source language. On Windows systems, the following extensions are used by default:

<b>C</b>	.c .C
<b>C++</b>	.cpp .CPP .cxx .CXX .cc .CC
<b>Web Security (via Checkmarx)</b>	.c .C .cpp .CPP .cxx .CXX .cc .CC .java

On non-Windows systems, the following extensions are used by default:

<b>C</b>	.c
<b>C++</b>	.cpp .cxx .cc .cp .c++ .C
<b>Web Security (via Checkmarx)</b>	.c .cpp .cxx .cc .cp .c++ .C .java

You can change these setting for any project from the main ("Project") tab in **Project Properties**, which can be launched as follows:

From QA GUI:

- **Project menu : Open Project Properties**

From the command line:

```
qapp <path-to-project-root-directory>
```

From QA Visual Studio:

- **Helix QAC menu : Admin : Project Properties**

From QA Eclipse through the right-click menu on your chosen project:

- **Helix QAC sub-menu : Open Helix QAC Project Properties**

## Setting the Root Directory of Your Sources

As stated before, code bases, in general, consist of many projects, sub-projects, components and/or modules. Often, the build of each entity is dependent on information from other entities, such as include directories.

Helix QAC projects are designed to be portable. For example, they can be checked into version control on one machine, and then checked out on another machine for analysis. However, in order for this to work, dependencies, such as third party and project include directories, must be capable of being located wherever the project has been checked out or moved.

Some companies maintain build-dependent directories absolutely. This necessitates that they are installed in the same absolute location on every machine that needs to build the project. Others maintain build-dependent directories relative to the current project. Many companies have a mixture of both absolute and relative build-dependent directories.

Helix QAC allows you to set multiple root directories for source files. More often than not, just a single root directory is enough, but you may need more than one root directory if your project uses third party code, for example.


By default, the source root directory is set to be the same as the project root directory, specified when the project was created. To change the source root directory, or to add additional directories, navigate to:

## Project : Open Project Properties : Project tab : Root Directories

The `PROJECT_ROOT` setting holds the location of the Helix QAC project. This cannot be changed. The `SOURCE_ROOT` setting is initially set to the `PROJECT_ROOT` value, but can be changed to a location more suited to your own file structure. To change the location, double-click on the cell in the Value column, modify the text, and click Save at the bottom of the screen.

### Note:

When you add a relative include path (for example `-i .\aaa\inc`) to a CCT file, `PROJECT_ROOT` is the start directory for that relative path. In the case of this example, `-i PROJECT_ROOT\aaa\inc` is searched.

If you need to have more than one source root directory, click on the **Add** button. In the dialog box that appears, specify a **Name** to describe the location, and either type in the **Directory** location, or use the  icon to the right of the field to browse to it. The location can refer to other root directories by wrapping the name in `${ }`. A path can then be appended, for example: `${PROJECT_ROOT}\src\calcs` to refer to a location two levels below the project root.

You can add files to a project by selecting **Add File(s)** from the context menu for the required project on the main screen .

Helix QAC then determines the "best match" of the file location to a source root. If there is no match, the absolute path to the file is used.

The power of project roots can be seen if a project definition is copied between users. Often, user A will specify the list of source files for the project before copying the project definition across to user B. It may be that the two users have the same location for third party code, but a different location for the source within the project itself. If so, then user B can just adjust the source roots to suit the locations on the local machine. The project is now ready for use. There is no need for user B to re-synchronize the project, as the files added by the user A will be found in their local locations.

In addition, the changes made by user B are not set into the normal project definition file `prqaproject.xml`: rather, they are stored in a user-specific file `"prqa\config\prqa-framework-app.xml"`. Project-specific or user-specific roots can be viewed for editing using the radio buttons beneath the table. The significance is that the original `"prqaproject.xml"` is still valid for all users of the project, and can be checked into source control without worrying about per-user differences. (Newly-added roots will be set into `"prqaproject.xml"` so that they are available to all users.)

Each time that a different source root is specified, Helix QAC will check whether any existing source files are better matched to the new root rather than an existing root. If so, the file will be put under the new root. That is, the path shown will be relative to the new root rather than the old. This technique can be useful in shortening some relative paths: by setting a new root at a lower level in the directory hierarchy, files switching to the new root will have a shorter relative path.

For example, sample project `"sample_cgicc_diff"` has three separate modules under its `SOURCE_ROOT`: `src`, `demo` and `diff`.

File `CgiUtils` has relative path `SOURCE_ROOT\src\cgicc\CgiUtils.cpp`. If a new root is added as:

```
CGICC_ROOT = ${SOURCE_ROOT}\src\cgicc
```

then, once the **Project Properties** dialog is closed, the files are reallocated. All the files that were under `SOURCE_ROOT\src\cgicc` are now under root `CGICC_ROOT`.

### Note:

Shortening the relative paths cannot be accomplished by simply changing an existing root location. In the example above, `CgiUtils.cpp` initially has the relative path `SOURCE_ROOT\src\cgicc`, which might

convert to the following absolute path:

```
c:\dev\sample_cgicc_diff\src\cgicc.
```

If SOURCE\_ROOT itself is changed to c:\dev\sample\_cgicc\_diff\src, the relative path is unchanged, giving the following incorrect absolute path:

```
c:\dev\ sample_cgicc_diff\src\src\cgicc.
```

This retention of existing relative paths is necessary to find files in the situation where user B needs to adjust the path provided by user A.

## Extraction of Configuration Data

### For Plug-in Users

If you are using Helix QAC Visual Studio or the Helix QAC Eclipse plugins, data extraction will be automatically done when you create your Helix QAC project. This is accomplished in Helix QAC Visual Studio in the following way:

- **Helix QAC** menu : **Admin** : *Create/Sync Project*.

and, in Helix QAC Eclipse by accessing the right-click menu on your project of choice and clicking on the following menu item:

- **Helix QAC** sub-menu : **Convert to Helix QAC Project**.

If you are using either of the Helix QAC Visual Studio or Helix QAC Eclipse interfaces, then skip to Section [Analyzing Your Project](#).

### For C and C++ Users

Helix QAC has various features that help in the extraction of configuration data. The method that you will use will depend on the nature of your build environment. The methods are as follows:

- **Build Process Monitoring**, which monitors your build process for the required information and populates a specified project with it.
- **Compiler Wrapping**, which wraps your compile command during the build process so that Helix QAC can extract necessary information from each command.
- **Manual Extraction**, which allows the user to add a file and its compile dependencies to a specified Helix QAC project.
- **Scripted Extraction**, which is a method based on the manual method, except that it can be re-run automatically whenever necessary.

Each of these methods is described in the following sections.

The most reliable way to extract data from your build environment is to use the **Process Monitoring** feature (refer to [Synchronization via Process Monitoring](#)). The feature is designed to work with many of the compilers and build automation systems on the market. However, there are cases where the feature will not work with a specific compiler or build system, in which case one of the other methods can be used.

The various available methods of data extraction reflect the extremely varied build systems with which Helix QAC is designed to work. Using any particular method does not restrict the way in which you can use a Helix QAC project as any valid project can be used with any of the Helix QAC user interfaces.

## Compiler Wrapping

This feature can be used with build systems that use an environment variable to specify the compiler to be used, for example:

```
CC='gcc-4.7'
CXX='cl.exe'
```

The feature works by assigning the `qaccli` command for adding a file and its dependencies to a Helix QAC project, to the required compiler variable. It applies to any build system that can alias the compilation command, for example a **Make**-based system:

```
make CC="qaccli admin -P <project-path> --compile-command 'gcc -c' --add-file --
ignore-rest"
```

In this situation, instead of your normal compiler receiving the arguments in the build, `qaccli` will receive them directly, and they will be placed after the `--ignore-rest` switch. The `qaccli` command will parse the arguments, search for a source file path, **Include** paths and macro **Definitions**, and, if a source file is found, it will be added to the project along with the other data. As soon as this is done, `qaccli` will use the argument to the `--compile_command` option and issue it to the system, compiling your files as normal.

The `qaccli add-file` feature returns a non-zero value if it fails to add the file. This may not be desirable in all circumstances as a build will normally exit if a tool returns a non-zero value. To prevent the build from exiting on **make**-based systems, a dash can be added in front of the command:

```
make CXX='-qaccli admin -P <project-path> -z g++-4.7 --add-file --' / CC='-qaccli
admin -P <project-path> -z gcc-4.7 --add-file --'
```

### Note:

`--` is a shortcut for `--ignore-rest`.

`-z <command>` is a shortcut for `--compile-command <command>`.

The `-z g++-4.7` part of the command tells Helix QAC to compile each file after extracting it together with its compile-dependent information. This is necessary for systems that build artefacts needed later in the build process. If your compile command contains spaces (for example, `g++ -Wall`) or other characters that may confuse a shell, then place the compile command in a script or batch file and make it executable.

For example, you could use a file named `compiler_cpp.sh` to wrap the compile command as follows:

```
#!/bin/bash
```

```
g++ -Wall "$@"
```

Given that a similar file for the C compiler could be created, the **make** command from the example above could now resemble the following:

```
make CXX='qaccli admin -P <project-path> -z compiler_cpp.sh --add-file --' /  
CC='qaccli admin -P <project-path> -z compiler_c.sh --add-file --'
```

The **Compiler Wrapping** method can be used in any build system that allows the substitution of the compile command. As with **Build Process Monitoring**, ensure that you clean the project before building.

**Note:**

To suppress Include paths when files are being added during the Compiler Wrapping, refer to [Suppress Include Paths](#).

## Manual Extraction

The `--add-file` option for the `qaccli admin` command is very versatile. For example, the following command will add the file `file.c` to a Helix QAC project along with its compile dependent information, the include path and the definition<sup>1</sup>:

```
qaccli admin -P C:\path\to\QAFproj --add-file -- /IC:\workspace\inc\DMYMACRO=1  
C:\workspace\src\file.c
```

The `--add-file` option will parse all data that follows it and extract any compile dependent information within that data.

Another way of using the `--add-file` feature is to use the operating systems tools to automatically add all files in a given directory structure. On Unix-like systems, this is quite simple:

```
find . -iname "*.c" | xargs -n qaccli admin -P <project-location> --add-file
```

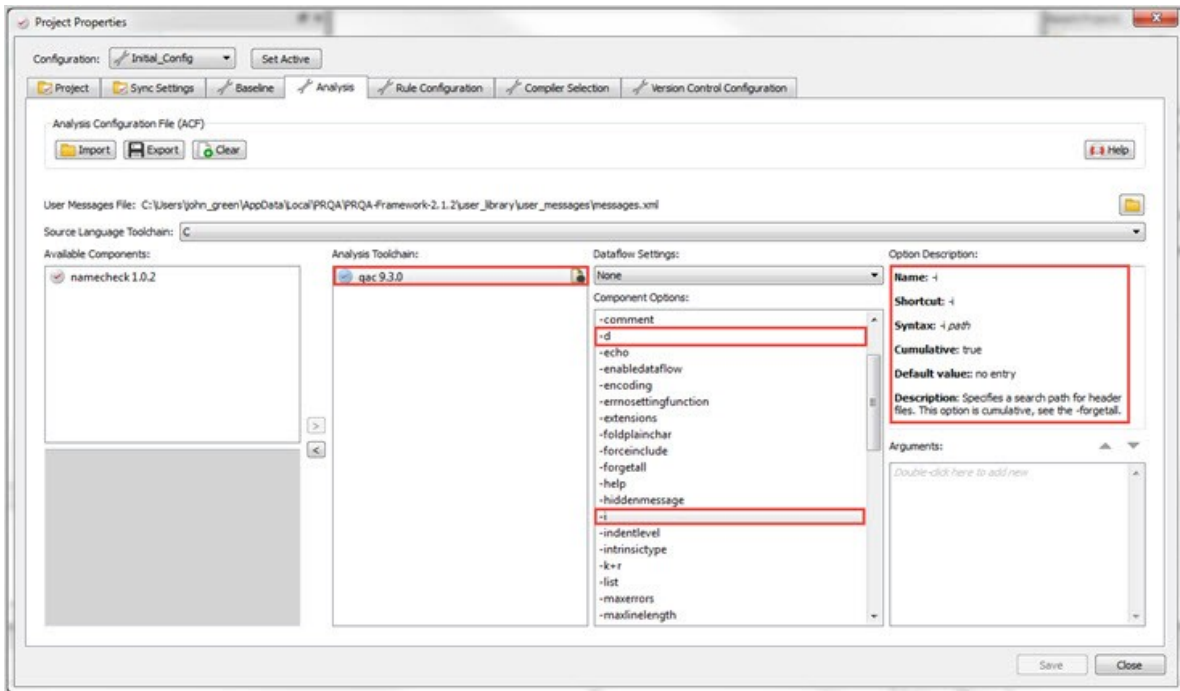
will add all files with a `.c` extension in the directory tree rooted at the current directory. The equivalent on Windows operating systems is as follows:

```
for /f %f in ('dir /s /b *.c') do qaccli admin -P <project-location> --add-file -- %f
```

This approach will not add any of the compile-dependent data for those files. This can be done manually by opening the QA Project Properties user interface for the Helix QAC project in question and selecting the analysis component that you wish to use to add the information. For the C language, select the C Source Language toolchain and qac analysis component, as shown [on the facing page](#).

---

<sup>1</sup>A space is optional between the `I` and the subsequent path. The feature will work with forms of include and define specifiers and formats for most compilers.



Likewise for the C++ language, choose the qacpp analysis component.

**Include Paths** and **Definitions** can be added using the following options, respectively:

- **Analysis** tab : select-analysis-component : -i
- **Analysis** tab : select-analysis-component : -d

This will add **Include Paths** and **Definitions** for all files for a specific source language.

Adding **Include Paths** and **Definitions** for a specific source file can also be done through QA GUI by selecting the source file in the file/folder tree and editing the properties at the bottom of the **Files** view.

Once you click the **Edit** button on the right of one of the highlighted fields, a text window opens up so that you can specify the particular path or definition.

## Scripted Extraction

The information can sometimes be extracted from a project file using a script and subsequently added to a Helix QAC project using the `--add-file` option for `qaccli`.

An example of how to do this with a Freescale CodeWarrior project file<sup>1</sup> using a simple Python script<sup>2</sup> can be found in the following directory:

```
<Helix QAC Install Directory>/common/doc-en_US/samples/data_extraction/
```

<sup>1</sup>The Freescale CodeWarrior project format is not text-based, but an XML representation of the file can be exported. This is what is used for the example.

<sup>2</sup>The example is written in Python, although any scripting language can be used.

## Analyzing Your Project

All three graphical user interfaces, Helix QAC Visual Studio, Helix QAC Eclipse and QA-GUI, have an **Analyze** menu or sub-menu. There are two types of analysis available: File-based analysis and RCMA<sup>1</sup> (Cross-Module Analysis). These two types of analysis are only related through the fact that file-based analysis results must be available in order for the project based RCMA to work. RCMA is described in [Cross-Module Analysis \(CMA\)](#).

File-based analysis is run on files independently. As such, it can be run on one file, a group of files, or all files in the project. Each file is analyzed with all analysis components in the toolchain for the specific source language of the file. Refer to Section [Analysis Configuration](#) for more information.

## QA-GUI

File-based analysis can be invoked on a specific file, files or the entire project using the following sub-menu:

- **Analyze** menu : **File-Based Analysis**

## Analysis Dialog Box

The **Analysis** dialog box is shown when Helix QAC is performing an analysis.

During an analysis, the dialog box will display the process and give you two options: **Stop** or **Abort**.

The **Stop** command allows Helix QAC to finish processing the current tasks. However, no further source files are analyzed.

The **Abort** command immediately quits all current analysis and no further source files are analyzed. This command is useful if you require a quick exit or if the **Stop** command is taking too long to complete.

## Helix QAC Eclipse

File-based analysis can be invoked on a specific file, a specific group of files, or the entire project, using the following sub-menu:

- **Helix QAC** menu : **Analyze** sub-menu : **File-Based Analysis**

## Helix QAC Visual Studio

File-based analysis can be invoked on a specific file, files or the entire project using the following sub-menu:

- **Helix QAC** menu : **Analyze** sub-menu : **File-Based Analysis**

To perform file-based analysis on one file or a subset of files, right-click on the selected file(s) and run File-based analysis.

## QA-CLI

To clean a project of analysis results and run file-based analysis, issue the following command:

```
qacli analyze -cf -P <project-path>
```

To do all of those actions for a subset of files within the project, use the -F option to provide a file containing a list of files with one entry per line:

---

<sup>1</sup>The "R" is for Relational and differentiates this version from the previous, memory-based implementation of CMA.



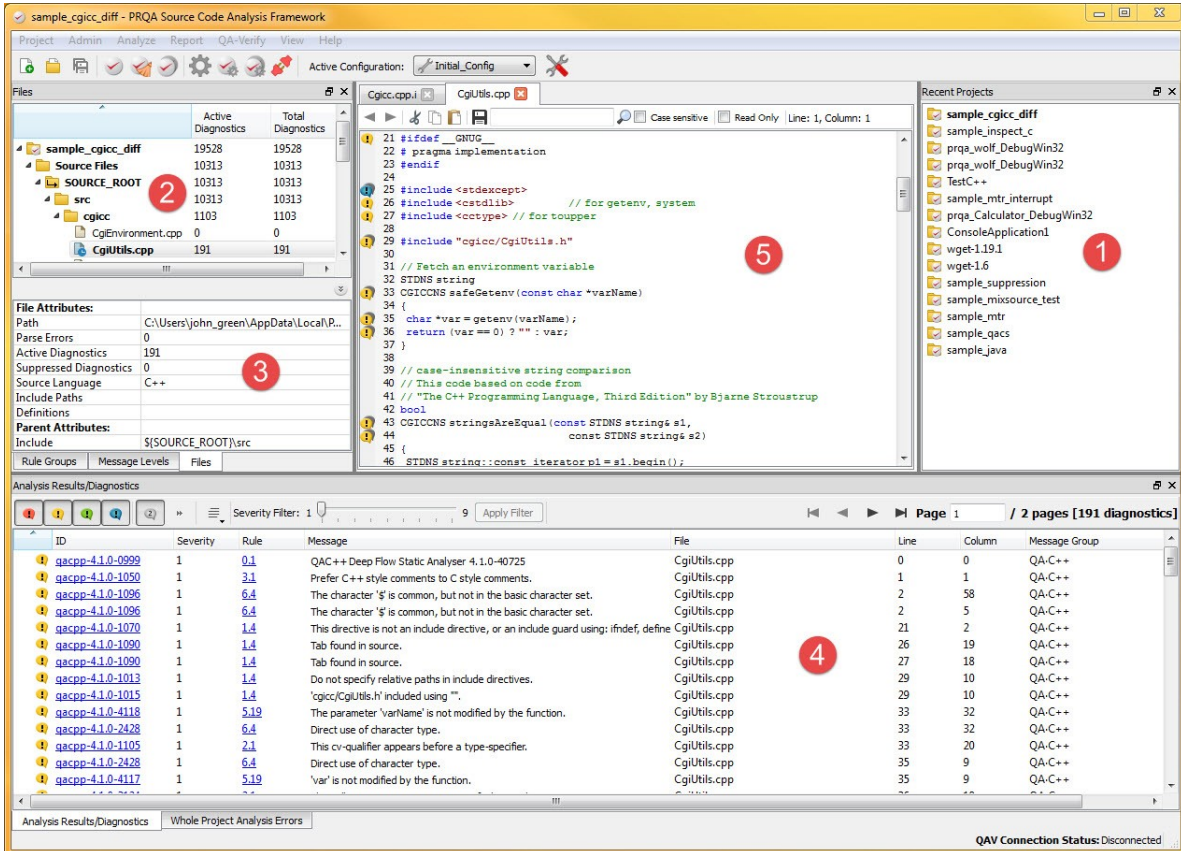
```
qaccli analyze -cf -P <project-path> -F <file-with-list>
```

This page Intentionally left blank

This page Intentionally left blank

## 3 | GUI Overview

This section describes the functionality that is available directly from the GUI. The various GUI panels are shown [below](#)



The panels are as follows:

1. **Recent Projects** available for opening.
2. The **Files** panel for the active project. You can switch from this panel to either the **Rule Groups** panel or the **Message Levels** panel by using the associated tabs. The GUI always opens at the last tab that was selected.
3. The attributes for the item that is currently selected in the above panel. These can be hidden using the double arrow button on the top right.
4. The **Analysis Results/Diagnostics** panel showing the filtered results for the file(s) selected in the **Files** panel.
5. The **Editor** that allows you to view the highlighted diagnostics within the source code, and modify the code.

**Note:**

The **File Attributes** are intended to be specific to a particular file, and the **Parent Attributes** are intended to be 'global' to the project, with the **File Attributes** taking priority. However, you should be aware that the **Parent Attributes** are 'read only' and may not appear for certain projects once they have been re-synchronized.

## GUI Features

The features covered include the most common actions that can be taken using the above panels, for example:

- Opening a project
- Selecting files for editing
- Viewing diagnostic messages in-line
- Viewing and filtering the diagnostics
- Viewing rules violated in selected files
- Opening message help content
- Opening rule help content

## Opening a Project (from the CLI)

It is possible to start QAGUI and automatically open a project by passing the projects path on the command line as an option. For example:

```
qagui -P /home/steve/projects/helloworld
```

The paths can be absolute or relative to the current working directory and should point to the directory containing the `prqaproject.xml` that you want to open.

## Opening a Project

A newly created project appears at the top of the **Recent Projects** panel on the right of the interface. When you hover over an entry in the panel, a tooltip appears showing the path to the project directory.

A recent project may be re-opened by double-clicking the name in the panel, or by rightclicking the project and selecting **Reopen This Project**. When you open a project if there is already an active one displayed in the current view, you are asked to confirm that you wish to change the active project.

The active project is indicated in bold type.

**Note:**

When a new project is opened, select the **Files** tab at the bottom of the upper left hand panel in order to select the files to be analyzed. Then proceed as in Section [Analyzing Your Project](#). The **Rule Groups**, **Message Levels** and **Analysis Results/Diagnostics** panels are not populated until files have been analyzed.

## Viewing the Diagnostics

Once a project has been analyzed, there are three possible panels within which to view the resultant diagnostics:

- **Files** panel
- **Message Levels** panel
- **Rule Groups** panel

The current panel can be changed using the associated tabs at the bottom of the panels.

## Files Panel

When a file is selected, its name is highlighted in bold within the panel and the source code is displayed in the **Editor**. Diagnostic results are displayed in the **Analysis Results/Diagnostics** panel.

Each of the analyzed source files and header files in the project is displayed with a corresponding icon, which indicates whether analysis results are up to date (indicated by a green icon with a tick), out of date (indicated by a blue icon with a clock), or whether there is a parsing error (indicated by a red warning icon). Hovering over an 'out-of-date' file in the Files Panel displays a tooltip describing what analysis needs to be done and why.

The **Active** and **Total** column counts show the current and overall numbers of diagnostic messages found in each file, respectively. The **Active** count can differ from the **Total** count for either of the following reasons:

- There are suppressed messages.
- A baseline has been specified via baseline suppression (refer to [Baseline Diagnostics Suppression](#)).

You can view the results from more than one file by using the following selection methods:

- CTRL-Left\_Click: select or deselect individual files or directories
- SHIFT-Left\_Click: select a range of files/directories.
- Left\_Click: select just one file/directory (and deselect all others).

Selecting a directory will automatically select all files within that directory and its subdirectories.

Deselecting a directory will deselect all files within that directory and its sub-directories.

It is the group of selected files that dictates the content of the **Rule Groups** and **Message Levels** panels. In order to migrate quickly to "parsing errors":

1. Select the files of interest - this may be the top-level directories if the files are unknown.
2. In the **Message Levels** tab, collapse the **Warnings** category - this brings the Errors category into view.
3. Right-click on the required error message and select **Show Only**.
4. The affected files are listed in the **Analysis Results/Diagnostics** panel.

Each selected file, or group of selected files, has an associated context menu, selected by right-clicking. The available options are as follows:

- **Analyze Selected File(s)**
- **Clean Selected Files(s)**
- **Open Preprocessed** <file name>
- **Remove**


- **Generate Report For Selected File(s)**
- **Upload Selected Files To Dashboard**
- **Upload Selected Files To Structure 101**

The context menu for a directory displays the following additional items:

- **Expand All Below**
- **Collapse All Below**
- **Add File(s)**

Tools such as RCMA can generate messages that do not belong in a single location in a particular source file. To display these messages there is a tree node named "CMA" (Cross-Module Analysis) in the **Files** panel. When the "CMA" node is selected, the messages are shown in the **Analysis Results/Diagnostics** panel (refer to [Viewing and Filtering the Diagnostics](#)).

The **Raw Sources** node displays any manually added raw source files, that is to say, preprocessed source files ending in .i. These are designed to be used for temporary analysis only, and are not maintained as part of the project. To add a raw source file:

1. Select **Analyze : Open Analysis Settings** in the GUI and ensure that **Generate Preprocessed Source for Analyzed Files** is checked.
2. Select **Analyze : Analyze Raw Source File**, and complete the resultant dialog as in the remaining steps.
3. Select the language of the file that you wish to analyze.
4. Click  and, navigate to `prqa/configs/<config_name>/output/_SOURCE_ROOT` to select your .i file.
5. Click **OK** to analyze the file.
6. The **Raw Sources** node appears.

There are two context menu options available for raw source files.

- **Raw Source Analysis:** This performs analysis on the selected raw source files.
- **Clean/Remove Selected File(s):** This cleans the results of the selected raw source files and removes the files from the project tree.

At the bottom of the Files panel is the collapsible summary table. It gives summary diagnostics for a single selected file.

**Note:**

It contains fields for the **Include Paths** and **Definitions** used by the file. Once you click the Edit button on the right of one of the highlighted fields, a text window opens up so that you can specify the particular path or definition.

At the bottom of the summary table, there's a **Parent Attributes** section. This section dynamically populates with fields relating to inherited data from higher level folders. In the same way, folders have their own summaries which can be edited.

The **Files** panel has two viewing modes that can be switched between via the sub-menu:

- **Helix QAC menu : Admin sub-menu : Manage GUI Settings**

This action can also be performed from any context menu invoked on the panel. The first mode separates source and header files within the tree, placing all header files inside the **Includes** folder; the second mode will place the header files in the existing source tree structure. Header files located externally to the source structure, or the project root directories, are situated within the **External Includes** folder.

## Message Levels Panel

The **Message Levels** panel displays the messages for the selected file(s) in a tree view, sorted by group, listed in numerical order. The counts at the top of the panel are those for the selected files only. Messages are grouped under the Analysis Component that produces them, for example "qac", "qacpp", or a Secondary Analysis component, for example "m3cm".

The **Active** and **Total** column counts show the totals of each diagnostic message found in the selected files. When a message is selected, the summary table at the bottom of the panel shows the **Rule Group** mapping and the rules enforced by the message.

## Rule Groups Panel

The **Rule Groups** panel opens with the name of the **Rule Configuration** group(s) to which the messages belong. For a C and C++ project, separate groups for each language are displayed.

When a message group is expanded, it shows a hierarchy of sub-groups, allowing you to drill down to the set of messages that enforce a particular rule.

The **Active** count can differ from the **Total** count for either of the following reasons:

- There are suppressed messages.
- A baseline has been specified via baseline suppression (refer to [Baseline Diagnostics Suppression](#)).

### Note:

Each level of the **Rule Group** tab sums all its child levels, but only counts unique diagnostic messages; this means that if a single message violates, for example, rules 2-4 and 2-2, the summation level above those two rules will count the offending message only once.

The summary table at the bottom of the panel shows the summary diagnostics for a single selected rule group or a single selected message.

The context menu for any message or folder in the **Message Levels** or **Rule Groups** panel allows you to create a diagnostics display filter. You can select **Show Only**, so that only the diagnostics for that message or folder are displayed in the **Analysis Results/Diagnostics** panel, **Hide** so that the diagnostics for that message or folder are not displayed, or **Show** to redisplay diagnostics that are currently hidden. By a combination of these options, you could, for example, choose to show all instances of a single message for a particular component and Rule Group.

### Note:

Right-click again and select **Reset Filter** on any item in the **Message Levels** or **Rule Groups** panel to remove the filters and redisplay all diagnostics.

## Viewing Diagnostic Messages In-Line

The source code for the selected file is displayed in the **Editor** in the center of the interface. Any diagnostic that is selected in the **Analysis Results/Diagnostics** panel is highlighted.

The diagnostics are marked in the **Editor** via icons on the left. These are color coded according to the diagnostic on the particular code line with the highest severity - red for Error messages, yellow for Warning messages, green for User messages, and blue for Information messages.

**Note:** Information messages can be traced to their corresponding diagnostics within the editor.

When an icon is selected, a separate text box displays all associated diagnostic messages, with access to the Help.

Suppressed diagnostics which have been made visible will also be shown in the popup. The suppression type is displayed near the start of the message and the whole line of text is color coded:

- Comment suppression (orange)
- Pragma suppression (green)
- Dashboard suppression (magenta)
- Baseline suppression (violet)
- Multiple of the above (red)

The Editor may be used to make quick changes to the source code for immediate reanalysis. The internal toolbar contains the typical editor features of Cut, Copy, Paste, Save, Refresh and Search. Searches can be case-sensitive, and key combinations CTRL-F and CTRL-B can be used for forwards and backwards searching. Each tab has a context menu available, on right-click, for the following file options:

- **Close**
- **Close all**
- **Close all but this**
- **Save**
- **Open containing folder**
- **Copy file path**

Information:

The context menus have a state dependent menu item which will appear if:

**Analyze:** The file has been saved but has out of date or no analysis results

**Clean File:** Always available.

**Save and Analyze File:** The file has been modified in the editor.

Use key combinations CTRL-Tab and CTRL-SHIFT-Tab to cycle forwards and backwards through the open files. CTRL-W closes the current tab.

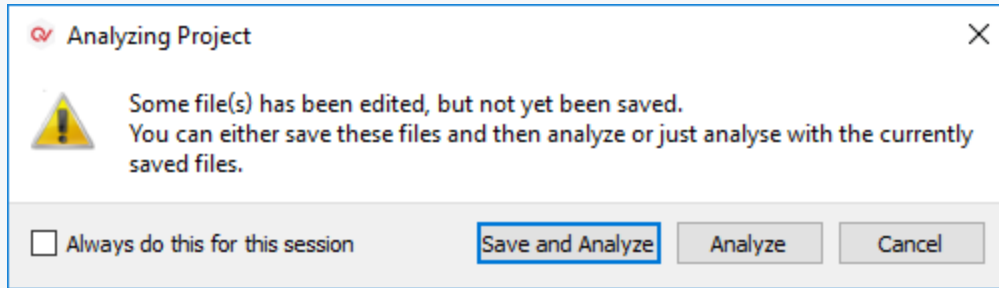
**Note:**



If you wish to add suppressions to the source code, refer to the individual Helix QAC for C and Helix QAC for C++ component manuals.

The code editor respects Tab characters in the source code and pads them out to the appropriate column. This value is specified as a parser option and can be modified by changing the parameter `-tabstop`. If no value is specified then a default value of 8 will be used.

If there are files that have been edited but not saved then before the analysis has been run the following dialogue is displayed



If **Always do this for this session** is selected then, the dialog will not be displayed the next time an analysis is done with an unsaved file. This setting is reset when the user exits Helix QAC and re-starts.

## Viewing and Filtering the Diagnostics

The Analysis Results/Diagnostics panel, which is selectable via the tab of the same name at the bottom of the screen, is populated when files are selected in the Files panel. It shows all the diagnostic messages, and sub-messages, found in each file. The data may be sorted by clicking on any column header (with the exception of the "Message" column).

The diagnostics are displayed in pages of results, with 100 diagnostics per page by default (the number of pages and current page number are shown at the top right of the panel). This default can be modified by selecting **Analyze : Open Analysis Settings** in the GUI and changing the **Max. number of diagnostics per page** within the **Analysis Settings** dialog box. For more information on this operation, refer to Section [The Analyze Menu](#)

There are three methods for changing the displayed page of results:

1. Entering the required page number.
2. Using the icon controls (Go to: First Page, Previous Page, Next Page, Last Page).
3. Using the following shortcut key combinations:
  - **First Page:** Alt Key + Up Arrow Key
  - **Previous Page:** Alt Key + Left Arrow Key
  - **Next Page:** Alt Key + Right Arrow Key
  - **Last Page:** Alt Key + Down Arrow Key.

Each message is categorized with an icon indicating an:

- Error message (red)
- Warning message (yellow)
- User message (green)
- Information message (blue)

To select a particular category of message for display, select the relevant colored icon in the top left hand corner of the **Analysis Results/Diagnostics** panel.

Similarly to how suppressed diagnostics are presented in the **Editor**, refer to Section [Viewing Diagnostic Messages In-Line](#), the **Analysis Results/Diagnostics** panel displays suppressed diagnostics using the same color convention: orange for comment suppression, magenta for Dashboard suppression, green for pragma suppression, violet for baseline suppression and red for diagnostics that are suppressed using multiple methods.

Clicking the sub-messages icon allows you to expand the messages that have further messages linked to them (indicating, for example, that an error may have originated in a different part of the code).

Within each category, each diagnostic is broken down into the following columns in the panel:

- **ID** (defined by component-version-ID)
- **Severity** (a Severity level of 0 is the least severe and a level of 9 is the most severe)
- **Rule(s)** that it enforces - if there is more than one, a context menu is available.
- **Message** text
- **File**
- **Line** number on which the error occurs
- **Column** number
- **Message Group** to which it belongs (that is to say, the **Rule Configuration**).

You can rearrange the columns by clicking on the column titles and dragging them to their new desired locations within the top row of the table.

You can also filter the diagnostic messages in accordance with the **Severity** level. If the **Severity** level filter is set at "n", then only messages with level "n" and above will be displayed.

For a particular diagnostic message, the subsequent action depends on how the message is selected:

- Clicking on the Message ID opens the associated message in the Helix QAC Message help window.
- Clicking on the Rule opens the associated rule in the Helix QAC Rule help window.
- Double-clicking on either the colored indicator on the left, or the Message text, opens the source file at the associated line in the Editor.

## Whole Project Analysis Hard Errors Panel

This panel, which is selected by clicking the **Whole Project Analysis Hard Errors** tab next to the **Analysis Results/Diagnostics** tab, displays any hard parser errors for the project, broken down by TU (Translation Unit), even if the errors were the result of the lack of an appropriate license. As with the **Analysis Results/Diagnostics** panel, you can click on the column headings to sort the errors. Any diagnostic that is marked as a hard error cannot be suppressed, it represents the most severe form of error and leaving this unresolved may result in spurious diagnostics and/or missed subsequent diagnostics in the TU.

The TUs are expanded one level by default to show the errors. There are also options on the context menu for any of the items shown that allow you to expand and collapse the errors as required: **Expand All Below** (to expand all the levels below the current item), **Collapse All Below** (to collapse all the levels below the current item), **Expand All Files** (to expand all the levels for all items to the default single error level), and **Collapse All Files** (to collapse all the levels for all items).

## Menu-based Operations

### Note:

Not all menu items apply to all programming languages.

## The Project Menu

### ■ Create New Project

This opens the New Project window, where you browse to the directory of the project source code and select initial configuration files. See Section [Creating a Helix QAC Project](#) for details.

The Helix QAC software stores information about a project by adding two files ("prqaproject.xml" and "prqaproject.xml.stamp") and a directory ("prqa") to the project directory. You should not modify these.

### Note:

You are warned if the directory already contains a "prqaproject.xml" file.

### ■ Open Project

Enables you to open an existing project by either selecting a "prqaproject.xml" file or a project directory. An error is displayed if the project directory does not contain a "prqaproject.xml" file.

### ■ Close Project

Closes the active project and clears the screen.

### ■ Close and Delete Project

Closes and deletes the active project, clearing the screen.

### ■ Delete Project

Launches a dialog that allows the user to choose a project to delete.

### ■ Synchronize

Opens a dialog that allows a project to be built in its own environment and monitored for the information required for Helix QAC analysis. For C and C++, the Helix QAC software derives the source code contents of a project from observations of the build process. You specify a script, having first read the list of constraints to ensure that the script adheres to them. Clicking Synchronize causes the Helix QAC software to execute the script and monitor the subsequent build process, identifying the source code files of interest.

### Note:

Because the Helix QAC software derives its own information from observations of the build process, no information will be gathered if the process does not build something (for example, because the code project has been recently built and nothing needs to be recompiled). It is therefore recommended that you clean the build first, or else have the scripted build processes clean it for you.

On successful completion of this synchronization process, the Files panel is populated with the source and header files found, together with any definitions needed.

#### ■ **Open Project Properties**

This opens the Project Properties window, where the following configuration functions may be performed:

- Set the Project and Source Root Directories
- Add file extension mappings for the programming language
- Perform Synchronization Configuration
- Set Baseline Diagnostic Suppression
- Perform Analysis Configuration
- Perform Rule Configuration
- Select Compiler Compatibility templates
- Perform Version Control Configuration.

#### ■ **Open Configuration Manager**

This opens the Configuration Manager dialog. Refer to [Configuration Manager](#)

#### ■ **Open CMA Project Editor**

This opens the CMA Project Editor window, allowing the configuration of CMA.

#### ■ **Upload Results to a Structure101 project**

This option opens a sub-menu with the following options:

- **Upload project to Structure101**
- **Upload selected files to Structure101**

#### **Note:**

This feature needs to be included in your license in order for you to be able to upload files to Structure 101.

- **Exit Application**

## **The Admin Menu**

#### ■ **Manage License Servers**

Opens the **License Server Management** dialog, where you can specify the host addresses and ports of the available license servers. The active server is the one at the top of the dialog.

## ■ Set Language

Sets the preferred language of this installation of Helix QAC. Selecting a different language (for example, Japanese) sets the GUI localization strings to those of the selected language. A restart of the GUI is then required.

By default, this is set to 'Auto' based on the machine locale and the availability of translations. If there are no suitable translations, the language is set to `en_US`.

## ■ Set Logging Level

Controls whether log files are written, and how much detail is written to them. The recommended logging level is "error". Log files are written to the `<local> \Perforce\<version>\app\logs` directory, where `<local>` is your local application data directory, and `<version>` is the Helix QAC version.

## ■ Import Configuration

Allows you to import settings from the same or previous versions of Helix QAC. When migrating from a previous version of Helix QAC to this newer version, it allows you to keep your original settings (for example, screen layout, license server settings, and so on). The option can also be used to help you preserve your settings when moving between installations on different machines.

## ■ Set User Data Location

By default, the location of the user data area is dependent on your desktop environment, that is to say:

- **Windows:** `%LOCALAPPDATA%\Perforce\Helix-QAC-2021.1`
- **Linux:** `$HOME/.config/Perforce/Helix-QAC-2021.1`

Use this option to select a new location where this data should be held, allowing, for example, the user data location to be a shared one.

**Note:** It is recommended that you immediately restart the particular application or plug-in (QA-GUI, QA Visual Studio or QA Eclipse) whenever this option is selected.

The target directory must exist and you must have **Write** permissions for it.

Initially, no files are copied from the previous user location. When the application is restarted the default files from the installation directory will be copied. No user-created files from the previous location are copied - these must be copied manually if needed.

## ■ Import CCT

Lets you add the provided CCT to the user data area so that the CCT can be used in projects. In addition, you can optionally import the CCT to the installation folder.

**Note:**

If an existing CCT is updated using this menu, all required files under the DATA folder must exist. This is to prevent the existing CCT from being removed before the import is done.

## ■ Manage GUI Settings

Opens the GUI Settings dialog, where various display options along with file list and font options can be modified.

The "**Panels**" allows the following to be modified:

- **Hide Source Files with 0 Diagnostics** If this is checked, scanned source files that have no diagnostics will be hidden from the file list.
- **Hide Header Files with 0 Diagnostics** If this is checked, header files that have no diagnostics will be hidden from the file list.
- **Embed Headers** is described in Section [Files Panel](#)
- **Sort Files in a Lexicographical Order** If this is checked, items in the file list are sorted using the legacy ordering. When unchecked, items are tokenized and ordered numerically, in alignment with Helix QAC Dashboard.
- **Max. Number of Diagnostics** per page sets the maximum number of diagnostics to be shown per page of the **Analysis Results/Diagnostics** panel.

**Note:** The default value is set at 100, although setting a lower value may result in improved performance for computers with limited resources.

The "**Font and Color**" page allows the user to change the font properties such as font family, size, weight and color. By default, the font used is a black 9pt mono-spaced font.

The "**Analysis Feedback**" page contains the following options:

- **Display results while analysis is underway** If this is checked, the user can navigate through the project and inspect results during an analysis.
- **Analyze after save a file** If this is checked, analysis will be performed on any files that are saved in Helix QAC's editor.
- **Save and analyze following inactivity** If this is checked, when files are modified in Helix QAC's editor they are automatically saved and analyzed if the user makes no changes after a period of time
- **Perform a quick analysis when automatic re-analysis may be slow** If checked, 'light' analysis settings are used for files that will be slow to automatically analyze. Manual analysis would need to be performed at the user's convenience when full fidelity is required.

## The Analyze Menu

For C and C++, the source files can be analyzed either individually or in a group. For C and C++, there are two kinds of analysis available: "file-based" and "CMA". "File-based" analysis operates at the translation unit level and checks for issues such as correct language use, dataflow and layout. CMA analysis operates across translation unit boundaries and checks for issues such as duplicate definitions, incompatible declarations and unused variables (refer to [Cross-Module Analysis \(CMA\)](#)).

For other languages, analysis operates on all the files in the project.

The menu options are as follows:

- **File-Based Analysis of project**  
Analyzes the currently selected files.
- **Clean project**  
Removes the diagnostics, and any manually added (.i) raw source files, from the project.

### ■ Run CMA Analysis

Performs CMA Analysis on CMA projects, which are created using Open CMA Project Editor, located below the Project menu.

### ■ Analyze Raw Source File

This option is primarily for analyzing pre-processed (.i) C and C++ files. Refer to Section [Files Panel](#).

### ■ Open Analysis Settings

This feature is intended to help control the generation of analysis information to aid debugging. the following options are available in the displayed dialog.

- **Stop Analysis Upon Failure** If this is checked, analysis stops as soon as an error is encountered in any of the files.
- **Generate Preprocessed Source for Analyzed Files** If this is checked, then .i files are produced ready for **Raw Source Analysis** (these are standalone files containing any "#include" headers).
- **Assemble Support Analytics for Failed Files** produces a .zip file containing the .met (or .arc), .i and .via files for the analyzed file(s) in the event of a parser failure.
- **Max. Number of Analysis threads** is the number of CPUs that can be allocated towards the Analysis task.
- **Number of times to re-try after failure** sets how many times an analysis is repeated when a license error occurs

## The Report Menu

- **Generate report for project**
- **Generate report for selected files**

Once you select one of the above options, select the type of report to be generated on the **Report Generation** dialog


Refer to Section [Standard Report Types](#) for the types of report that can be generated, the location of the reports (the reports are automatically opened at this location if you leave **Open Report Location** checked in the dialog), and the meaning of 'Ignore Dependencies'. Click **OK** to generate the selected report.

## The Helix QAC Dashboard Menu

This allows interaction with Helix QAC Dashboard, which is a centralized application that manipulates, summarizes, and gives wide visibility to, the results generated using Helix QAC.

The options are as follows:

### ■ Connect to Helix QAC Dashboard Server

You will note that, in the bottom right hand corner of the screen, the **Connection Status** to Helix QAC Dashboard is displayed. By default, it is **Disconnected**. You can connect to the required host by either selecting this option or clicking the  icon on the toolbar, and then specifying the host and username (which you can allow Helix QAC to remember for next time), and password. Once Helix QAC Dashboard is connected, this is confirmed in the bottom right hand corner of the screen. Hovering over the connection will show a tooltip confirming the user that you are connected as.

## ■ Unify Project

This option allows projects to be shared, and enables the downloading of suppressions, and snapshots to be used as baselines, from Helix QAC Dashboard. Uploading a new definition creates a project in Helix QAC Dashboard, which, in conjunction with the Helix QAC project, forms a "Unified Project". The definitions for existing "Unified Projects" can also be updated.

### Note:

If you are using a "Unified Project", then you can only use a single configuration (refer to [Restrictions](#) in Section [Projects with Multiple Configurations](#)).

## ■ Upload Results

This option allows you to upload either an entire, analyzed project to Helix QAC Dashboard, or else a selection of analyzed files. Assuming that you are connected to the corresponding version of Helix QAC Dashboard, select either Upload Results : **Upload Project <project> to Dashboard** or **Upload Results : Upload Selected Files to Dashboard**. The **Upload Results** dialog is displayed.

The **Upload Results** dialog is used to populate values that will be used when uploading results to Helix QAC Dashboard. At a minimum you must enter a Project Name and this will be used to identify the upload on Helix QAC Dashboard. The following optional values may also be specified:

- **Snapshot Name** : The name of the snapshot. If not specified then a name based upon the date & time will be used.
- **Snapshot Parent** : The name of the parent snapshot, to which the new snapshot will be directly connected. If this entry is not provided then the last uploaded snapshot is chosen as parent.

You can also select how and if source files will be uploaded - next to Upload Source, select **None**, **All**, or **Only not in VCS** to upload only the source that is not yet in the Version Control System set up under the Version Control Configuration tab in Project Properties. You can also determine the Upload Path - hover over the settings to see the help, before selecting **Root**, **Relative** or **Absolute**. Expanding Advanced Settings reveals optional version control settings that can be appended to the upload, and an option to set the number of threads created when uploading to Helix QAC Dashboard. Repository Path specifies the source code repository of the Helix QAC Dashboard project and VCS File will upload a local VCF.

To commence the upload, click **OK**. Once the resultant progress bar indicates completion of the upload, open the host and you will see the project or files in Helix QAC Dashboard under the new Project Name that you supplied.

## ■ Download Unified Project

This option allows you to create a new Helix QAC project as part of a "Unified Project". It can also be used to update an existing "Unified Project".

## ■ Download Baseline

This option downloads a snapshot and the corresponding source code, which together are used to generate baseline diagnostic suppressions.

## ■ Download Suppressions

This option downloads suppressions that have been defined within Helix QAC Dashboard. These suppressions are applied against diagnostics in the local source code.



## The View Menu

This menu provides you with the options to turn off and on (by unchecking and checking) any of the following on the Helix QAC GUI:

- **Analysis Results/Diagnostics**
- **Whole Project Analysis Errors**
- **Message Levels**
- **Rule Groups**
- **Files**
- **Recent Projects**
- **Toolbar**

The following options allow you to change the layout back to its default layout, and close all the Editor tabs, respectively:

- **Reset Layout**
- **Close All tabs**

The following option redisplay the initial page that provides access to some Help on associated components:

- **Welcome Page**

## The Help Menu

This menu provides access to the following online documentation:


- **Helix QAC Help**
- **Rules** (the default configuration)
- Individual component manuals (dependent on installation)
- The **Quick Start** guide for Helix QAC GUI
- **About** (that is to say, the details of your version of Helix QAC and of each of the associated components).

This page Intentionally left blank

This page Intentionally left blank

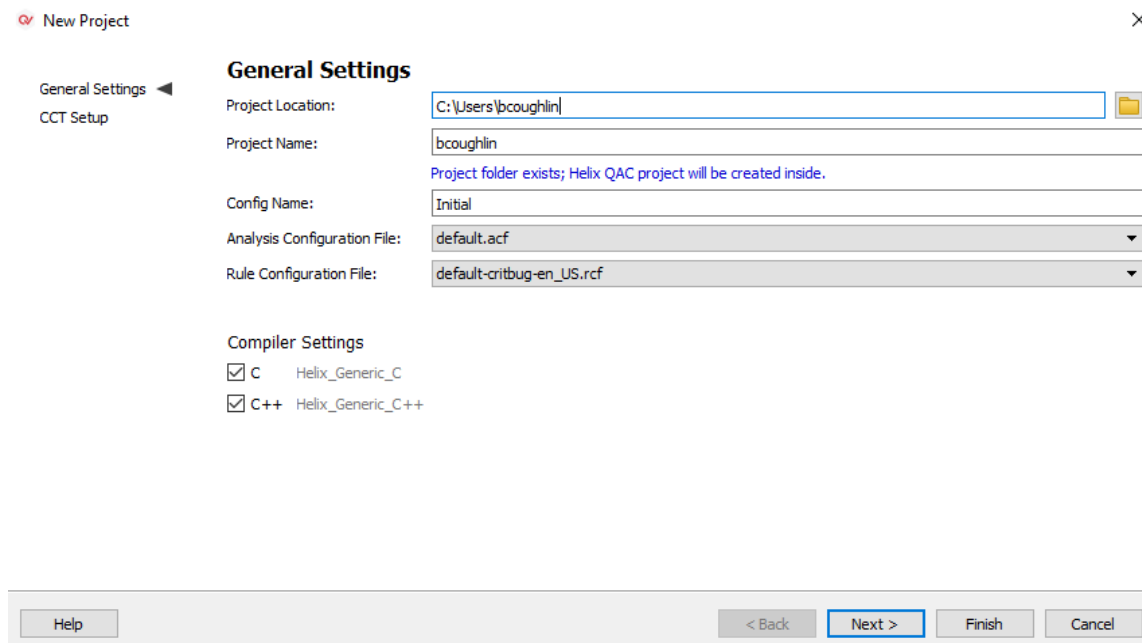
## 4 | Projects

### Using the Project Creation Wizard

You can create a new Helix QAC project by clicking  on the toolbar, selecting **Project** :

**Create New Project**, or else typing **Ctrl + N**.

The first page of the **New Project wizard** is displayed, as shown [below](#).



The side panel on the left tracks the progress through the wizard, the items are click-able for navigation.


After a new project is successfully created, the **New Project Wizard** will store the entered values for ease of use when creating multiple projects.

The following sections describe each field or panel, and the information that needs to be provided:

### Page one: General Settings

#### Project Location

The first page of the New Project wizard is displayed as shown [above](#) .

Click  and browse to the folder containing the Helix QAC project. The path to the folder then becomes the **Project Location**.

For example, if a Helix QAC project was created in the following directory:

C:\Users\CodeProjects\Project\_Alpha\

then the **Project Location** would contain the same path.

**Note:**

In order for the path to be valid, you must have the requisite permissions, and it must comply with the restrictions on naming (refer to [Filenames](#) in the Appendices).

If you select a **Project Location** for which there is no folder that matches, Helix QAC will attempt to create a new project folder for you.

## Project Name

The **Project Name** is the human-readable name for the project as displayed in the **Recent Projects** list. It defaults to the folder containing the Helix QAC project, and so, in the example above, it would default to "Project\_Alpha". However, you can change it to be anything you like, provided that it complies with the restrictions on naming (refer to the Appendix [Filenames](#)).

## Config Name

The **Config Name** is the name for the first configuration in a project. By default it will have the name "Initial". However, you can change it to be anything you like, provided that it complies with the restrictions on naming (refer to the Appendix [Filenames](#)). After project creation it is also possible to rename a configuration.

## Analysis Configuration File

You can select the **ACF (Analysis Configuration File)** to be associated with the project, to determine the toolchain of analysis components to be used. There is a default **ACF** available, and other **ACF** files to address specific requirements can be written by yourself, with guidance from Programming Research Ltd. as required.

## Rule Configuration File

You can also select the **RCF (Rule Configuration File)** to be associated with the project. The following **RCF** files are provided: the default file contains the entire message set for the particular language; the "critbug" file contains a reduced set designed for legacy code; and the "security" file contains a reduced set intended for users whose immediate concern is security-related defects and violations.

Other **RCFs**, for example the one for the "m3cm" compliance module, may only show one message group because the file only applies to one target language (in the case of this example, the C language).

Other **RCFs** to address specific requirements can be written by yourself, with guidance from Programming Research Ltd. as required.

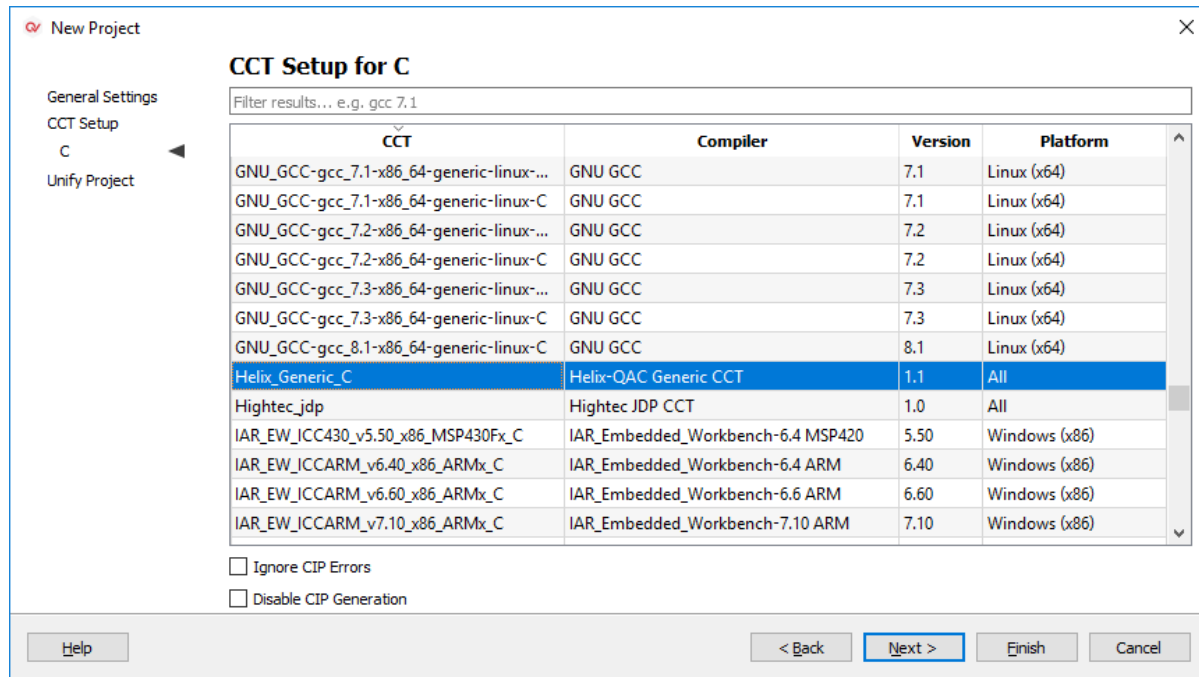
## Compiler Settings

Select the language(s) that apply to your Project. The CCT which will be used on clicking **Finish** is shown next to the compiler selection. This default CCT may be changed on the next page of the wizard if required.

**Note:**

At least one language must be checked before **Next** become active

## Page two: Compiler Compatibility Templates



You must select at least one CCT (Compiler Compatibility Template) to be associated with the project (although you can select one CCT for the C language component and another for the C++ language component).

A CCT is a template of configuration settings that allows the particular component to match the behavior of your compiler. The interface presents all available CCT files, although you can filter them by typing a search string into the text box.

### Note:

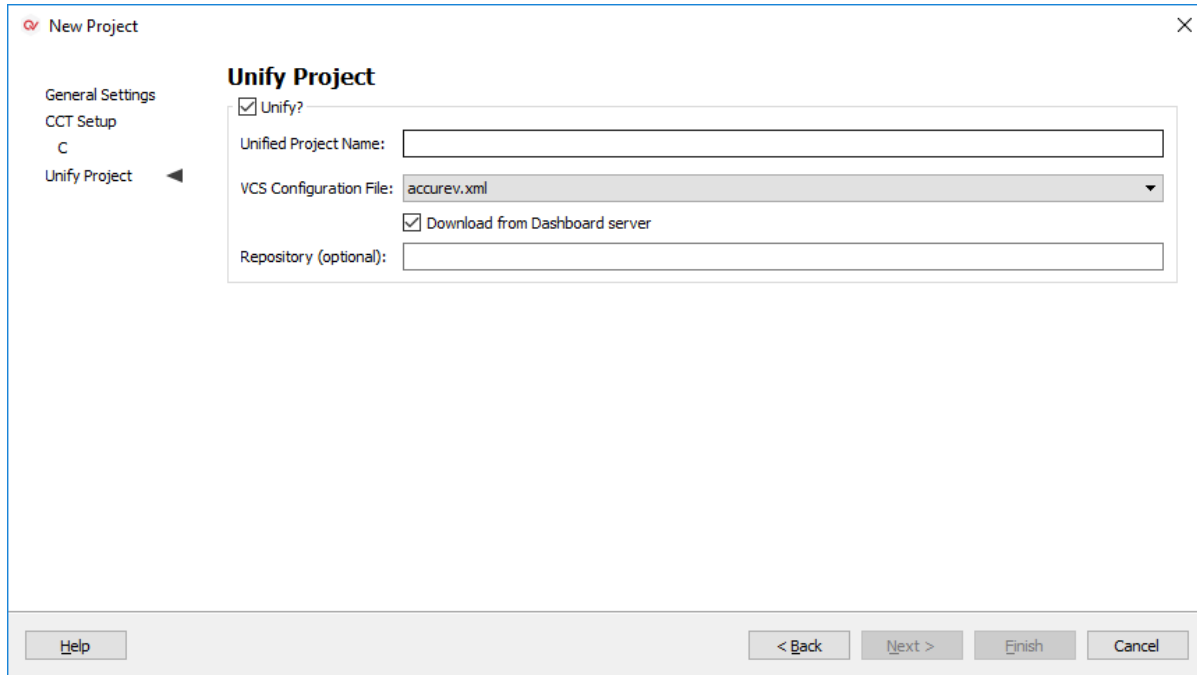
If both C and C++ has been selected, then two **CCT Setup** pages will appear.

Once you have identified a suitable CCT file, select it and press **Next** (or **Finish** if there are no more pages).

### Note:

You can change the selected CCT file(s) after a project has been created, via the Project Properties dialog.

## Page three: Unified Projects



Provided there is a connection to Dashboard, the **Unify Project** page will appear. Project definitions can be centralized by storing them in Dashboard. Such projects are called "Unified Projects". More details on this subject can be found in [Working with Dashboard](#).

Although the recommended approach is to create a "Unified Project" by selecting **Project : Upload Project Definition** once you are happy with your Helix QAC project (refer to [The Dashboard Menu](#)), you can, if required, create a "Unified Project" direct from the **Unify Project** page.

## C and C++ Compile Dependencies

Source files, more often than not, contain **Include** statements. Included files may themselves "include" other files. To analyze a particular file, Helix QAC needs to know where to locate the files that have been specified by all of these **Include** statements, otherwise the analyzers will fail to resolve symbols that are defined in those files. The locations to search for **Include** files are known in Helix QAC as **Include Paths**.

Files are often compiled with definitions that are specified on the command line that compiles the file. For example, the following compile command specifies that the preprocessor is to substitute all uses of the MACRO token with the literal 1 in the source file to be compiled, as well as in all files that have been included in its **Include** dependency chain.

```
gcc -o file.o -I/include/directory/path -DMACRO=1 file.c
```

The files that you wish to analyze, and the Include Paths and Definitions used to compile those files, are normally contained within your build automation or IDE project files. It is necessary to extract this information to populate your project. You can do this using either [Scripted Extraction](#) or [Project Synchronization](#)

## Populating Your Project

When first created, Helix QAC projects only consist of a basic project directory structure and project configuration files. The next step is to populate the project with the source files that you wish to analyze, and the compile dependency information needed to analyze those files.

Populating an empty Helix QAC project can be done in two ways through the Helix QAC·GUI: [Adding Files](#) can be done for any language; [Observing a Build](#) is for C and C++ only.

## Adding Files

Right-clicking on a folder in the [Files Panel](#) displays the option to add one or more files. The extensions of added files must be recognized by the Helix QAC project, and these can be set in the **Project Properties** (refer to [Setting the File Extensions for Your Sources](#)). After files have been added, any project-wide **Include Paths** or **Definitions** need to be set as options to the component in the **Project Properties' Analysis** tab (although they can also be set on a file-by-file basis using the summary table at the bottom of the [Files Panel](#)).

### Note:

The **Project Properties' Sync Settings** tab cannot be used for setting options for manually added files (this tab is only for use with Process Monitoring - refer to [Observing a Build](#)).

Selecting **Add File(s)** displays the **Add files** dialog box. You can use this to add single and multiple directories of files that match the **File/Directory Pattern**.

## Observing a Build

The second method, for C and C++ only, is to have Helix QAC·GUI observe a build of the code project. You trigger a build, and Helix QAC·GUI monitors the process and identifies files being compiled, paths being used, and various other information. Refer to [Synchronization via Process Monitoring](#).

### Note:

Triggering a build cleans the Helix QAC project of all existing files first, and only files that are used in the build will be added. If the code project's build is up to date and no compilation takes place, then the Helix QAC project will be empty. For this reason, you should clean your project first.

Various settings in preparation for this are set in the **Project Properties' Sync Settings** tab.

Many of the following settings will have been set when you previously selected a CCT:

- **Include Path Options:** these should be the switches that the compiler recognizes as indicating an include path.
- **Define Symbol Options:** these should be the switches that the compiler recognizes as defining a value.
- **Option Script:** Path to a script that can extract Include Paths and Definitions from a compiler command line
- **Compiler Settings File Options:** for compilers that store options in a file, these settings allow you to identify that file<sup>1</sup>.

---

<sup>1</sup>For example, Microsoft's Visual Studio compiler stores definitions, include paths and other compiler options in `file.rsp`

- **Quotes Escaped:** this option helps Helix QAC decide how to parse quotes. If the option is checked, any unescaped quotes will be used to encase spaced options. For example, if the option is checked, then the following: `/DMAX (A,B) = "A < B"` will create a macro which verifies whether `A` is less than `B`. If the option is left unchecked, then all quotes will be taken as literals, and the following string will be output: `"A < B"`.
- **Exclude Processes - Comma Separated:** this option allows you to supply a comma separated list of process names that should be ignored. For example, if your build script uses the `mv` command to move files from one location to another, and the `rm` command to remove files from disk, then you may wish to enter `mv,rm` here to ignore these processes.
- **Exclude Processes - Regular Expression:** this option allows you to supply a regular expression for the process names that should be ignored.<sup>1</sup> As above, if your build script uses the `mv` command to move files from one location to another, and the `rm` command to remove files, use `\b (rm|mv) \b` to ignore these processes.
- **File Filter:** this option allows you to supply a comma separated list of files and/or folders that should not be added to the Helix QAC project.

#### Note:

For the other settings, refer to [Compiler Settings File Options](#).

The above settings will persist as part of the project. Once you are satisfied, you can trigger the operation by selecting **Project : Synchronize**. Then proceed as in [Process Monitoring via QA-GUI](#).

## Project Portability and Sharing

Helix QAC provides the two kinds of project portability described in the following subsections.

### Generic Project Portability

In the case of generic project portability, it is just the configuration that is being distributed, which, without relative source files, means the distribution of a "headless project" (refer to [Creating a Helix QAC Project](#)).

Once you create a project and then edit any or all the settings using the Project Properties, the project can be copied anywhere. The Synchronization Analysis (typically using **Project : Synchronize, with Run Analysis At Same Time** checked) will then have to occur on any machine to which the project is copied.

### Project-Specific Portability

Project-specific portability is intended for the redistribution of a complete project, including information extracted from build environments, such as locations of files, **Include Paths** and **CLI Macro Definitions**. As a result, the setting for the source code `PROJECT_ROOT` needs to be set to a location that suits your organization's build dependency environment (refer to [Setting the Root Directory of Your Sources](#)).

A Helix QAC project folder serves as the project identifier and contains the following information required for portability:

- The "prqaproject.xml" file.
- The source folder.

---

<sup>1</sup>The exact specifications of the regular expression (REGEXP2) can be found here [doc.qt.io/archives/qt4.8/qregexp.html](http://doc.qt.io/archives/qt4.8/qregexp.html)



- The "prqa" folder containing the following sub-folders:
  - "upgrade" - contains files relating to any upgrade of the project from a previous version. Otherwise, the sub-folder may be empty.
  - "unified" - populated only if the project is "Unified".
  - "configs" - contains at least one configuration (the default is **Initial**), although others may be added (refer to [Projects with Multiple Configurations](#)).

Each configuration sub-folder within "configs" contains the following:

- "reports" - may be empty.
- "output" - may be empty.
- "config" - contains the following: a single ACF, a single RCF, and one or more CCTs.

Everything else is non-portable and is not needed to re-analyze the project on a different machine, and therefore does not need to be checked in.

In the case of project-specific portability, the source files must be in the same location relative to the source code `PROJECT_ROOT` on the new machine.

#### Note:

Most Version Control systems have an "ignore" mechanism that is used to ignore certain files and file types. This same mechanism should be used to ignore all nonportable files and directories contained in the project directory structure, for example any cip folder that may have been generated.

## Compiler Selection

The **Compiler Selection** tab allows you to search and select CCTs that match your compiler.

The radio buttons lets you switch between C or C++ CCTs. The currently selected CCT is shown to the right of the radio buttons.

## CCT Filtering

There are many available CCTs for some languages. Type a search string into the filter box in order to narrow the search down to the CCT required for your system.

## Selecting a CCT

To select a CCT, click on one from the list and press the **Use CCT button**. Or double click on a selection to pick a CCT.

## Updating a Project CCT

If the CCT used by the project is different to the master copy, the Update CCT checkbox is enabled. Checking the box causes the CCT for the project to be updated when the Save button is pressed.

## CIP Settings

There are several CIP settings that can be modified which correspond to the **Currently Active CCTs**:

- Selecting the **Ignore CIP Errors** check box will ignore any errors returned from the CIP generation scripts. If the CIP script is producing errors and you want analysis to continue then select this option (the analysis results may be partial depending upon the generated CIP content).
- Selecting **Disable CIP Generation** will stop Helix QAC from running (and cleaning) the CCT generation scripts. This functionality can protect any external or custom CIP files from being overwritten.

## Baseline Diagnostics Suppression

Baseline diagnostics suppression is used to suppress old diagnostic warnings when the decision has been made to begin a new cycle of development.

The baseline consists of the set of diagnostics present when the baseline was generated. In any subsequent analysis, if a diagnostic can be identified as being the same as a diagnostic within the baseline, it is suppressed. Sophisticated "diff"-style techniques are used to match diagnostics, even if some of the surrounding code has changed.

Baseline suppressions are applied at view time. They do not affect the generation of diagnostics. This means that you can see exactly what has been suppressed by selecting the **Show baseline suppressed diagnostics** at the top of the **Analysis Results/Diagnostics** panel.

### Note:

It is your responsibility to ensure that there are no timeout messages if you intend to generate a baseline. This is because timeouts can lead to inconsistencies in generated messages and therefore inconsistent baseline suppression. This is particularly important if **Dataflow Analysis** is enabled, because this form of analysis can time out when analyzing complex code.

Baselining is particularly useful when dealing with legacy code, where code changes are often seen as undesirable. Apply a baseline to the legacy code, and then, once the baseline is in place, diagnostics resulting from code changes will be displayed, whereas diagnostics from the original code will not.

## Generating a Baseline

There are three types of baseline:

- **Baseline from a Dashboard snapshot** (if you are using a "Unified Project").
- **Baseline by Version Control.**
- **Baseline by Local Copy.**

The options for **Baseline Diagnostics Suppression** are displayed by opening the **Project Properties** and selecting the **Baseline** tab.

The options panel is initially disabled. Select the **Use Baseline Diagnostics Suppression** check box to enable it:

## Baseline by Dashboard snapshot

This option is available if the project is "Unified", that is to say, if it has been stored in Dashboard. At least one snapshot must have been uploaded to Dashboard from the project.

Generating the baseline involves selecting a suitable snapshot, and downloading the diagnostics from the snapshot. This process is controlled via the **Dashboard : Download Baseline** menu option.

A full description of baselining from Dashboard is given in [Downloading a Baseline](#).

#### Note:

When uploading to Dashboard, ensure that the source code is included in the upload. If not, then Helix QAC will not be able to suppress diagnostics with a baseline downloaded from Dashboard.

## Baseline by Version Control

Before generating a baseline by Version Control, the project must have been fully analyzed, and a version control script should have been created in the **Project Properties' Version Control Configuration** tab.

Select **Baseline by Version Control**, and then click **Generate Project Baseline** and save the new options. The old warnings will be suppressed.

## Baseline by Local Copy

This option allows you to indicate that all current diagnostics (excluding CMA diagnostics) are to be suppressed. If there are no current diagnostics (for example, because the project has not yet been analyzed, or has been recently cleaned), no diagnostics will be suppressed.

Click **Generate Local Baseline**. This creates the suppressions for the project and automatically updates the location of the suppressions information. This location is visible below the **Generate Local Baseline** button. Then select the **Baseline by Local Copy** radio button.

#### Note:

Selecting the Baseline by Local Copy radio button without having generated a baseline results in an error message advising you to create the baseline first.

Any changes must be saved.

When you close the **Project Properties** dialog, the project will need to be cleaned and re-analyzed in order to suppress the diagnostics. Suppressed diagnostics can be seen by toggling the **Show baseline suppressed diagnostics** option in the **Analysis Results/Diagnostics** panel.

## Projects with Multiple Configurations

You can now create a project with multiple configurations, saving you the effort of creating duplicated projects with different settings. All newly created projects will automatically support multiple configurations and any previously created projects will automatically be upgraded with this functionality. In fact, each project will be classed as one with multiple configurations - even if it only has a single configuration.

## Restrictions

There is a different ACF, RCF, VCF and **User Messages File** for each configuration. In addition, each configuration will have its own database, and hence the diagnostics, output and reports may differ between configurations. However, all configurations share the same source files, and so any options that affect which files are in the project (for example, adding files via **Project : Synchronize** or via the right-click menu in the [Files Panel](#)) apply to the project as a whole.

With regard to Dashboard, you can only unify a project that has a single configuration (refer to [The Helix QAC Dashboard Menu](#)). This restriction is enforced.

**Note:**

Configuration names are case sensitive, and you are not allowed to create configurations that differ only by case. Therefore, for example, you cannot create configurations "Foo" and "foo".

## Multiple Configurations with QA-GUI

Each Helix QAC project must have at least one configuration. This is known as the **Initial Configuration** and is automatically created on building a new Helix QAC Project (or upgrading an existing Helix QAC Project). This then also becomes the **Active Configuration** by default. Refer to [Configuration Manager](#) for the creation of new configurations for your project.

## Selection of Alternative Project Configurations

You can change the **Active Configuration** via the drop-down menu of the same name on the Helix QAC GUI toolbar or via the **Project Properties** (at the top of the dialog box).

When you change configuration, Helix QAC will update automatically to reflect the new configuration settings, and the load analysis diagnostics (if they exist for the newly selected configuration).

**Note:**

The analysis results for each configuration are kept separate, because different settings can lead to different diagnostics, dependent, for example, on whether **Dataflow Analysis** is on or off.

## Configuration Manager



The **Configuration Manager** is accessible from Helix QAC GUI, via either the toolbar or the menu option **Project : Configuration Manager**.

The **Configuration Manager** dialog lists all the configurations for the Helix QAC Project, with the current default configuration highlighted.

When a configuration is selected, you can Copy, Rename or Delete it, or set it as the **Default Configuration** to be used when no explicit configuration is specified (this is principally for CLI compatibility, but is also useful when using multiple-project CMA for which the projects consist of multiple configurations).

## Configuration Changes - Project Properties

Besides being able to change the **Active Configuration** at the top of the **Project Properties** dialog box, you can also modify configuration settings under the **Analysis** tab.

The tabs that contain settings which affect the whole project (for example, the **Project** and **Sync Settings** tabs) are indicated by the  next to the tab name. The remainder of the tabs are those that contains settings which affect just the **Active Configuration**, and these are indicated by the  next to the tab name.

**Note:**

Changing the **Active Configuration** within **Project Properties** will also load analysis diagnostics if they exist for that configuration.

## Multiple Configurations with Helix QAC·CLI

Support has been added to the Helix QAC·CLI subcommands to support working with multiple configurations. If you are only working with a single configuration then no changes are needed to your usual CLI subcommands and options.

Most subcommands already use `--qaf-project` (or `-P`) to specify which Helix QAC project to use. Now, an additional option can also be used: `--config` (or `-K`). This allows you to specify which configuration to use for the operation. If this is not specified then the Default Configuration will be used. You can retrieve the list of configurations for a project by using the `--list-configs` option, for example:

```
qacli admin -P . --list-configs
```

This will list all the configurations associated with the project in the current directory. The default configuration will also be highlighted.

This default can be changed using the `--set-default-config` option:

```
qacli admin -P . --set-default-config -K <config>
```

Sub-commands have also been added to allow you to **Copy** (`--copy-config`), **Rename** (`--rename-config`) and **Remove** (`--remove-config`) configurations. The syntax is as follows:

```
qacli admin -P . -K <source> --copy-config <target>
qacli admin -P . -K <source> --rename-config <target>
qacli admin -P . -K <source> --remove-config
```

**Note:**

"<source>" must be an existing configuration and the "<target>" configuration must not already exist.

To remove a configuration, you need to have at least two configurations in the project - that is to say, a project must always have at least one configuration.

If you remove the configuration that is currently the default, then a new default will be selected automatically. You can subsequently change this using the `--set-default-config` option.

If you rename a configuration that is currently the default, the default will also move.

## Deleting a Project

You can remove an existing Helix QAC project by doing any of the following:

- If the project is already open in Helix QAC, select **Project : Close and Delete Project** from the main menu. You are prompted for confirmation that you wish to close the project and remove all Helix QAC files from the project. Following confirmation, the project is closed, the Helix QAC files are removed from the project, and the project name is removed from the **Recent Projects** list if appropriate.
- If you wish to browse to the project, select **Project : Delete Project** from the main menu. Then use the resultant file explorer dialog to select the required project. You are prompted for confirmation that you wish to remove all Helix QAC files from the project. Following confirmation, the project is closed if necessary, the Helix QAC files are removed from the project, and the project name is removed from the **Recent Projects** list if appropriate.
- If the project is on the **Recent Projects** list, right-click on it and select **Remove Helix QAC files and project from list**. You are prompted for confirmation that you wish to remove all Helix QAC files from the project. Following confirmation, the project is closed if necessary, the Helix QAC files removed from the project, and the project name is removed from the list.

**Note:**

Deleting a project using any of the above options will not remove your source code and associated files - it is only the Helix QAC files and diagnostics that are removed.

## Optimizing a Project

For large projects, the size of "prqaproject.xml" may become unwieldy as **Include Paths** and **Definitions** can be duplicated across thousands of files. The size of "prqaproject.xml" greatly impacts the performance of Helix QAC. Optimization deduplicates these dependencies by moving any shared data from individual files to parent folders.

Project optimization can be achieved by selecting **Optimize project XML during save manually** in the **Project Properties** dialog.

**Note:**

Suppressed Include Paths ([Suppress Include Paths](#)) are not optimized, and so they remain attached to individual files.

## 5 | Analysis Configuration

### The Analysis Tab

The **Analysis** tab in **Project Properties** allows the main parameters of the **Analysis** toolchain to be defined using [The Analysis Configuration Panel](#).

Each setting established at this level can be exported to a dedicated file with the extension `.acf` (Analysis Configuration File).

A description of the options available for each individual component can be found in the manual for that component.

### The Analysis Configuration Panel

#### Importing, Exporting and Clearing Configuration Settings

You can do the following via the **Analysis Configuration File (ACF)** controls:

- Click the **Import** button in order to select a new **ACF** with different settings.
- Click the **Export** button in order to save the existing setting to a specified location, thereby creating an **ACF** that can be imported into other projects.
- Click the **Clear** button in order to remove the existing settings.

**Note:**

Clicking the **Save** button at the bottom of the panel saves any changes made to the existing configuration.

Once an **ACF** has been added to a project it's name will automatically be changed to `project.acf`.

### User Messages File

The **User Messages File** allows you to assign your own messages for each component against existing or newly created diagnostic numbers. Optionally, you can set an associated HTML help file and references for a message - refer to [Creating User Messages](#).

**Note:**

Helix QAC will copy the selected HTML help file to the same folder as the **User Messages File**.

The **User Messages File** is generated automatically when you create new messages via

Helix QAC·GUI (refer again to [Creating User Messages](#)). This is stored as a human-readable XML text file that can be edited manually as well as stored in a repository.

It is advisable to restart Helix QAC after making manual changes to the User Messages XML File.


**Note:**

The **User Messages File** is applied to Helix QAC and not to individual projects.

It is also possible to set the **User Messages File** via the command line interface refer to [Administration of User Messages File](#) File for the appropriate command.

For details on the XML file for manual construction, we recommend that you create a simple **User Messages File** via Helix QAC GUI to observe the XML structure, and then replicate the XML structure according to your needs.

## Creating User Messages

To create a new user message, click  to the right of the component selected under **Analysis Toolchain**

The User Messages Dialog is displayed.

Scroll all the way down to **User Specified** and right-click on either **User Specified** itself, or the required component beneath it, and select **New User Message**. Then enter a **Message ID** in the range 6000-9999, enter the associated text, followed by the appropriate **Severity** (0-9) (refer to [Viewing and Filtering the Diagnostics](#)). You can also, if required, link the message to an HTML Help File and enter any **Reference(s)**.

Click **Create**.

To use a newly defined user message with a rule, then, once you have saved the configuration, proceed as in [Messages](#).

## Selecting the Source Language Toolchain

A source language toolchain is a set of components applied to a particular programming language, inclusive of the parser for that language.

Additionally, a language toolchain may include optional modules, for example compliance modules to check against the MISRA standard.

Use the **Source Language Toolchain** drop-down to select the required toolchain.

## Mixed Language Toolchain

When using mixed-language projects, Helix QAC will apply the toolchain for the relevant language. For example, in a project using the C and C++ programming languages, the C toolchain will be run against the C code, and the C++ toolchain against the C++ code.

## Common C and C++ Components Toolchain

The **Common C and C++ Components** toolchain is designed for analysis where you wish to detect issues that may exist in your system as a whole. Underneath this toolchain is where you will be able to set up and configure Cross-Module Analysis if it is available on your system (using the "RCMA" component).



## Selecting the Toolchain Components

Once the toolchain has been selected, it is possible to select within the **Analysis Toolchain** panel which components will make up the toolchain for the particular language.

By default, the primary analyzers such as Helix QAC for C and Helix QAC for C++ - the C and C++ parsers respectively - are already available in the panel. Secondary analysis components are available in the left hand **Available Components** panel. If a secondary component needs to be part of the current analysis, it can be added to it by using the ">" button; it can conversely be removed by using the "<" button.

Many components require the primary analysers and RCMA to be in the toolchain as they process the results from them. If you attempt to add a component to the toolchain that's dependent on a component not currently in the toolchain, you will be prompted and given the option to automatically add the required components.

Further to this, if you add a component to the toolchain and there are no corresponding rules, you'll be presented with the option to automatically import an appropriate RCF.

This page Intentionally left blank

This page Intentionally left blank

## 6 | Rule Configuration

Helix QAC uses an RCF (Rule Configuration File) to define the mapping of warning messages detected by analysis, to rules. The **Project Properties' Rule Configuration** panel presents the rules in a tree-like structure that is organized into levels and sub-groups. When the lowest sub-group/rule is selected, the actual enforcement messages are shown.

The panel opens with the RCF for the current project.

You can select a new RCF by clicking the Import button. The imported file must have the extension `.rcf`. When you select the required file, the Confirm update dialog is displayed.

The options are as follows:

- **Replace:** The file is replaced by the one specified.
- **Update:** The existing file is updated, replacing the mappings for the existing components.
- **Merge:** The existing file is updated, merging rulesets from both files with the same component (in the case of a merge conflict where the new and old values differ, then the new values are used).

**Update** should be selected if you are already using a default RCF and you still require default mappings. **Merge** should be selected where you have customized the RCF and wish to retain the customizations (where possible).

Select the required option, or else **Cancel** to leave the dialog without making any changes. Notes:

- If the results of the **Replace**, **Update** or **Merge** operation are not as required, then close the **Rule Configuration** panel without saving your changes.
- Once an RCF has been added to a project it's name will automatically be changed to `project.rcf`.

## The Default Rule Configuration Files

Some RCFs are delivered with the product. Refer to [Rule Configuration File](#) for guidance.

## Rule Configuration Operations

In the **Rules** panel, a number of operations are available from the context menu that appears when you right-click on a selected level. These are covered in the following sub-sections.

### Note:

These operations will have a direct effect on the project's **Active Configuration**.

## Edit

Every level in the rules tree below the analysis component has an associated **Rule ID**, **Rule Text**, **Rule Help** file and, optionally, one or more **Categories**, associated with it. To modify a particular level, first right-click on it and select **Edit** to display the **Rule Editor** dialog.

You can de-activate rules within the **Rule Editor** dialog (by unchecking the **Active** box), and modify any of the associated values (**Rule ID**, **Rule Text**, **Rule Help** file and **Categories**). You can also insert of new **Categories** by double-clicking inside the **Categories** panel.

Changes that are saved within the dialog are reflected in the **Rule Groups** panel after analysis.

**Note:**  
Rule IDs may be duplicated.

## Remove

Following confirmation, this operation removes the selected level, and all its sub-levels, from the RCF.

This option should be used with caution as the blocks of enforcement messages are also removed. However, only the local copy of the RCF in the project configuration subfolder is affected (refer to [Project-Specific Portability](#)). The original RCF can be restored by reloading it using the **Import** button at the top of the **Rule Configuration** panel.

## Disable/Enable

Disabling a rule in the Rules panel means that the analysis results, that is to say the warning messages, are no longer mapped to it. Initially, the **Rule Groups** panel displays a disabled rule.

When the project is re-analyzed, warning messages are no longer mapped to the rule, and it is not shown in the tree. The message counts are reduced accordingly.

You can re-enable a rule by right-clicking on it in the Rules panel and selecting the **Enable** option.

## New Rule

A new rule may be defined at any level in the tree. Just right-click on the level below which you wish to create the new rule (or on the name of the component, for example "Helix QAC for C" or "Helix QAC for C++", if you wish to create a new top level rule). Then select **New Rule** in to display the Rule Editor, complete the fields, and click **Save**.

The **New Rule** feature is useful for defining customized rule groups in a hierarchy.

## Messages

The rule levels in the hierarchy may contain either sub-groups or actual enforcement messages, or both. For example, although our new rule contains sub-levels, we can go on and associate these with enforcement messages.

## Associating Messages with Rules

To associate one or more messages with a rule level, right-click on the level and select **Messages**. The **Rule Enforcement Messages** dialog is displayed. Each of the available analysis components has its own set of messages for the following categories: **Errors**, **Messages** and **User Messages**. Open the required group and check the messages that are to be selected for the particular rule level.

**Note:**  
User Messages are defined as in Section Creating User Messages.

## Disassociating Messages from Rules

To disassociate one or more messages from a rule, right-click on the message(s) in the **Rule Enforcement Messages** panel under the **Rule Configuration** panel, and choose either **Remove selected** or **Disable selected**. These actions have the same effect as deselecting the message(s) using the **Rule Enforcement Messages** dialog, although **Disable selected** will leave the message(s) visible on the panel, albeit in a dimmed state, so that you can quickly re-enable the message(s) (by choosing **Enable selected** from the same right-click menu).

**Note:**

Selecting the Remove All Messages option disassociates all messages from a rule, and simultaneously removes the messages from the Rule Enforcement Messages panel.

Do not forget to click **Save** to keep your changes.

## New Rule Configuration

It is possible to define a completely **new Rule Configuration** hierarchy, after first clicking the **Clear** button to remove the existing one. Start in the **Rules** panel, and, from the right-click menu, select **New Rule Group** and specify the name of a new top level group. Then right-click on the **Rule Group** and select **New Rule** to display the Rule Editor dialog. This allows you to create a new rule. Finally, you can apply enforcement messages by selecting **Messages** from the right-click menu for the rule and then proceeding as in Section [Associating Messages with Rules](#)

Continue as above until you have the required hierarchy. When you click **Save**, the hierarchy will be saved under the local project configuration as `project.rcf`, within the name of the folder associated with the **Active Configuration**. You can also click the **Export** button to save the **Rule Configuration** under a different name and location.

**Note:**

The original RCF can be restored by reloading it using the **Import** button at the top of the **Rule Configuration** panel.

This page Intentionally left blank

This page Intentionally left blank

## 7 | Project Synchronization

Helix QAC can perform various types of synchronization.

**Process Monitoring** synchronization is used to derive source files, **Include Paths** and **Definitions** from an existing C or C++ project. It works with the majority of IDEs (Integrated Development Environments), build systems (such as **Make**, **SCons** and **CMake**), and compilers.

Several plug-ins are also provided to enable synchronization with popular build systems such as **MS Build**.

---

### Synchronization via Process Monitoring

**Process Monitoring** works by recognizing compiler processes that are spawned by a build process, analyzing your files as they are added to the project. It needs to be able to recognize the compiler to do this.

You can monitor processes for a C or C++ project using either QA·CLI (refer to [Process Monitoring via QA·CLI](#)) or QA·GUI (refer to [Process Monitoring via QA·GUI](#)).

**Note:**

Source code projects are rarely static, with files and dependencies being regularly added and removed. The **Process Monitoring** feature allows you to re-synchronize your Helix QAC project whenever you need to.

### Process Monitoring via QA·CLI

If you are performing **Process Monitoring** via QA·CLI, you need to be able to build your project on the command line.

As an example, let us suppose that the `make all` command builds a project and that the `makefile` is located in the `/home/Perforce/workspace/` directory. To populate your project, change directory to the project directory and issue the following command:

```
qacli sync -P <project-path> --type MONITOR <build-command>
```

Helix QAC starts the build process and attempts to extract the necessary information. Each file that is extracted is added to the Helix QAC project.

If your build command is over-complex, it may contain characters that are open to misinterpretation by the shell. This can be overcome by creating a short script that runs your build, as in the following example:

```
#!/bin/bash
cd /home/prqa/workspace/                # necessary if run from QA GUI
export SOME_MACRO=10
make clean                             # ensure to clean first
make target1 CXX=g++-4.7 CC=gcc-4.7;
```

A Windows batch file similar to the above script is as follows:

```
c:
cd C:\prqa\workspace
set SOME_MACRO=10
nmake clean
nmake target1 CXX=cl.exe CC=cl.exe0;
```

Save the script, giving it a name, make the script executable, and issue the following command<sup>1</sup>:

```
qacli sync -P <project-path> --type MONITOR MyScript.sh
```

Naturally, the project must be cleaned prior to building. Otherwise, nothing may be compiled and there will be nothing to monitor.

## Process Monitoring via QA-GUI

To use the **Process Monitoring** feature with QA-GUI, it is necessary to use a script, such as the one in [Process Monitoring via QA-CLI](#). Operating in a graphical user shell is quite different from operating on a command line, in that a graphical user shell or windowing environment, such as the Microsoft Windows desktop environment, has no concept of a "current working directory". For this reason, it is necessary either to specify the working directory explicitly (using the **Optional Working Directory** field referred to below) or to have the script specified in the **Enter Build Command** field modify the directory in which you wish to build your project.

To access the feature, select **Project : Synchronize** from the main menu, the **Synchronize** dialog is shown, and then ensure that **Process Monitor** is the option selected in the **Select Sync Type** drop-down.

The fields to be completed in the dialog are as follows:

- **Optional Working Directory**: An optional directory from which to run the build command specified below. If this is specified, then commands can be entered as if on the command line, for example via **Make** or some other build or compilation utility.
- **Enter Build Command**: This is a mandatory field, in which you can specify either a build script or the executable for an IDE, for example, Visual Studio (this IDE would then be observed for code building activity).

### Note:

To help you locate paths, there is a  icon to the right of the fields to allow you to browse to them.

A full build must be performed when doing a synchronization. Any existing files in the Helix QAC project will be removed if they are not in the current build.

### Note:

Check the **Run Analysis At Same Time** box at the bottom of the dialog if you wish the project files to be analyzed at the same time as they are being built.

When you click **Synchronize**, you will be presented with information regarding the addition of files to the Helix QAC project.

---

<sup>1</sup>POSIX paths are valid on Windows. In any case, Helix QAC understands both Windows and POSIX paths. However, paths that contain spaces must be surrounded by quotes.



**Note:**

Check the Optimize Project box if you want the project size to be small (refer to [Optimizing a Project](#)).

If **Show Build Output** is checked, the synchronization process dialog will provide more information about the build process, explained below in Section [Verifying Process Monitor Synchronization](#).

## Verifying Process Monitor Synchronization

An effective method of investigating issues with Process Monitor Synchronization is to use the advanced Build Output Synchronization dialog, which can be enabled by checking **Show Build Output**. Once the synchronization process starts, a progress dialog is displayed that has more in-depth information concerning the output of the build process and processes found.

An overview of the dialog tabs:

- **Files Found** captures the same functionality as the standard Synchronization Progress dialog; it shows whether any files were found and their location, and the success of the build command.
- **Compile Output** and **Compile Issues** populate with the output and error streams from the build process, respectively.
- **Processes Detected** lists all processes that were found during synchronization. Each item in the list can then be expanded to show what build command line(s) a process has been called from.

## Synchronizing a Visual Studio Project

Helix QAC provides a plug-in (**QAVSConverter**) to facilitate the synchronization of a Visual Studio (VS) C or C++ project, offering you the possibility of converting one or more VS projects to a Helix QAC project.

## Preparing the Helix QAC Visual Studio Plug-in prior to Synchronization

The plug-in options are as follows:

### **-p, --vsproj <VSProjectFullName> [Required]**

- You can specify the full path to the Visual Studio project. A single Helix QAC project will be created that includes all the files from the Visual Studio projects specified with this option. A list of Visual Studio projects can be sent for synchronization, as follows:

```
-p VSProjectOneFullPathName VSProjectTwoFullPathName ...
VSProjectXFullPathName
```

### **-c, --cfg <ConfigurationName|PlatformName> [Optional]**

- You can specify the Visual Studio Configuration for which the Helix QAC project will be created, using the format `<ConfigurationName|PlatformName>` (for example, "Debug|Win32", "Release|x86", "Debug|AnyCPU"). If no Configuration name is given, then the active Configuration from the Solution will be used. This option should not be used if the `-t` and `-f` options are used.

Alternatively use the following 2 options to specify the Visual studio configuration name and platform:-

- `-t, --cfgType <ConfigurationName> [Optional]` where `<ConfigurationName>` is one of the Visual studio project configuration names eg; `_Debug`, `Release`

- `-f, --cfgPlatform <PlatformName> [Optional]` where `<PlatformName>` is one of the Visual Studio platform names eg:- Win32, x86, x64 etc

Important: Use the `-t` and `-f` options together instead of the `-c` option. Do not use singly.

#### **-s, --solutionDir**

- You can specify the solution directory, this may be required where a project uses common properties shared with other projects within the solution.
- `--solutionDir C:\Users\ccarr\AppData\Local\Perforce\Helix-QAC-2020.2\samples\sample_VS19x86-2020.2\src\`

**Note: The trailing '\' character is required.**

#### **Note:**

It is important that, prior to the start of synchronization, the Visual Studio project has been built successfully.

#### **Contents of Helix QAC project:**

- All **Definitions** extracted from the VS project, including the predefined ones.
- All project and system **Include Paths**.
- Pre-compiled headers for C++ projects.
- All files with extensions supported by Helix QAC.
- All files included in the build for the specified configuration.

## **Performing the Visual Studio Synchronization using the GUI**

A Helix QAC project, created to analyze a Visual Studio project, can be synchronized by selecting **Project : Synchronize** from the main menu (or by using keyboard shortcut: Alt + A, Alt + B), and then selecting **Visual Studio Build** as the **Sync Type** in the **Synchronize** dialog.

The input fields are as follows:

- **Optional Working Directory:** The directory where the **QAVSConverter** command will be executed. If the field is left empty, the project root directory will be used as the working directory.
- **Project File(s):** One or more paths to the Visual Studio projects (`.vcxproj`). Multiple paths, when provided, must be separated by spaces. Each path must be wrapped in double quotes.
- **Configuration :** The Visual Studio build configuration used by the project you wish to synchronize.
- **Platform:** The platform which the project is built on.

#### **Note:**

Check **Run Analysis At Same Time** in the dialog if you wish the project files to be analyzed at the same time as they are being built.

**Note:**

It is strongly recommended that you perform a build of the project outside of the GUI before attempting the synchronization (in order to reduce the amount of time needed for synchronization).

Having completed the input fields, click **Synchronize** to start the synchronization process.

## Verifying the Outcome of the Synchronization

Once the synchronization process has started, the progress window is displayed.

## Manual Synchronization Methods

Helix QAC provides two methods of project synchronization that do not require build commands. Using the GUI, the only required field is the path to a single file and the current project will be synchronized. For details about these same methods using QA-CLI, please refer to [Report](#).

## JSON Compilation Database

Synchronization can be carried out using a JSON Compilation Database file, a JSON based data format that is used by the Clang compiler. The Compilation Database contains the working directory, compile command and include paths required to compile each translation unit.

A single JSON Compilation Database entry has this format:

```
"directory": "/home/steve/Temp",
"command": "g++ -std=c++11 -I . enum.cpp",
"file": "/home/steve/Temp/enum.cpp"
```

refer to <http://clang.llvm.org/docs/JSONCompilationDatabase.html> for further details.

There are various build systems that support the generation of JSON Compilation Databases including Clang, CMake (version 2.8.5 onwards) and Ninja(1.2 onwards). This manual will not cover the generation of the databases with all build systems, however in the case of CMake, all that is required is the following flag to be added to the build:

```
cmake -DCMAKE_EXPORT_COMPILE_COMMANDS=ON
```

If your build systems do not currently support this generation, there are many tools that can intercept the compilation process and generate a corresponding compilation database. One of the more prominent tools is Bear<sup>1</sup>, and allows for simple usage:

```
bear BUILD_COMMAND
```

which generates a `compile_commands.json` file in the current directory.

---

<sup>1</sup>The compiler interception tool, Bear, can be found here <https://github.com/rizotto/Bear>

## Build Log

The 'Build Log' method of synchronization is used similarly to JSON Compilation Database synchronization, but its application is more generalized as many generic build system logs are suitable. This method of synchronization will work given that:

- The build log is newline-separated for each source file that was compiled.
- The lines that contain a path to a source file also contain the **Include** paths and macro **Definitions** needed to compile the file.

If your build system does not support the creation of a build log in this format, then, for most IDEs and build systems, a log can be created by using a script that wraps your compiler and outputs all calls to the compiler executable to a log file. The following is an example `gcc` wrapper using `Bash`:

```
#!/bin/bash
echo "$@" >> /home/user/mybuild.log
exec /usr/bin/gcc "$@"
```

Using this example, all calls to `gcc` must be redirected to this executable script. Rather than manually altering your project's build information, a simple method is to name this executable '`gcc`' and place it on the environment variable `PATH` prior to the actual `gcc`. It should be noted that if your project has calls to `gcc` from multiple directories, this script would collate all relative paths and thus would be unusable without modification. A way to overcome this problem is to convert all relative paths to absolute ones within the bash script:

```
#!/bin/bash
for var in "$@"
do
    if [[ $var = *".c"* ]]; then
        var="${PWD}/${var}"
    fi
    echo -n " ${var}" >> /home/user/mybuild.log
done
echo >> /home/user/mybuild.log
exec /usr/bin/gcc "$@"
```

When the **Optional Working Directory** field is left unspecified, the paths within the build log are considered absolute or relative to the project root directory.

## Synchronization Options

In the majority of cases, synchronization works with the default options provided. However, if problems are encountered, then they can be solved by changing the existing defaults.

Settings in the disabled (read-only) fields will have been read from the CCT file. If the CCT has its own **File Options**, this set of options will also be displayed. These synchronization settings can be changed if the project synchronization is not successful.

## Include Path Options

The **Include Path Options** contain the flags used by the compiler to specify an **Include path**. The defaults of `-I` for Linux and `/I` for Windows should be suitable for most settings. Some compilers may use or allow other flags and these can be added in the **Include Path Options** if required. For example, **GCC** also allows `-iquote` to be used.

## Define Symbol Options

The **Define Symbol Options** contain the flags used by the compiler to specify and define a macro on the command line. The defaults of `-D` for Linux and `/D` for Windows should be suitable for most requirements. If your compiler uses other flags, they can be added to the comma separated list or used to replace the defaults.

## Compiler Settings File Options

Compilers can store the command line options in a file, and the **Compiler Settings File Options** contain the **Flag** and file **Extension** used to specify these. The file Extension can be left blank if it not used.

If the **Literal Quotes** box is checked, all quotes are treated as literal and passed to the analyzers as if they had been written in the settings file, otherwise the synchronization expects literal quotes to be escaped.

Each set of options is saved under a **Name**. To modify an existing set of options, highlight the associated **Name**, and after making your changes, click **Save New Setting** to overwrite the existing options. If you enter a new **Name** in the dialog, or amend an existing **Name**, then, when you click **Save New Setting**, the associated options will be saved as a new entry under **Compiler Settings File Options**.

To delete an entry, select the **Name** in the list and click **Delete Selected**.

## Exclude Processes

Synchronization may detect false positives for compiler processes. This can cause source files to be incorrectly added to the project, or for the **Definitions** and **Include Paths** to be incorrectly formatted. The **Exclude Processes** option can be used to prevent this from happening.

Exclusion can be done by either:

**CSV** : Which is a list of process names to exclude. Only the first word on the command line is checked and it must be an exact match.

**Regular Expression** : The supplied regex is used against the entire command line for a match.

### Note:

Any process that mentions a source file can result in a false positive. For example, `rm, ln` or `/bin/sh`.

Compile commands are often launched with `/bin/sh` but the command line may not be fully resolved at this point, which can lead to incorrect information being found.

Following a synchronization, all compiler processes used to create the project will be displayed. Any processes that do not match your compiler should be added to the **Exclude Processes** list and then the synchronization run again.

## File Filter

Using the File Filter, it is possible to filter out individual source files and folders. String matching is used, and so, for example, `project/subproject` will filter the folder `/home/project/subproject`, as well as `project/subproject1.cpp`.

## Suppress Include Paths

Checking this setting at the bottom of the dialog will suppress **Include Paths** when files are being added during the synchronization or **Compiler Wrapping** (refer to [Compiler Wrapping](#)).

## 8 | Version Control Configuration

Helix QAC operates on the code that is currently being worked on locally by the developer. In most cases, this will have originally come from a Version Control System, but it is the local version of the code that the analysis results are generated for. Analysis only needs to know the location of the local code, it does not need to know where it came from. For most people working on the desktop, there is no need to tell Helix QAC about the Version Control System.

By contrast, Dashboard normally works directly against a Version Control System. Dashboard deals not just with the current code, but it holds a series of snapshots from the lifetime of the project. Extracting historical code from a Version Control System is the most efficient way of handling this. It also ensures its reporting is accurate and traceable. Results uploaded to Dashboard normally come from a server-side automated process that extracts code from the Version Control System before generating the results. There is then a clear correspondence between results and versioned code.

A Dashboard Unified project (see [Working with Helix QAC Dashboard](#)) will have a configuration file attached to it that gives details on the Version Control System. This is known as a Version Control Compatibility File (VCF) file. There is no need to create this file locally, this is done centrally within Dashboard, and the necessary file is simply pulled down from the server as part of the project definition.

The only circumstances where a VCF file needs to be created or adjusted locally are:

1. If the [Baseline by Version Control](#) feature is being used then baselining needs a reference copy of the code, so that it can compare the current code with the code that existed when the baseline was taken.
2. The current project is not a Unified project, but results generated against checked-in code need to be uploaded to Dashboard.
3. For some Version Control Systems, a VCF file that is pulled down from Dashboard may need adjustment for local usage.

To provide the necessary information, select '**Open Project Properties**' and then select the '**Version Control Configuration**' panel.

A number of scripts exist for popular version control systems. They can be imported using the '**Import**' button. When imported, the script is displayed in the 'VCF Script' panel. The script can be edited if necessary to work with your particular Version Control System.

You should first verify that the script works (see [VCF File Testing](#)). Once this has been confirmed, be sure to save the **Project Properties**. The script will then be used by Helix QAC.

### Note:

Once a VCS script has been added to a project it's name will automatically be changed to `project.vcf`.

## VCF File Testing

To check that the VCF file is configured correctly it can be tested using the options provided.

First ensure that Helix QAC can interface to the project's Version Control System. Select a source file using the Browse button and select the 'Version Test' button. Data from running the test will be displayed at the bottom of the Version Control Configuration panel. Errors will be shown if there are problems.

To ensure Helix QAC can retrieve a file enter a version number into the 'Test Source File Version' field. This should be a valid variant of the source file selected and then press the 'Diff Test' button. The changes in the file will be shown in the display box if the retrieve was successful.

## Using Helix QAC Without a Version Control System

If a Version Control System is not used but it is required to upload to Dashboard then the 'single' or 'single-sop' VCF script can be used. This will include file version and author information in the snapshot.



## 9 | Working with Helix QAC Dashboard

Helix QAC Dashboard is a centralized application that manipulates, summarizes, and gives wide visibility, to the results generated through Helix QAC.

For Helix QAC users, the key interactions with Helix QAC Dashboard are as follows:

- Helix QAC Dashboard can provide centralized project configurations.
- Analysis results can be uploaded to Helix QAC Dashboard, with each different set of uploaded results creating a new snapshot in Helix QAC Dashboard.
- A snapshot from Helix QAC Dashboard can be used as a baseline.
- Suppressions defined in Helix QAC Dashboard can be downloaded and applied to local diagnostics.

---

### Connecting to Helix QAC Dashboard

Data is uploaded to, or downloaded from Helix QAC Dashboard, via a Helix QAC Dashboard connection, for which you need a logon. Each user should have a unique logon for Helix QAC Dashboard, both for licensing reasons and because Helix QAC Dashboard has a permissions scheme that allows administrators to control the entitlements given to individual users (generally you will need "**Project Admin Rights**" to create a project and "**Upload**" rights to upload to an existing project). If you do not have a logon, ask your Helix QAC Dashboard administrator to create an account for you.

To log on to Helix QAC Dashboard, select the following menu option:

**Dashboard menu : Connect to Helix QAC Dashboard** (or click the  icon on the toolbar).

This brings up a connection dialog, where you should enter the address of the Helix QAC Dashboard server, and your credentials.

**Note:**

The Helix QAC Dashboard server address is the same as the one you would use to access Helix QAC Dashboard through a browser. Your Helix QAC Dashboard administrator should have the details. The default port number is 8080, but this can be different on different sites if this clashes with another application, or violates a security policy.

You can ask for the details to be remembered, to simplify your next logon. For security reasons, the password will not be stored. The dialog will store the history of previous connection details, which simplifies the logon process if you access different Helix QAC Dashboard servers for different projects.

**Note:**

Connection details are stored separately for each user on the machine. Other users will not be able view your logon credentials.

Once the logon details have been entered, click **Connect**. For QA-GUI, the connection status will be shown in the bottom right hand corner of the main screen. Two error codes are shown if the connection attempt fails: the first error code is a Qt error code; and the second error code is derived from the HTTP standard error code system.

**Note:**


Clicking on this connection status area at any time also brings up the connection dialog.

Some Helix QAC Dashboard menu items will remain disabled until a successful connection is made. If you have a successful connection and some menu items continue to remain disabled then the account you are using to connect to Helix QAC Dashboard does not have permissions to perform these actions.

If you connect from Helix QAC to Helix QAC Dashboard from a second machine, the connection from the first machine will be broken. You can, however, have simultaneous connections from Helix QAC and from a browser (which is the standard way of accessing Helix QAC Dashboard).

## Disconnecting from Helix QAC Dashboard

To disconnect from Helix QAC Dashboard you must bring up the Helix QAC Dashboard Connection dialog, by one of the following methods:

- Select the following menu option **Dashboard menu : Manage Helix QAC Dashboard Connection**.
- Click the  icon on the toolbar.
- Click on the "**Helix QAC Dashboard Connection Status**" in the bottom right hand corner of the main screen status bar.

Choosing "**Disconnect**" will logoff from Helix QAC Dashboard and terminate the connection.

## Centralizing Project Definitions

Most code projects are worked on by more than one person. For reasons of efficiency and consistency, it is recommended that one person sets up the Helix QAC project, and then distributes it to anyone else who needs it. The work done by the first person to ensure that all the correct options have been selected, does not then need to be repeated by the others.

A project definition is just a configuration file, and can be shared through a source control system, or simply by copying. However, using Helix QAC Dashboard to share these configurations has the advantage of unifying the project so that, on the Helix QAC Dashboard server, it has both the Helix QAC Dashboard project configuration and a stored Helix QAC configuration. The Helix QAC part of the Unified project can be downloaded onto a local machine. A permanent link is maintained between the local project and the project on the Helix QAC Dashboard server. This link helps with both uploading results and downloading baselines and suppressions.

**Important:**

Although, you can create multiple configurations for a project within Helix QAC, so that you can switch between ACFs, RCFs and CCTs within the one project, it is only possible to unify a project that has a **single** configuration (refer to [The Helix QAC Dashboard Menu](#)).

## Creating a Unified Project

The process starts with a single user creating a normal Helix QAC project with a single configuration. This user should add all the relevant files to the project, ensure that the options are set correctly for that particular set of code, and that the results are as expected. The user needs to be aware that the project configuration will be shared, and that the project roots should therefore be created with the full set of users in mind.

A particularly important "user" is the one performing the automated analysis that takes place on a central server, as described in [Desktop versus Server-side Working](#).

Whereas desktop users can make adjustments to a project configuration once they have downloaded it, this is not possible in Helix QAC Dashboard. You can, for example, change the Dataflow Settings and the other configuration settings for the parser components (although there will potentially be differences in analysis output when compared with that of the initial Helix QAC Dashboard snapshot and the supplied baseline/suppression).

Once you are satisfied with the project configuration, you can upload it to Helix QAC Dashboard. You must already be logged into Helix QAC Dashboard and have "Project Admin Rights" permissions. To Unify and upload the project definition select the following option:

#### Dashboard menu : Unify Project : Create

**Note:**

If the command is disabled, connect to Helix QAC Dashboard first.

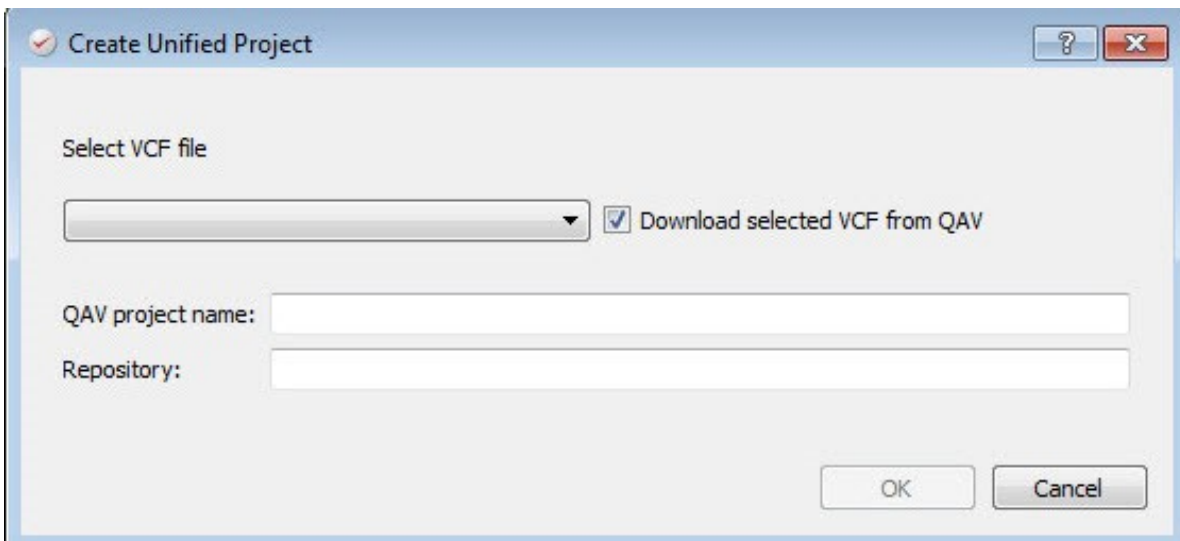
In the dialog that is shown, proceed as follows:

1. Select a VCF (Version Control Compatibility File) from the drop-down list. This file holds all the details used by Helix QAC Dashboard to connect to the version control system in order to access the source files. The VCF file selected needs to match the version control system used for the particular code project.

**Note:**

Normally, you would leave the 'Download selected VCF from Helix QAC Dashboard' check box selected. This makes the VCF file available to the local system for operations such as baselining.

2. Enter a name for the Unified project. Typically, this would be the same name as used for the code project, but it could be any name that is meaningful to the users of the project.



The screenshot shows a Windows-style dialog box titled "Create Unified Project". It features a standard title bar with a question mark icon and a close button (X). The main content area is divided into sections. The first section, "Select VCF file", contains a dropdown menu and a checked checkbox labeled "Download selected VCF from QAV". Below this, there are two text input fields: "QAV project name:" and "Repository:". At the bottom right of the dialog, there are two buttons: "OK" and "Cancel".

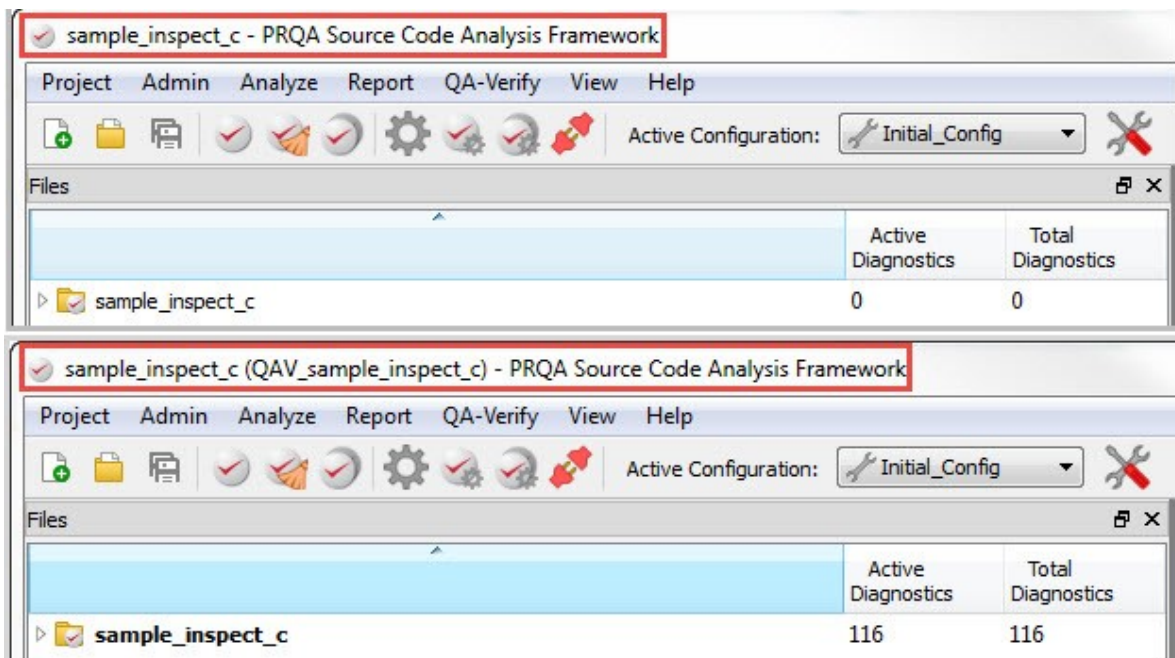
Important:

The name needs to follow the same restrictions as those supplied in [Validation](#). Note the final paragraph, which states that the Dashboard project name cannot contain any dots ('.').

3. Version control systems need additional project-specific information to access the source code. Enter this information into the Repository field. Refer to the Dashboard documentation for more details.

The act of uploading the project configuration will turn the project into a Unified project, which means that:

- A linked project will be created within Dashboard, ready for an initial snapshot upload.
- The Helix QAC project configuration will be stored centrally in Dashboard so that other users can download it.
- The project name will be supplemented on the Helix QAC title bar with the name of the Unified project in brackets. Refer to the example in Figure 9.3, which shows the name of a project before and after it was "Unified", where the name in brackets is the "Dashboard project name" supplied in Figure 9.2.



## Downloading a Unified Project

Before you download a Unified project and start using it to analyze code, ensure that the code project is in place and has all the necessary code files, and that any previous project in Helix QAC is closed.

To download it, navigate to:

**Dashboard menu : Download Unified Project : Create**

**Note:**

If the option is disabled, connect to Helix QAC Dashboard first.

In the resultant dialog, select the project that you want to download, and click **OK**.

Once the project has been downloaded, it should be ready for immediate use. However, in some cases, it may be necessary to make some adjustments to suit the local environment. For example, if the source files are in a location that is not the same as the locations of the other people working on the project, adjust the project roots as described in [Setting the Root Directory of Your Sources](#). If additional files have been added to the code project since the Unified project configuration was uploaded, the project should be updated as described in [Populating Your Project](#).

## Working on a Unified Project

When a Unified project is downloaded, all the analysis and rule settings are delivered to the local environment so that local analysis can be performed in the same way as for the original desktop project. In addition, baseline and suppression information is supplied from the Helix QAC Dashboard project so that the display of local diagnostics (including those from local changes) can be filtered.

It is now possible to amend a existing project configuration, just as for any other Helix QAC project. You can, for example, change the **Dataflow Settings** and the other configuration settings for the parser components (although there will potentially be differences in analysis output when compared with that of the initial Helix QAC Dashboard snapshot and the supplied baseline/suppression).

## Updating a Unified Project

Projects change over time, and it may well be that the project configuration held centrally in Helix QAC Dashboard becomes out of date. The configuration can be updated using the following Helix QAC option:

**Dashboard menu : Unify Project : Update**

Just as with project creation, the project should be put into a state suitable for sharing before it is uploaded.

Once it has been updated, other users can download the project configuration using the following option:

**Dashboard menu : Download Unified Project : Update**

This will overwrite their existing configuration, so, as with the initial download, they may need to make some adjustments to suit the local environment.

## Maintaining the Unified Ruleset

For changes to the ruleset it is no longer necessary to use the Helix QAC Dashboard tool ConfigGUI. Changes can be made using the **Project Properties Rule Configuration** panel (refer to [Rule Configuration](#)) and these can be propagated to Helix QAC Dashboard by updating the Unified project, see [Updating a Unified Project](#).

## Creating a Unified Project as a New Project

The recommended approach for creating a Unified project is to first create a local project, and then convert it to a Unified one. This allows you to validate the project configuration locally, before uploading it to the Helix QAC Dashboard server.

However, it is also possible to create a project as a Unified project in the first instance, from the **New Project** dialog, by selecting **Unified Project**. Additional fields then become accessible, as described in [Creating a Unified Project](#).

If this is done, the project configuration is uploaded as soon as the dialog closes, and before any source code files have been added to the project. Users downloading this configuration will need to do their own project population as described in [Extraction of Configuration Data](#).

Any changes made to your local project after the **New Project** dialog closes (including any made via the **Project Properties** dialog that opens directly following the **New Project** dialog) will not be uploaded to Helix QAC Dashboard by default. You need to upload these changes using:

**Dashboard menu : Unify Project : Update**

## Uploading Results to Helix QAC Dashboard

Analysis results can be distributed more widely by uploading them to Helix QAC Dashboard. A single set of uploaded results becomes a snapshot in Helix QAC Dashboard, representing the project status at a point in time. If results are uploaded regularly, then Helix QAC Dashboard holds a series of snapshots, one for each upload. Together they form a record of progress over a period of time.

The value of these snapshots is greatly enhanced if the results come from known versions of the source code, with a consistent set of options being used to produce those results. We have noted that a number of customers have achieved this by integrating the uploading of results as part of their server-side build automation. For this, those customers made use of the QA·CLI function described in [Upload](#).

**Note:**

It is not recommended that different snapshots are uploaded from different locations to a common, shared project. This should be done from a central location in a controlled fashion, one snapshot at a time.

Where there is no central location, as with GUI-driven projects that are shared as informal projects, snapshots can be created from full or partial contents.

**Note:**

If you wish to create what is termed a Solo project with a single snapshot that is continuously overwritten, for example, if you wish to review code that has not yet been checked in, then create the project on the Helix QAC Dashboard server **Project Helix QAC Dashboard**, and be sure to select the option **Single Snapshot Mode**. For further details, please refer to the Helix QAC Dashboard documentation.

**Caution:**

Care should be taken not to remove or overwrite a snapshot that is still needed. Create a new *Solo* project if necessary.

## Uploading via QA·GUI

Whilst uploading via server-side analysis is the recommended approach for non-Solo projects, it is also possible to upload results from QA·GUI.

**Note:**

Results can be uploaded regardless of whether or not your project is a Unified project, although you will not be able to share the results with other users as long as the project is not Unified.

There is no need to upload the whole code project in the case of a Solo project. For example, if the review is to be focused on just three or four files, then upload only those files, using the following option:

**Helix QAC Dashboard menu : Upload Results : Upload Selected Files to Helix QAC Dashboard**

However, in the case of a non-Solo project, care should be taken to upload the full project, using the following option:

## Helix QAC Dashboard menu : Upload Results : Upload Project to Helix QAC Dashboard

### Note:

To upload you must already be logged into Helix QAC Dashboard and have "Upload" permissions.

This is because, if only selected files are uploaded, this affects the inheritance aspect of the snapshot sequence representing the project history. Trend reports will be affected by the gap in data, but, more importantly, Helix QAC Dashboard suppressions may also be affected. Suppressions against diagnostics in a file are, by default, inherited by the next snapshot, and, if that file is not uploaded as part of the snapshot, then the suppressions will not be inherited by either this snapshot or any subsequent snapshots.

Refer to [The Helix QAC Dashboard Menu](#) for how to use the above options, for the **Upload Results** dialog. Normally, all you need to specify is the name of the snapshot. If the source code has been checked into the version control system, leave the **Upload Source** selection as **None**.

## Viewing a Snapshot

Once a snapshot has been uploaded, it is available for viewing within Helix QAC Dashboard. It can show additional information about functions, such as Function Structure diagrams and calls to and from the functions. Annotations can then be added against the source code and used as part of a code review process in conjunction with other users.

## Downloading a Baseline

The concept of baselines was introduced in [XML Indent Size](#). Helix QAC Dashboard will typically hold many snapshots throughout a project's history. Any of these can be used as the baseline. One approach is to select a snapshot representing a release, on the grounds that the diagnostics still present at the point of release were regarded as not significant. Another approach is to use a recent snapshot, so that the only diagnostics seen on the desktop are newly created issues in freshly written code.

To download a baseline, navigate to:

### Helix QAC Dashboard menu : Download Baseline

In the resultant dialog, select the snapshot and click **OK** to start the download.

Once the baseline has been downloaded, you can make it active at any time, from the **Project Properties' Baseline** tab.

Select the **Use Baseline Diagnostics Suppression** check box and the **Baseline by Helix QAC Dashboard Snapshot** radio button. Once the baseline has been set, those diagnostics that appear both in the baseline snapshot, and in your own analysis results, will be suppressed ([Baseline Diagnostics Suppression](#)) describes how these suppressions are displayed).

### Note:

Baselines can be downloaded as often as required. Downloading a new baseline will overwrite the old one.

## Downloading Suppressions

Individual diagnostics can be hidden from view through suppressions. There are two main mechanisms available:

- Insert instructions into comments in the source code (described in the documentation for the individual analysis tools).

- Define suppressions against the results held in Helix QAC Dashboard (described in the Helix QAC Dashboard documentation).

Suppressions defined within the source code are applied automatically. Suppressions defined within Helix QAC Dashboard need to be downloaded from Helix QAC Dashboard before they can be applied by Helix QAC.

To download suppressions, navigate to the following option:

#### **Helix QAC Dashboard menu : Download Suppressions**

In the resultant dialog, select the snapshot and click OK to start the download. The suppressions downloaded are those that cause a diagnostic to be suppressed within the chosen snapshot. Because of the 'carry forward' functionality of Helix QAC Dashboard suppressions, the actual definition of the suppression may come from an earlier snapshot.

Once the suppressions have been downloaded, they will automatically be applied when you view analysis results. Even if you have edited your local source code since the snapshot was uploaded, Helix QAC can identify instances where the same diagnostic has been generated in a slightly different location. The suppression in the snapshot will still be applied within your own code.

#### **Note:**

In order for this operation to work, the snapshot will have had to be uploaded to Helix QAC Dashboard in conjunction with the source.

New suppressions are typically created in Helix QAC Dashboard on a regular basis. To obtain the latest suppressions, simply download a more recent snapshot. However, if much of the code that you have locally is older than the latest check-in (which can happen if you need stability in the code on which you are not actually working), you should select a snapshot for which the source code is most similar to your own. This is because Helix QAC Dashboard only carries forward suppressions while the diagnostic is still present. A suppression needed for your older code may no longer be present in the latest "fixed" code.

#### **Note:**

The Suppression Report (refer to [Standard Report Types](#)) will only show justification text for comment-based suppressions in the source code. Suppressions created in Helix QAC Dashboard, however, will appear in the report with no justification. Similarly, when uploading results to Helix QAC Dashboard, deviation records will not be created for justification text associated with comment-based suppressions in the source code.



## 10 | Cross-Module Analysis (CMA)

Cross-Module Analysis (CMA) applies only to C/C++ projects.

With C and C++, source files can be compiled independently of one another. A source file will include header files to ensure that all the necessary information for compilation is available, without needing to look at other source files. A source file together with its included header files is called a translation unit. Just like the compiler, Helix QAC for C and Helix QAC for C++ analyze one translation unit at a time.

While single file analysis can provide most of the required diagnostics, some additional checks necessitate the files being considered together. These checks include finding duplicate or conflicting declarations; finding unused code; reporting on potential name confusion; and detection of recursion. This type of analysis is what is meant by CMA. A component called RCMA can be used to carry out these checks (refer to [Analyzing Your Project](#)). Just like a linker within a build, RCMA will operate on all the files within a Helix QAC project.

### Single and Multi-Project CMA

C and C++ allow executables to be built either in a single step, or in multiple steps e.g. by adding libraries into the definition of the executable. The libraries and the executable are all separate code projects, with the build scripts set up to build them sequentially, rather than as a single unit.

Helix QAC projects normally correspond to code projects. If the executable is built from several libraries, there would often be a Helix QAC project for each library, as well as a Helix QAC project for the executable itself.

In the context of CMA, it is important to work with what will become a build-able entity, for example a final executable with all libraries linked. RCMA needs to see all the code that will be made into the executable, including code within the libraries that are built into the executable. This means that RCMA needs to work with several Helix QAC projects. To allow for this, CMA projects can be created that contain many individual Helix QAC projects. Section [CMA Project Editor](#) describes how to set up a CMA project.

Not all C and C++ executables are built on several steps. Often, there is just a single code project that contains all the code that goes into the executable. There would then just be a single Helix QAC project corresponding to this code project, which contains all the code files. It is possible to create a CMA project that contains just the one Helix QAC project. However, to keep things simple, Helix QAC provides the ability to run the CMA checks against a single Helix QAC project, rather than a CMA project. Running RCMA against a Helix QAC project is called "single-project CMA"; running RCMA against a CMA project is called "multi-project CMA".

### Single-Project CMA Analysis

Single-project CMA analysis can be performed against a Helix QAC project by adding the RCMA component to the toolchain "Common C and C++ Components" (refer to Section [Common C and C++ Components Toolchain](#)).

You can then configure RCMA for the needs of your analysis, before selecting:

**Analyze menu : Run CMA Analysis**

**Note:**

RCMA makes use of information calculated by Helix QAC for C and Helix QAC for C++. All files must have been analyzed by one of these components before RCMA can do its own checks.

## Multi-Project CMA Analysis

Multi-project CMA analysis is performed against a CMA project. See Section [CMA Project Editor](#) for details of how to create a CMA project.

To run CMA analysis, select:

### Analyze menu : Run CMA Analysis

You need to have an active Helix QAC project to be able to use the command. It brings up a dialog, listing the available CMA projects. Choose one and click OK to start the analysis.

The results of the CMA analysis will be stored against the current project, and can be viewed alongside the normal analysis results for that project. If you open a different project, you will not be able to view the CMA results just calculated. Instead, perform a CMA analysis against the new project to see the CMA results.

If, for example, a CMA project X contains Helix QAC projects A and B, then:

1. Open project A, and run CMA analysis against CMA project X. The results will be visible from project A.
2. Open project B, and no CMA results will be visible. Analyse CMA project X from B, and the results will then become visible from B.
3. If the results are then cleaned from project B, this will have no effect on the results still stored against project A.

If a single Helix QAC project belongs to several CMA projects, you will need to analyze each of those CMA projects to obtain the full set of results for the Helix QAC project.

## CMA Project Editor

CMA analysis operates across translation unit boundaries and checks for issues such as duplicate definitions, incompatible declarations and unused variables.

To open the **CMA Project Editor** that is used to manage the CMA projects, select:

### Project menu : Open CMA Project Editor

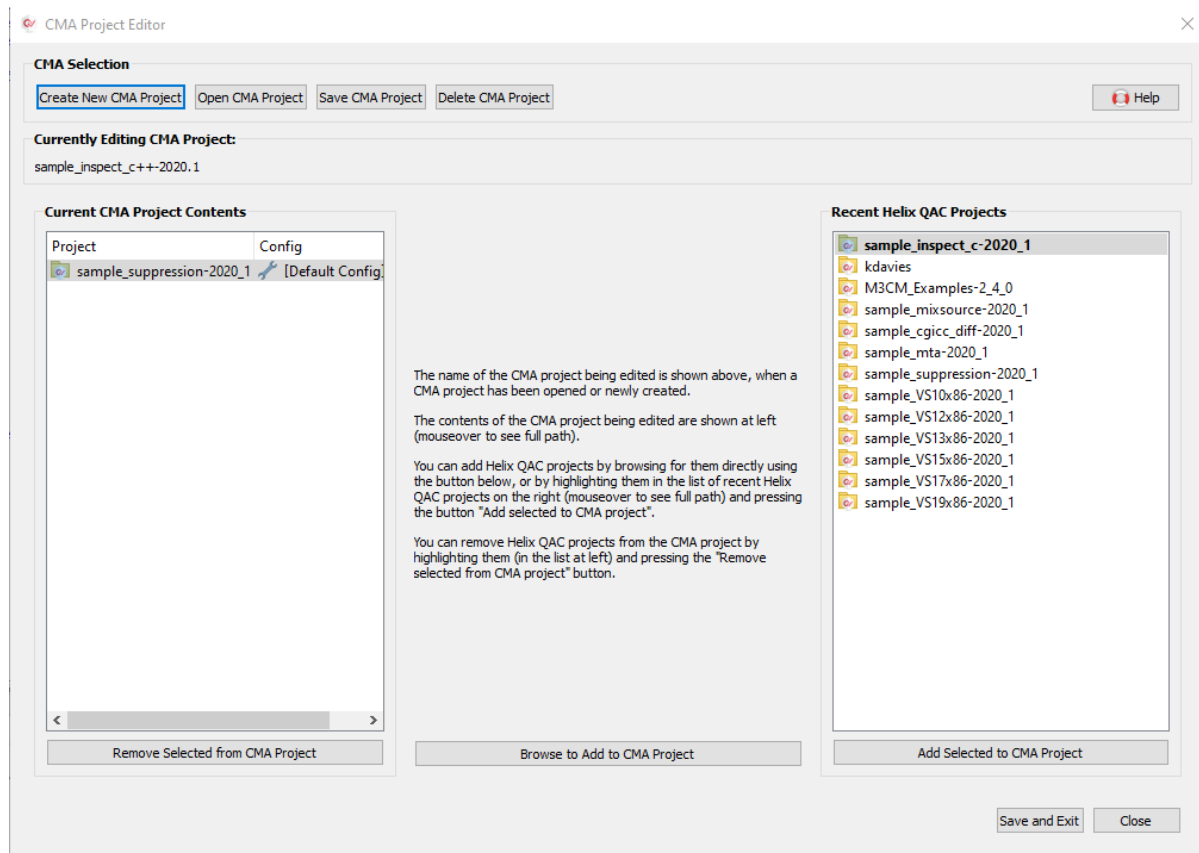
This displays the **CMA Project Editor** panel.

From this panel, you can do the following:

- Create a new CMA project.
- Open an existing CMA project.
- Add Helix QAC projects to, or remove - projects from, the current CMA project.
- Save the current CMA project.
- Delete an existing CMA project.

When you click the "Create New CMA Project" button, you are invited to enter the name of a new project, which then becomes the "Currently Editing CMA Project". Alternatively, when you click the "Open CMA Project" button, you are invited to select from a list of existing CMA projects - the one selected then becomes the "Currently Editing CMA Project".

To add a Helix QAC project to the "Currently Editing CMA Project", either select it in the list of "Recent Helix QAC Projects" on the right and click the "Add Selected to CMA Project" button to



move it to the "Current CMA Project Contents", or else click the "Browse to Add to CMA Project" button to search for the Helix QAC project folder to add to the "Current CMA Project Contents". Only C or C++ projects may be added.

To remove a Helix QAC project, select the desired project in the "Current CMA Project Contents" and click the "Remove Selected from CMA Project" button.

Now that projects can contain multiple configurations (refer to Section [Projects with Multiple Configurations](#)), an added project will initially be mapped with the **Default Configuration**. Any other configurations for that project are selectable by double-clicking on the current configuration (listed under "Config") and selecting the required configuration from the drop-down list. So, if, for example, you want to perform two analysis runs with different configurations (maybe different versions of Helix QAC for C), you have two possible ways of doing this:

- In the **CMA Project Editor**, ensure that the **Default Configuration** is selected. Then, in the **Configuration Manager** (refer to Section [Configuration Manager](#)), set the **Default Configuration** for each individual Helix QAC project, and perform the CMA analysis. Next, for each project, set a different **Default Configuration** and rerun the analysis.
- In the **CMA Project Editor**, set the specific configuration for each project and then perform the CMA analysis. Then, back in the **CMA Project Editor**, set another configuration for each project and rerun the analysis.

The CMA project can be saved at any point during the change process, by clicking "Save CMA Project". To save it once all changes have been completed, click the "Save and Exit" button.

If a CMA project is not required, it can be removed using the "Delete CMA Project" button. From the resultant dialog, select the CMA project that is to be removed, and then confirm the deletion.

## Viewing CMA Data

Referring to Section [Multi-Project CMA Analysis](#), you will recall that the current CMA results are stored against the project for which the CMA analysis was last run. They are stored under the "CMA" branch in the Files panel.

Clicking on the "CMA" branch will cause the messages to be displayed in the **Analysis Results/Diagnostics** panel (refer to Section [Whole Project Analysis Hard Errors Panel](#)). For any given message, click on the sub-messages in the panel to see the locations in the source code Editor (refer to Section [Viewing Diagnostic Messages In-Line](#)).

**Note:**

Some diagnostics calculated by RCMA relate to multiple locations. For example, if RCMA detects a duplicate declaration, it will generate a top-level diagnostic highlighting the problem, plus sub-diagnostics that identify the locations of each duplicate. In this event, the top-level diagnostic does not have a location of its own.

## Cleaning CMA Data

To clean the diagnostics generated from RCMA, you need to clean the particular Helix QAC project referenced in the CMA project Editor (refer to Section [CMA Project Editor](#)) for which the CMA analysis was last run. Once this project is open, select the following option:

**Analyze menu : Clean project**

**Caution:**

Cleaning a project will remove all diagnostics for the project, not just the CMA results.

## 11 | Reports

### Introduction to reports

The Reports functionality inside Helix QAC provides several standard analysis and metrics reports. It uses a plug-in architecture which allows you to add additional reports licensed from Programming Research Ltd and to write your own customized report generators if you wish.

### Standard Report Types

Reports can be generated using the following report types:

<b>CRR</b>	The Code Review Report summarizes metrics and messages from files, functions and classes. It can also display some code visualizations, includes, calls, relations and function structure. It provides a broad overview of the code.
<b>HMR</b>	The HIS Metrics Report produces metrics according to the HIS (Hersteller Initiative Software) specifications - a group of leading automotive manufacturers whose goal is the production of agreed standards within the areas of software modules for networks, process maturity, software test & tools, and programming of ECUs (Engine Control Units).
<b>MDR</b>	The Metrics Data Report generates an XML file that you can use as a source of metrics data for your own further examination.
<b>RCR</b>	The Rule Compliance Report contains data on violations of rules that are specified in a Helix QAC project's Rule Configuration File.
<b>SUR</b>	The Suppression Report provides information on message diagnostics that have been suppressed during analysis.

The Suppression Report provides information on message diagnostics that have been suppressed during analysis.

For the dialog used to run the reports, please refer to [The Report Menu](#). If you select the 'Ignore Dependencies' option, then dependency checking is skipped and reports are able to be produced even for those files for which analyses remain out of date, for example because RCMA is in the toolchain.

**Note:**

If you do select the 'Ignore Dependencies' option, then you should be aware that, because the dependency checking has not been run (and subsequent analysis has also not been run), the reports may not be 100% complete. For this reason, all reports generated using this option are marked as 'Draft' in the output.

The QA-CLI equivalent of this option is

```
qacli report -P . -t <report_type> --ignore
```

(refer to Section [Report](#)).

By default reports are written to the Helix QAC project folder,

```
prqa/configs/<config_name>/reports
```

with names in the format,

`<name_of_project>_<Report Name>_<datestamp>_<timestamp>.<type>`

such as,

`myproject_HMR_06032018_155447.html`

and can be copied and shared with other users. The output location can be overridden by either typing a directory pathname or browsing to a location using the folder icon in the dialog.

**Note:** You must have write access to this directory. If the directory does not exist it will be created.

**Note:**

Failure during analysis compromises reports accuracy, as the crashes must be fixed for accurate analysis and consequently reporting

## Components of Function Structure

The Code Review Report includes function structure diagrams which show the control flow structures within source code. Decisions (if, switch, and the condition part of a loop) are displayed as forks in the structure. The corresponding join, further to the right, indicates where the paths rejoin. Backward arcs, caused by loops, are shown as dotted lines.

The function structure diagrams show the following code structures:

- [Straight Code](#)
- [If](#)
- [If-Else](#)
- [Switch](#)
- [While Loop](#)
- [For Loop](#)
- [Nested Structures](#)
- [Break in a Loop](#)
- [Return in a Loop](#)
- [Continue in a Loop](#)
- [Unreachable Code](#)

Each component progresses from the left to the right as the control flow would progress through the source code.

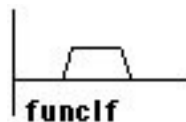
## Straight Code



```
static int code = 0;
void funcStraight (void)
{
    code = 1;
}
```

There is only one path through this particular function, which is represented as lying along the x-axis.

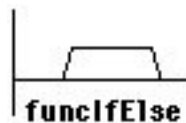
## If



```
static int code = 0;
void funcIf (void)
{
    if (code > 0)
    {
        code = 1;
    }
}
```

This function has two paths. The first path is through the if statement and is shown by the raised line. The second path is where the if condition is false and is represented by the x-axis.

## If-Else



```
static int code = 0;
void funcIfElse (void)
{
    if (code > 0)
    {
        code = 3;
    }
    else
    {
        code = 4;
    }
}
```

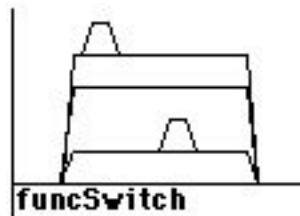
```
}  
}
```

This function has two execution paths. The first path is the if sub-statement represented by the raised line. The second path is the else sub-statement represented by the x-axis.

**Note:**

The body of this structure is longer than the body of the if statement. If the two arms of the if-else were not straight line code, the body of the if branch would appear at the left hand end of the raised line and the body of the else branch would appear to the right of the lower line.

## Switch



```
static int code = 0;  
void funcSwitch (void)  
{  
    switch (code)  
    {  
case 1:  
    if (code == 1)  
    {  
        /* block of code */  
    }  
    break;  
case 2:  
    break;  
case 3:  
    if (code == 3)  
    {  
        /* block of code */  
    }  
    break;  
default:  
    break;  
    }  
}
```



In the switch statement, the x-axis represents the default action, and each case statement is shown by a raised line. The two briefly raised lines represent the if statements within case 1 and case 3.

The diagram shows how the if statements are staggered. The if of case 1 is shown on the left and the if of case 3 is shown on the right.

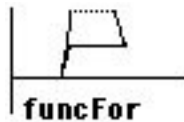
## While Loop



```
static int code = 0;
void funcWhile (void)
{
    while (code > 0)
    {
        --code;
    }
}
```

In the *while* loop, the x-axis represents the path straight through the function as though the *while* loop had not been executed. The solid raised line shows the path of the *while* body. The dotted line shows the loop to the beginning of the *while* statement.

## For Loop



```
static int code = 0;
void doSomethingWith(int);
void funcFor (void)
{
    for (int i = 0; i > code; ++i)
    {
        doSomethingWith(i);
    }
}
```

The for loop is similar to the while loop as for loops can be rewritten as while loops. For example, funcFor in the above example could be written as:

```

void funcFor (void)
{
    int i=0;
    while (i > code)
    {
        /* body */
        ++i;
    }
}

```

## Nested Structures



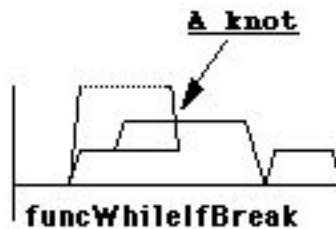
```

static int code = 0;
void funcWhileIfElse (void)
{
    while (code > 0)
    {
        if (code == 1)
        {
            code = 0;
        }
        else
        {
            --code;
        }
    }
}

```

This is an *if-else* contained within a *while* loop. The first solid raised line represents the *while* loop while the inner raised solid line represents the *if-else* loop. Other function structure components can be similarly nested.

## Break in a Loop



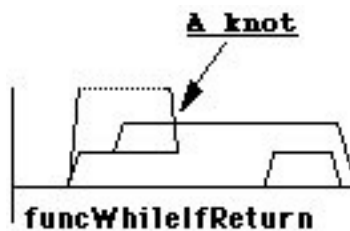
```

static int code = 0;
void funcWhileIfBreak (void)
{
    while (code > 0)
    {
        if (code == 3)
        {
            break;
        }
        --code;
    }
    if (code == 0)
    {
        code++;
    }
}

```

The break jumps to the end of the while statement and causes a knot (this is where the control flow crosses the boundary of another statement block, indicating unstructured code). In this case, once the break has jumped to the end of the while statement, the next part of the program - the if statement - is executed.

## Return in a Loop



```

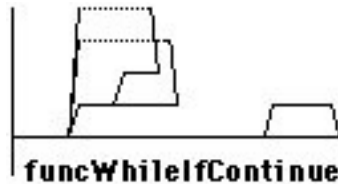
static int code = 0;
void funcWhileIfReturn (void)
{
    while (code > 0)
    {
        if (code == 3)
        {
            return;
        }
        --code;
    }
    if (code == 0)
    {
        code++;
    }
}

```

```
}  
}
```

The *return* statement causes the program to jump to the end of the function. This jump breaks the control flow and causes a knot in the code.

## Continue in a Loop



```
static int code = 0;  
void funcWhileIfContinue (void)  
{  
    while (code > 0)  
    {  
        if (code == 3)  
        {  
            continue;  
        }  
        --code;  
    }  
    if (code == 0)  
    {  
        code++;  
    }  
}
```

The *continue* statement causes the program to jump back to the beginning of the while loop.

## Unreachable Code



```
static void funcUnReach (int i)  
{  
    if (i)  
    {
```

```

        i = 1;
    }
    goto Bad;

    if (i)
    {
        i = 2;
    }

    Bad:
    if (i)
    {
        i = 3;
    }
    return;
}

```

The raised red section represents code that is unreachable.

The structure and approximate position of the unreachable code is shown. It occurs after the first *if* condition and before the last *if* condition. Its position above the main structure shows that the control flow misses the middle *if* condition.

## Custom Report Plug-ins

You can write report generators of your own, typically Python scripts, which will be fully integrated into Helix QAC if you follow certain conventions.

Generators should be located in the `report_plugins` subfolder of the Helix QAC installation folder, and should generally have names of the form `Three_Capitalized_Words.py` (but see [Report Generator Naming](#) for more details).

Scripts may be written in Perl with a `.pl` extension, but you must install and configure Perl yourself so that a simple "perl" command will invoke it. A file with any other extension (or none) will be passed straight to your operating system's command interpreter (eg `cmd.exe` for Windows, `bash` for Linux, etc).

Helix QAC will invoke any report generator twice, once to determine its requirements, and once to actually generate the report when those requirements have been processed. The first invocation will be with the single parameter `"-i"`, and the generator is expected to respond by printing the following:

```

data: xml
[ require: project ]
[ require: cma ]
[ require: violations ]
[ require: suppresssions ]
[ require: rcf ]
[ require: *cf ]
[ require: umsg ]

```

```
[ require: qaf ]
format: <output type>
[ acronym: <TLA> ]
```

The 'data' line defines the input format and must be 'xml'. (No other formats are presently supported.)

The optional 'require' lines determine the information that will be available to the report. The first four ('project', 'cma', 'violations', 'suppressions') will affect the contents of the main XML file, while the others will provide access to additional files which may be needed.

'rcf' requests a copy of the Rule Control File be placed in the report output directory and provided as a "-r" parameter to the second invocation. If the copying should fail for any reason, a warning message is logged and a reference to the original is provided instead.

'\*cf' requests that all the configuration control files - currently the Rule Control File and Analysis Control File - be provided as parameters ("-r" and "-a" respectively). Note that these are references to the originals and no extra copies will be made, contrary to plain 'rcf' above. You should make your own copies if you need them.

'umsg' requests that the file containing any user-defined messages be provided as a "-u" parameter. This file is common to all projects, but unique per installation and/or user of Helix QAC.

'qaf' requests that the Helix QAC installation folder be provided as a "-q" parameter.

The 'format' line should typically be 'html', 'xml', 'txt', etc, and will be used as the extension of the output file.

Finally, 'acronym' if present should be derived according to the rules in [Report Generator Naming](#). The effect of an inconsistent acronym is not defined.

The second invocation will have multiple parameters defining the data inputs, output and other options:

```
-d <xml input file>
-o <output file>
[ -r <rcf> ]
[ -a <acf> ]
[ -u <umsg> ]
[ -q <qaf> ]
[ -s ]
```

These should be obvious in the light of the above description, except for "-s". This means that dependency checking has been skipped and the results may not be 100% reliable. In such a case, the report should indicate that status, typically by appending "(Draft)" to the title and providing an explanatory note.

**Note:**

The precise format of the XML results file is not explicitly documented and may change between releases of Helix QAC. Examining the built-in reports is currently the best way to gain sufficient understanding to write your own.

You may notice that the standard Python scripts all support long parameter names as well as short ones, but Helix QAC will always use the short forms only. Long names are provided purely as a convenience for using the scripts independently and user-written scripts are under no obligation to provide them.

## Report Generator Naming

The report generators are located in:

```
<Helix QAC installation dir>/report_plugins
```

Any file in here will be used as a report generator, and can be written in any language (although the existing ones are written in Python). In the GUI, the report will be identified by taking the filename, stripping any extension and then replacing any underscores with spaces. For example:

```
Custom_Code_Report.py
```

will be presented as:

```
Custom Code Report
```

For the command line, the report is identified by a three-letter abbreviation formed from the initial character of each word. For example:

```
qacli report -P . -t CCR
```

If the name contains fewer than three words, extra letters are taken (and capitalized) from the first word, and conversely, if it contains more, the abbreviation is formed from the first, second and last words.

Abbreviations must be unique, but this is not explicitly checked. If you have two names which resolve to the same abbreviation, an arbitrary choice will be made without warning.

## Improving Python Performance

The standard Python package includes modules to read and write XML, but a higher performance option 'lxml' is available, and on large reports the generation time differences can be substantial. If this is installed, the standard report generators will use it automatically, and of course, you can copy the technique for your own custom reports.

`lxml` is a Python wrapper around the `libxml2` library. It is compatible with the standard Python `ElementTree` module, which is used by default when `lxml` is not available.

## lxml Installation

### Linux

All popular distros have the Python module and underlying library available from their default repositories, and the lib is very likely installed already.

For example:

```
aptitude install python2.7-lxml
```

or,

```
dnf install python2-lxml
```

Alternatively, you can obtain it from the Python Package Index (PyPI):

```
pip install lxml
```

And, of course, most distros have a GUI software package manager as well as the command line.

### Windows

It's easiest to use pip (as above) on Windows, but pip itself may need to be installed (or upgraded) first. See <https://pip.pypa.io/en/latest/installing/> for more information.

The Windows versions of lxml include their own internal copy of the libxml2 library.

*Any*

Alternatively, the lxml packages (Python 'wheels') for different OSes can be obtained here for manual installation:

<https://pypi.python.org/pypi/lxml>



## 12 | QA-CLI

### Introduction to QA-CLI

The QA-CLI tool is the command line interface to Helix QAC.

The tool is designed to be used for integrating Helix QAC with build servers, but can also be used on a desktop.

It features a command line interface of the form:

```
qacli <subcommand> [options...]
```

where related features are grouped under a subcommand, not unlike most version control system interfaces<sup>1</sup>.

Features are grouped under the following subcommands, which can be viewed by issuing the `qacli -h` command:

admin	Manage Helix QAC, CMA projects and the installation.
analyze	Run analysis on existing projects.
baseline	Manage baseline settings on a project.
export	Export settings.
help	Provide help information.
import	Import settings.
pprops	Configure the Project Properties
report	Generate analysis reports.
sync	Synchronize files in a project.
upload	Send results to external systems.
view	View analysis results.

Verbose usage information on the options for each subcommand can be accessed by issuing the following command:

```
qacli <subcommand> --help
```

**Note:** The shortcut for the above is `qacli <subcommand> -h`.

Version information for the installation can be found using `-v` or `--version`, for example:

```
qacli --version
```

There are two options that appear across many subcommands: `--qaf-project` (or `"-P"`) and `--config` (or `"-K"`).

---

<sup>1</sup>Each subcommand and its options conform to the GNU/posix command line interface standard, chosen because of its widespread adoption and maturity for such use cases.

The first option is "--qaf-project" and its shortcut is "-P". Its argument is the path to the directory of the Helix QAC project with which you wish to work. For all subcommands except "admin", this option defaults to the current working directory.

The second is "--config" and its shortcut "-K". This is used to select a configuration for projects with multiple configurations (refer to [Projects with Multiple Configurations](#) for further details).

## Common Error Messages

### < n > success and < m > failures

Typically, when an operation has been completed that involves several files, such as an analysis of an entire project, a summary message like the above is generated. The message gives the total number of successful and failed operations for every file. If there are any failures, then the message also highlights which logfile to consult for further details. For example:

```
0 successes and 1 failure.
```

```
See log for more details: /home/cartman/.config/Perforce/Helix-QAC-
2020.1/app/logs/Helix-QAC_20200117T102642_17976.log
```

### < file > --no results for this files in the database

When performing a view operation, the above message may be seen. The message means that the file has not been analyzed.

### < file > --zero diagnostics found

This message indicates that the file has been analyzed, but there are no suitable diagnostics generated.

## Progress Output

For some files, a CLI command may take a considerable time to process. Where the underlying component provides feedback to the CLI, progress will be output with a standard *Progress Line*. The line will begin with the text "Progress (<label>)", where "<label>" indicates the component/subsystem providing the feedback, with one dot (".") being added for every 3% of progress. A maximum of 35 dots will be output. When complete, the line will end with "done". For example:

```
/home/frodo/cgicc_diff_clean/src/diff/version.c:qac:0:1:1
/home/frodo/cgicc_diff_clean/src/diff/util.c:qac:0:2:0
/home/frodo/cgicc_diff_clean/src/cgicc/CgiEnvironment.cpp:qacpp:0:1:2
/home/frodo/cgicc_diff_clean/src/cgicc/CgiUtils.cpp:git push:qacpp:0:2:1
/home/frodo/cgicc_diff_clean/src/cgicc/Cgi.cpp:qacpp:0:3:0
Progress (CMA): ..... done
```

Each output line is colon-separated and uses the following format:

```
<pathname>:<component>:<return code>:<# files analyzed>:<# files yet to be analyzed>
```

where:

- `pathname` is the absolute pathname of the file that was analyzed.
- `component` is the abbreviated name of the component (for example `qac`).
- `return code` is the code returned from the analysis by the component. A non-zero value indicates an error.
- `# files analyzed` is the incrementing total of the number of files analyzed so far by the CLI command.
- `# files yet to be analyzed` is the decrementing total of the number of files that remain to be analyzed by the CLI command. This number will count down to zero.

Any whole project analysis components that provide progress feedback (such as RCMA) now also output messages of progress notification. These mid-analysis, interim messages do not contain a return code, for example:

```
/home/frodo/single_project_rcma:rcma::1:125
/home/frodo/single_project_rcma:rcma::56:70
/home/frodo/single_project_rcma:rcma::100:26
```

However, on completion of the analysis, the return code will be populated as usual, for example:

```
/home/frodo/single_project_rcma:rcma:0:126:0
```

## Filelists

Various CLI commands can take a filelist as an argument, for example:

- `qaccli admin -F | --add-files <filelist>`
- `qaccli admin -D | --remove-files <filelist>`
- `qaccli analyze -F | --files <filelist>`
- `qaccli sync -t BUILD_LOG`
- `qaccli view -F | --files <filelist>`

A `filelist` is a text file containing a list of the pathnames specifying the files to which the command applies, with one pathname per line. The pathnames can be either relative to the current working directory or absolute.

Normally, a `filelist` will be encoded in the local code page for the machine (for example Code Page 850 for a typical Western Europe Windows machine, or Code Page 932 for a typical Japanese Windows machine). However, other encodings are now supported so that, for example, a `filelist` can now be encoded in UTF-8.

### Note:

If you select a non-native encoding format, ensure that you set a BOM (Byte Order Mark), to ensure that the correct encoding is used when the filelist is processed.

## Bash command line completion for QACLI

Bash command line completion is functionality through which bash helps users by presenting possible options when the user presses the **tab** key enabling them to enter commands more efficiently,

To enable it *source* the provided script in one of your bash startup scripts (`.bashrc` for example): **source <Helix Install Dir>/scripts/qaccli-completion.bash** and now your QACLI experience will be more user-friendly.

Press **tab** and it should auto-complete for most fixed items and provide files and directories for relevant commands. It will even be capable of more complex completion, for example for CMA projects.

## Admin

The QA-CLI *admin* subcommand provides features for creating and managing projects and for managing the Helix QAC installation as a whole.

Verbose help on the command can be accessed as follows:

```
qacli admin -h
```

## Installation Management

All of the following settings are per-user settings (i.e. they will not apply to a different user on the same machine).

To set the address of a Helix QAC license server from where the Helix QAC system should look to retrieve a license:

```
qacli admin --set-license-server <port@host|host[:port]>
```

where the argument refers to an application layer address, for example, 5055@192.168.1.10, 5055@myhost or 192.168.1.10:5055

Helix QAC is packaged with sample projects and a sample configuration, as well as compatibility configurations for various compilers. When you start Helix QAC for the first time, a copy of this data is made in what is called the user data area. The default location of this area is dependent on your desktop environment:

- Windows: %LOCALAPPDATA%\Perforce\Helix-QAC-<version>
- Linux: \$HOME/.config/Perforce/Helix-QAC-<version>

Due to the nature of this data, and the fact that it can be modified and extended by the user, the data is not deleted during uninstallation. If you wish to remove it, or you wish to have Helix QAC re-create it the next time it runs, issue the following command:

```
qacli admin --remove-user-data
```

This will delete all contents from the user data area, so be careful that you do not inadvertently delete something that you wish to retain.

### Note:

You can define a location for the user data area. See [User Data Location Management](#) for further details.

The various Helix QAC interfaces can be presented in different languages. In a default set-up the language is set as 'Auto', which is based on the machine locale and translations that are available. If no suitable translations (for the locale) is available, the default language is en\_US.

The following command allows the user to change the specified language:

```
qacli admin --set-language <locale>
```

The locale is typically a string describing the language. For backwards compatibility and convenience, several aliases are provided:

- Auto == Most appropriate language based upon machine locale
- EN == en\_US

- US == en\_US
- JP == ja\_JP

To query the list of available languages, or to display the current language, the `--list-languages` option can be used:

```
qacli admin --list-languages
```

If you wish to set a specific number of CPUs to be used for analysis, use the following command:

```
qacli admin --set-cpus <integer>
```

Helix QAC projects maintain configuration data that is grouped into various configuration files. This is so that the same configuration file can be used when creating many projects that may require a similar configuration for certain features. Each configuration file type has a directory in the user data area where they are maintained, and this is where the Helix QAC interface searches for files to present as choices when a project is being created or modified. The following command is used to create a new configuration file of a specific type from a template:

```
qacli admin --create-config-file <acf|rcf|vcf>
```

The configuration file types that can be created correspond to:

- **acf** Analysis Configuration File
- **rcf** Rule Configuration File
- **vcf** Version Control Compatibility File

## Helix QAC Project Management

A Helix QAC project is identified by the directory in which it resides and it does not have a specific name that identifies it, other than the directory path. A project contains configuration files that contain related settings. Creating a new Helix QAC project is a case of specifying the directory where the project should be created and the configuration files that it should contain.

A Helix QAC project consists of:

- At least one CCT (Compiler Compatibility Template)
- Only one RCF (Rule Configuration File)
- Only one ACF (Analysis Configuration File)
- Zero or one VCF (Version Control Compatibility File)
- Optional configuration name

The following command creates a new Helix QAC project:

```
qacli admin --qaf-project-config --qaf-project <directory> --cct <path> --rcf <path> --acf <path> [--vcf <path>] [--config <config-name>]
```

For the creation of a new project, the following constraints apply:

- A Helix QAC project must not exist in the specified directory location.
- The specified directory need not exist but its parent directory must exist.
- The `--rcf` option must be specified and its argument must be a path to a valid Rule Configuration File.
- The `--acf` option must be specified and its argument must be a path to a valid Analysis Configuration File.

- The '--cct' option must be specified at least once in the command and its argument must be a path to a valid Compiler Compatibility Template.

In addition:

- The '--cct' argument may be specified more than once in order to add support to a project for more than one compiler (for example, a C compiler and a C++ compiler). Other optional configuration files may also be specified.
- The '--vcf' option can be used to specify a path to a valid Version Control Compatibility File (for retrieving version control information that is used for baselining and performing uploads to a Dashboard server).

The same command as that specified above to create a new project, can be used with an existing Helix QAC project to add and replace configuration files, with the following default behavior:

- If an RCF is specified, the RCF in the project is replaced with the one specified.
- If an ACF is specified, the ACF in the project is replaced with the one specified.
- If a VCF is specified, the VCF in the project, if one exists, is replaced with the one specified. Otherwise, the specified VCF is added.
- If a CCT is specified, it is added to the project. No CCTs currently in the project are removed. A project may have multiple CCTs, but only one may be active for each source language at any one time. The changing of the active status of CCTs must be carried out using the QA·Project Properties interface (command: `qapp <projectpath>`).

#### Note:

- The above behavior can be further controlled using the --update-method option (refer to [Upgrading a toolchain component in a project](#)).

Internally, Helix QAC maintains paths as relative to source code roots, as described in [Setting the Root Directory of Your Sources](#). The default source code root is SOURCE\_ROOT. This location can be set by the following command:

```
qacli admin -P <project-path> --set-source-code-root <directory>
```

Where `directory` can be an absolute path or a relative path from the project root directory.

Various Root Directories can be set to aid project portability. To set use the following command giving the unique name of the root and the directory that it references, for example:

```
qacli admin -P <project-path> --set-project-root BOOST_INC --path /opt/boost_168/include
```

```
qacli admin -P <project-path> --set-project-root QT_INC --path /opt/qt-484/include
```

To unset the name of the project root, use the following command:

```
qacli admin -P <project-path> --unset-project-root <name>
```

Root Directories can be set explicitly for project-level configuration using the switch --project-level. For more details, refer to [Setting the Root Directory of Your Sources](#):

```
qacli admin -P <project-path> --set-project-root <name> --path <path> --project-level
```

To list the root paths defined in the project, use the following command:

```
qacli admin -P <project-path> --list-project-roots
```

## Configuration File Management

To list the configuration files that are used within a project, use the following command:

```
qacli admin -P <project-path> --list-config-files
```

**Note:**

Remember, the ACF, RCF and VCF files will have default names; *project.acf*, *project.rcf* and *project.vcf*.

To set the default configuration file (to be used when no other configuration is specified using QA·CLI), use the following command:

```
qacli admin -P <project-path> -K <configuration-name> --set-default-config
```

**Note:**

This command is mainly for use with projects having multiple configurations. If you remove the configuration that is currently the default, then a new default will be selected automatically. If you rename a configuration that is currently the default, the default will also move.

To retrieve the list of configurations for a project, use the following command:

```
qacli admin -P <project-path> --list-configs
```

To copy a configuration file that is used within a project, use the following command:

```
qacli admin -P <project-path> -K <source-configuration-name> --copy-config <new-configuration-name>
```

The new configuration can then be tailored as required, for example using:

```
qacli admin -P <project-path> -K <configuration-name> --qaf-project-config [-A <acf-file>] [-C <cct-file>] [-N <ncf-file>] [-R <rcf-file>] [--update-method <replace|update|merge>] [-V <vcf-file>]
```

To rename a configuration file, use the following command:

```
qacli admin -P <project-path> -K <old-name> --rename-config <new-name>
```

To remove a configuration file, use the following command:

```
qacli admin -P <project-path> -K <configuration-name> --remove-config
```

**Note:** For the above options, refer also to [Multiple Configurations with Helix QAC·CLI](#)

Sometimes it is necessary to modify a configuration file quickly within a Helix QAC project. To do so, the project and configuration file type must be specified. Issue the following command:

```
qacli admin -P <project-path> --edit-config-file  
<acf|rcf|cct-c|cct-cpp> [-K <config-name>]
```

where the arguments are as follows:

- **acf:** The project's Analysis Configuration File.
- **rcf:** The project's Rule Configuration File.

- **cct-c**: The project's 'active' Compiler Compatibility Template for the C language.
- **cct-cpp**: The project's 'active' Compiler Compatibility Template for the C++ language.

To import the complete user configuration from a previous installation of Helix QAC, use the following command:

```
qacli admin --import-user-config
```

You can also use the above command with either the `--force` or `--path` argument: `--force` will force the import, overwriting any existing configuration; `--path` allows you to specify a path to an existing configuration.

If you wish to delete a Helix QAC project by removing Helix QAC project-related data from a directory, use the following command:

```
qacli admin -P <project-path> --remove-project-files
```

Normally, when a Helix QAC process is operating with a specific project, it locks the project temporarily, preventing other Helix QAC processes from acting on the project simultaneously. If this process ends abnormally, it may leave the lock in place, preventing all other processes from working with the project. If you find that you are locked out of a project, but are certain that there are no other Helix QAC processes running, the following command will remove the locks that are on the project:

```
qacli admin -P <project-path> --remove-locks
```

## CMA Project Management

A CMA (Cross-Module Analysis) project is identified by a name. When a CMA project is first created, it is empty, with no Helix QAC projects (modules) associated with it. The following command creates a new CMA project, which can subsequently be used to associate modules with it and conduct analyses:

```
qacli admin --create-cma-project <NAME>
```

This will create a new empty CMA project that can be referred to with the name specified.

The following command deletes an existing CMA project:

```
qacli admin --delete-cma-project <NAME>
```

**Note:** No change to any modules associated with the specified CMA project name will occur as a result of this call. It simply removes a name that is used to refer to a set of modules for the purpose of CMA.

The following command can be used to associate a module (Helix QAC project) with a CMA project:

```
qacli admin -P <project-path> -M <cma-project-name> -i [-K <config-name>]
```

In order to disassociate a module from a CMA project, the following command can be used:

```
qacli admin -P <project-path> -M <cma-project-name> -x
```

**Note:** `-M` is the command for **Modify**, `-i` is the shortcut for `--module-insert`, and `-x` is the shortcut for `--module-remove`.



## Extraction of C/C++ Analysis Data

After creating a project, the next thing to do is to populate it with the source files and analysis data from your build (that is to say, the Include paths and macro Definitions used to compile each source file). This data needs to be extracted from your build system in some way.

The easiest, and recommended, way is to use Process Monitoring (refer to [Process Monitoring via QA·CLI](#)).

An alternative way to extract data from your build and add it to a Helix QAC project is to use *Compiler Wrapping* (refer to [Compiler Wrapping](#)). This uses `--compile-command <command>`, for which the shortcut is `-z <command>`.

**Note:** Special attention should be paid to `--ignore-rest` (shortcut `--`), following which the build and other arguments can be added for direct parsing without them having to obey the QA·CLI syntax.

Another option is to use a [BUILD LOG](#) to add the data.

Finally, if all else fails, files can be added explicitly, as explained in the following subsection.

## Adding Files Explicitly

Files, and their corresponding **Analysis Priority**, **Include Paths** and **Definitions**, can be added to a project explicitly using the following command:

```
qacli admin -P <project-path> --add-file [--analysis-priority <integer>] --ignore-rest <arguments>
```

The shortcut for the *add-file* switch is `-a`, and the arguments represent the data to be added. For example, for a C project:

```
qacli admin -P <project-path> -a --analysis-priority 100 - -Isrc/
-Ithird/party/lib/include -DMACRO=1 src/file.c
```

**Note:** `-P` is a shortcut for `--qaf-project` and `--` is a shortcut for `--ignore-rest`.

For the case when a file already exists in the project, the file will be updated with the new argument data. An additional interface is provided to add files specified in a filelist. The `--add-files` option can be used to action this:

```
qacli admin -P <project-path> --add-files <path-to-filelist> [--analysis-priority <integer>]
```

The shortcut for this switch is `-F`.

## Updating Folders

Following a synchronization or project optimization, dependency data (Include Paths and Definitions) may become associated with folders in the project. Files downstream from folders with dependency data will inherit these dependencies, and so it's useful to have a mechanism for modifying the folders directly. It is possible to modify folders known the project, for example:

```
qacli admin -P <project-path> -a --folder -- -Ithird/party/lib -DMACRO=1 src/
```

The shortcut for the *folder* switch is `-f` and it cannot be used in combination with the *compile-command* switch, see [Compiler Wrapping](#). Similarly to [Adding Files Explicitly](#), multiple folders, alongside files, can be specified in the argument list.

## Optimization

When files are added to the project in bulk, it may be useful to optimize the project. For more information about project optimization, refer to [Optimizing a Project](#)

```
qacli admin -P <project-path> --optimize --add-files <path-to-filelist>
```

The shortcut for the optimize switch is "-z".

## Removing Files Explicitly

Generally, the simplest method of ensuring that files are consistent in a project is to use the "sync" commands described in [Extraction of C/C++ Analysis Data](#) - these will add and remove files as needed. However, as with [Adding Files Explicitly](#), a similar means is provided of removing files.

Files can be removed from a project explicitly using the following command:

```
qacli admin -P <project-path> --remove-file --ignore-rest <arguments>
```

The shortcut for the *remove-file* switch is "-d", and the arguments represent the files to be removed. For example, for a C project:

```
qacli admin -P <project-path> -d -- src/file.c
```

**Note:** "-P" is a shortcut for *--qaf-project* and -- is a shortcut for *--ignore-rest*.

An additional interface is provided to remove files specified in a filelist. The "--remove-files" option can be used to action this:

```
qacli admin -P <project-path> --remove-files <path-to-filelist>
```

The shortcut for this switch is "-D".

## Removing Folders

Individual folders can also be removed, similarly to files:

```
qacli admin -P <project-path> -d --folder -- src/
```

The shortcut for the folder switch is "-f".

Removing a folder using this method will also remove any immediate files that directly rely upon it, but any child/sub folders will not be affected.

## Optimization

When files are removed in bulk, it may be useful to optimize the project. For more information about project optimization, refer to [Optimizing a Project](#)

```
qacli admin -P <project-path> --optimize --remove-files <path-to-filelist>
```

The shortcut for the optimize switch is "-o".

## Specifying Helix QAC Dashboard Credentials

Helix QAC is able to upload and download data to and from Helix QAC Dashboard using qacli commands. All these commands require authentication with Helix QAC Dashboard. There are several ways of performing the authentication:

- Add the credentials into each command, as when issuing the command to list the Unified projects:

```
qaccli admin --list-unify-project --url <[protocol://]host[:port]> --username  
<uname> --password <pwd>
```

- Add the credentials into each command, but specify a file that holds the password:

```
qaccli admin --list-unify-project --url <[protocol://]host[:port]> --username  
<uname> --password-file <file>
```

- Use a token, as in:

```
qaccli admin --list-unify-project --url <[protocol://]host[:port]> --username  
<uname> --password-file <file>
```

For an automated set of scripts, the advantage of using a token is that it avoids having a password listed in each command, possibly scattered across many scripts. A token is created in a single place, using the following command, where the `--url` parameter is the address of the Helix QAC Dashboard server:

```
qaccli admin --get-token --expire-token <days> --url <[protocol://]host[:port]> --  
username <uname> --password-file <path>
```

This returns a string, which can be used in subsequent commands, as shown in the `--list-unify-project` examples just above. Tokens are time-limited, with a default duration of one day. Use the `--expire-token` parameter to specify the number of days prior to expiry.

For reasons of brevity, all of the examples in the following sections use the token mechanism, but in each case it is also possible to use the full set of `url/username/password` (or `password-file`) options.

## Working with Unified Projects

The section [Working with Helix QAC Dashboard](#) describes how project definitions can be shared amongst other users by storing them on Helix QAC Dashboard as Unified projects. This can all be done through command line options.

To convert your local project into a Unified project, use the `--convert-to-unify` command, which uploads the project definition to Helix QAC Dashboard (this is equivalent to using the **Create Unified Project** dialog).

```
qaccli admin -P <project_path> --convert-to-unify --project-name <unifiname>  
--server-vcs <file> --use-local-vcf --repository-path <path> --url  
<[protocol://]host[:port]> --token <tkn>
```

**Note:** Only projects with a single configuration may be unified.

The arguments to the above command are as follows:

- *P* (or *--qaf-project*) gives the location of the Helix QAC project.
- *project-name* is the name given to the project in Helix QAC Dashboard. Once the project has been uploaded, the local Helix QAC project will be adjusted to use this name.
- *server-vcs* gives the name of the version control system file to be used. The list of files can be found by running the command:

```
qacli admin --list-server-vcs --url <[protocol://]host[:port]> --token <tkn
```

- *use-local-vcf* specifies that a local copy of the Version Control Compatibility File should be used where available. If *use-local-vcf* is not specified, then the file associated with the *server-vcs* option will be downloaded and used as the VCF.
- *repository-path* contains any additional information needed by the version control system holding the source code. This is an optional parameter.
- *url* is the address of the Helix QAC Dashboard server.
- *token* is the string value returned from the *get-token* function.

Alternatively, the *--username* and *--password* (or *--password-file*) parameters can be used.

Once the project has been uploaded to Helix QAC Dashboard, it is available for other users to download. Users can see which Unified projects are available for download by using the following command:

```
qacli admin --list-unify-project --url <[protocol://]host[:port]> --token <tkn>
```

The actual download is performed by using the following command:

```
qacli admin -P <project_path> --pull-unify-project --project-name<unifyname> --url  
<[protocol://]host[:port]> --token <tkn>
```

If the definition of the Unified project needs to be updated, the new definition can be uploaded to Helix QAC Dashboard using the following command:

```
qacli admin -P <project_path> --push-unify-project --url <[protocol://]host[:port]>  
--token <tkn>
```

## Upgrade

Each version of Helix QAC that is released is, more often than not, released with new and/or updated analysis components. Projects that have been created by older versions of Helix QAC will have Rule Configuration Files and Analysis Configuration Files that refer to the versions of analysis components that are not shipped with the new Helix QAC version. If you wish to convert your existing projects to refer to the new versions of components instead of the old versions, the following command can be issued:

```
qacli admin --qaf-project <project-path> --upgrade [-K <config-name>]
```

This will upgrade the default configuration of a project. For projects with multiple configurations you can use the `-K` option to upgrade a specific config, alternatively you can specify `-K "*"` to upgrade all configurations in the project.

This will only work if Helix QAC possesses an upgrade map for the specified project version. This command will not add any new features or components (for example, new messages or analysis components) to your configuration files. It will replace corresponding features from the older versions with the newer versions of those features. It will also remove any features or components that no longer exist in the newer versions.

Should you prefer to continue using older versions of components, do not use the `--upgrade` command. You will instead need to tell Helix QAC where to find the old components using the following command:

```
qaccli admin --add-component-search-path <path>
```

This will cause the older component to appear in the toolchain editing screen (the Analysis tab of the Project Properties dialog). Use the arrow button to add the older component into the active toolchain.

**Note:**

You should not mix old and new versions of components. If you have used the `--add-component-search-path` option, set the older component as active for all the projects run on that machine.

Using older components in a newer version of Helix QAC is not a supported configuration - there is no guarantee of compatibility.

To list the component search paths, you can use the following command:

```
qaccli admin --list-component-search-paths
```

This command lists the user-specified component search paths that are set for the current installation of Helix QAC. To remove a user-specified component search path, you can use the following command:

```
qaccli admin --remove-component-search-path <path>
```

## License Server Management

To add a license server for Helix QAC:

```
qaccli admin --set-license-server <port@host|host[:port]>
```

By default, the license server is added at the top of the list. To add it at the bottom, append the `--low` switch, for example:

```
qaccli admin --set-license-server <port@host|host[:port]> --low
```

To remove a license server from Helix QAC:

```
qaccli admin --remove-license-server <port@host|host[:port]>
```

To list the license servers of which Helix QAC is aware:

```
qaccli admin --list-license-servers
```

To check the licenses assigned to Helix QAC under the current license server setup:

```
qaccli admin --check-license
```

## User Data Location Management

As previously described (refer to [Installation Management](#)), the user data location is by default dependent on your desktop environment. It is possible to override this behavior and specify a location where this data should be held, allowing the location to be shared among users.

To set the location:

```
qacli admin --set-user-data-location <directory>
```

### Note:

- The directory must exist and you must have Write permissions to it.
- The target directory must exist and you must have Write permissions to it.
- Initially, no files are copied from the previous user data location. When the next qacli command is issued, the default files from the installation directory will be copied. No user-created files from the previous location are copied - they must be copied manually if needed.
- You should immediately restart the application (using QA·GUI, Helix QAC Visual Studio or Helix QAC Eclipse as appropriate) whenever the set-user-data-location command is issued.

To obtain the current location (or the default if no location has previously been set):

```
qacli admin --get-user-data-location
```

To restore the location back to the default:

```
qacli admin --reset-default-user-data-location
```

This will reset the user data location so that it is once more tied to the user environment. The restrictions in the *Note* section above for the `--set-user-data-location` command also apply to this command.

## Administration of User Messages File

A User Messages File allows you to assign your own messages for each component to existing or newly created diagnostic numbers. Optionally, you can set an associated HTML help file and references for a message.

The User Messages File is a simple XML file that can be created manually or via the GUI (refer to [Creating User Messages](#)).

The command `--user-messages`, when used with `--qaf-project-config`, will add the User Message XML file to your Helix QAC system:

```
qacli admin --qaf-project-config --qaf-project <directory> --cct <path> --rcf  
<path> --acf <path> --user-messages <user-messages-file>[--config <config-name>]
```

**Note:**

- The User Messages File is applied to Helix QAC overall and not to individual projects.
- It is recommended that you create a simple User Messages File via the GUI to observe the XML structure and then replicate the structure for your needs.

## Source Extensions

Helix QAC determines how to process a source file based on the language of the source file, for example, C or C++. It does this using the file extension of the source file and the operating system. For each project, a set of file extensions is mapped to a source language. See Setting the [Setting the File Extensions for Your Sources](#) for the default values.

The following commands allow you to modify and view these mappings.

To add a new extension, use *--add-source-extension*, for example:

```
qaccli admin --add-source-extension CCC -P . --target-language C
```

This will add the extension "CCC" to the project in the current directory and associate it with the C language.

To remove an extension, use *--remove-source-extension*, for example:

```
qaccli admin --remove-source-extension CCC -P . -T C
```

This will remove the extension "CCC" from the project in the current directory.

To view the list of extensions associated with a language, use the *--list-source-extensions* option, for example:

```
qaccli admin --list-source-extensions -P . -T C++
```

This will list all extensions associated with the current project for the C++ language.

**Note:**

- The extensions are case sensitive.
- If no leading "." is present, then it is assumed. Therefore, ".CCC" and "CCC" are identical.
- The following language types are available: C, C++, C/C++ and WEB\_SECURITY.

## Analyze

The QA-CLI 'analyze' subcommand allows you to perform analysis on existing Helix QAC and CMA projects.

Verbose help on the command can be accessed as follows:

```
qaccli analyze -h
```

## Analysis of Helix QAC Projects

The following command can be issued to perform analysis on a Helix QAC project:

```
qaccli analyze -P <project-path> --file-based-analysis [-K <config-name>]
```

The '-P' option can be omitted if your current working directory is the same directory as that of the project. This applies to all subcommands except the 'admin' subcommand. The shortcut for the *--file-based-analysis* switch is '*-f*'.

The following command can be issued to clean a Helix QAC project:

```
qaccli analyze -P <project-path> --clean [-K <config-name>]
```

The shortcut for this switch is '-c'.

The following switch can be used to stop analysis at the moment of the first analysis failure:

```
qaccli analyze -P <project-path> -f --stop-on-fail [-K <config-name>]
```

The shortcut for this switch is '-s'.

**Note:** If you wish to kill the commands currently running in the command window, use *Ctrl+C*.

When running Inter-TU Dataflow Analysis (refer to the Helix QAC for C or Helix QAC for C++ manual), it can be beneficial to run the analysis more than once. Running it twice should result in a reduction of the number of possible issues and a potential increase in the number of Definite, Apparent and Suspicious issues found.

To enable Inter-TU Dataflow Analysis, use the *--inter-tu-dataflow* command:

```
qaccli analyze -P <project-path> -f --inter-tu-dataflow [-K <config-name>]
```

The shortcut for this option is *-I*.

**Note:** The above option cannot be used with CMA and is only available for Helix QAC for C and Helix QAC for C++. You should ensure that your configured value for *df::inter* is non-zero. This option replaces the deprecated *--repeat* option.

To specify a particular file to be analyzed, the path to the source file can be provided, as follows:

```
qaccli analyze -P <project-path> -f <source-file-path> [-K <config-name>]
```

**Note:** The above option cannot be used with CMA.

To specify a set of files to be analyzed, the path to each source file must be placed in a *filelist*, which is a text file made up of one path per line. To analyze the files in the list:

```
qaccli analyze -P <project-path> -f --files <path-to-filelist> [-K <config-name>]
```

The shortcut for this option is '-F'.

**Note:** The above option cannot be used with CMA.

The following command will keep processing files and reporting progress per file even if a license check failure has occurred:

```
qaccli analyze -P <project-path> -f --force-complete [-K <config-name>]
```

**Note:** The above command and the earlier *--stop-on-fail* command are mutually exclusive.



## Retry after analysis failure

Depending upon network stability and/or limits on your license it's possible for the analysis to fail with a licence failure or perhaps due to available machine resources a parser could fail. Now, by default analysis will be attempted upto three additional times. You can modify the number of times it will retry using the `--retry` option, for example:

```
qacli analyze -P <project-path> -f --retry 5
```

This will try upto five additional times to analyze a file if a license or parser failure occurs, with a small wait before each retry. A value of zero will disable retries.

## Monitoring Progress

As files are analyzed progress is output to the console (refer to [Progress Output](#)). Progress from the analysis of source files can additionally be written to a file which can then be monitored to gauge the progress of the analysis, with one line output for each file processed. The last line will begin with "Progress(<label>)", followed by one dot (".") for every 3% of progress that the final component has taken. When complete, it will end the line with "done". For example:

```
qacli analyze -P <project-path> -f --output-progress <output-path> [-K <config-name>]
```

The shortcut for this option is `'-o'`.

By default the delta time for analysis is not displayed. If you wish to add this to the output (both console and file) then specify `--show-timings`, shortcut `-T`. The output will then be appended with the time taken to analyze each file. It will be in the format: ([nd] HH:MM:SS)

Where 'nd' is the number of days (only shown if greater than zero) and HH:MM:SS are the number of hours, minutes and seconds that the analysis took to complete. For example:

```
qacli analyze -P <project-path> -f --show-timings [-K <config-name>]

Progress(Cleaning): ..... done
/home/cartman/sample_inspect_c/src/inspect.c:qac:0:1:0 (00:00:01)
Progress(File-based Analysis): ..... done (00:00:02)
```

## Analysis Timings

By default the time taken for analysis is not displayed. If you wish to add this to the output (both console and optionally the progress file) then specify `--show-timings`, shortcut `-T`. The output will then be appended with the time taken to analyze each file. It will be in the format:

([nd] HH:MM:SS)

Where 'nd' is the number of days (only shown if greater than zero) and HH:MM:SS are the number of hours, minutes and seconds that the analysis took to complete. For example:

```
qacli analyze -P <project-path> -f --show-timings [-K <config-name>]

Progress(Cleaning): ..... done
/home/cartman/sample_inspect_c/src/inspect.c:qac:0:1:0 (00:00:01)
Progress(File-based Analysis): ..... done (00:00:02)
```

The time taken to analyze each file will be shown and additionally the total analysis time will also be shown at the end of the command.

**Note:**

- All times are rounded to the nearest second.
- The individual times are for analysis only and will not include any time taken by Helix QAC to process each file.
- Given the two points above it's possible that summing all the individual times will not match the displayed total analysis time.

## Simultaneous Synchronization and Analysis

It is possible to simultaneously synchronize (add files to a project) and analyze them:

```
qaccli analyze -P <project-path> --build-command <script>
```

The shortcut for this switch is '-b'. The script, which is generally a build command will be executed and the build monitored. As files are detected they will be added to the project. At the end of the build any added files will then be analyzed, for example:

```
qaccli analyze -b make

cc -c -ohello\ world.o hello\ world.c
cc -ohello\ world hello\ world.o
/home/unicorn/sample_projects/Hello World-C/hello world.c - found
The command 'make' has completed with an exit code of 0
/home/unicorn/sample_projects/Hello World-C/hello world.c:qac:0:1:0
Progress(Sync Analysis): ..... done
```

## Analysis of CMA Projects

As outlined in [CMA Project Management](#), Helix QAC projects can be associated with CMA projects as "modules".

As a useful side effect, this feature can be used to conduct all analyses on a number of Helix QAC projects. To conduct CMA, specify the name of the CMA project and the name of the Helix QAC project against which the results are to be stored:

```
qaccli analyze -P <project-path> -C <cma-project-name> [-K <config-name>]
```

If no Helix QAC project is specified, the default is to use the first in the list of Helix QAC projects in the CMA project.

All of the normal switches apply for CMA (except for *--file-based-analysis*, which is implicit). For example, the following command:

```
qaccli analyze -P <project-path> -C <cma-project-name> -csga [-K <config-name>]
```

will clean the associated modules first before performing file-based analysis on each, halting if an analysis failure is encountered, generating preprocessed source for each source file that is analyzed, and creating an archive containing support data for any file that fails analysis.

## Raw Source Analysis

Sometimes it is beneficial to analyze a file that is not part of a Helix QAC project, for example, to analyze a preprocessed file in order to debug configuration issues. The following command can be used to do this:

```
qaccli analyze -P <project-path> --raw-source <file-path>
--language-cct <cct-path> [-K <config-name>]
```

### Note:

- If no CCT is supplied then 'Helix\_Generic\_C++.cct' is used by default.
- If the file is omitted, then you may also use '-F' to specify a filelist.
- Only the Helix QAC for C and Helix QAC for C++ components support Raw Source Analysis.
- The files must be '.i' files, that is to say, files previously generated by analyzing with the 'Generate Preprocessed Source' option enabled.

## Debugging Analysis

For C and C++, preprocessed source is useful for debugging configurations and code in general. Helix QAC will generate preprocessed source files in the `prqa/configs/<config_name>/output/SOURCE_ROOT` directory structure of a project. The following switch can be used to generate a preprocessed source file for each file that is analyzed:

```
qaccli analyze -P <project-path> -f --generate-preprocessed-source
[-K <config-name>]
```

The shortcut for this switch is '-g'.

When analysis configuration problems result in source files failing to analyze successfully, it is helpful to gather information and relay it to a Programming Research Support Engineer. The following switch specifies that Helix QAC is to create an archive for each file that fails analysis, with each archive containing the `.met` (or `.arc`), `.i` and `.via` files (this is equivalent to the QA GUI option **Assemble Support Analytics for Failed Files** - refer to Section [The Analyze Menu](#)):

```
qaccli analyze -P <project-path> -f --assemble-support-analytics
[-K <config-name>]
```

The shortcut for this switch is '-a'. The archive will be created in the `prqa/configs/<config_name>/output/SOURCE_ROOT` directory structure of the Helix QAC project.

## Baseline

The general functionality of baselines is described in [Baseline Diagnostics Suppression](#).

Verbose help on the command can be accessed as follows:

```
qaccli baseline -h
```

## Setting a Dashboard Baseline

This involves downloading code and diagnostics from a Dashboard snapshot to form the actual baseline. Once this has been done, you need to tell Helix QAC to apply the baseline.

**Note:**

These baselines are only available for Unified projects, which have links to projects within Dashboard.

The commands that obtain data from Dashboard need to connect to Dashboard, using the URL for the Dashboard server, as well as logon credentials. As stated in [Specifying Helix QAC Dashboard Credentials](#), the logon credentials can involve either specifying a username and password, or the use of a token. The example calls listed here use the token method.

The available snapshots for the Unified project can be listed using the command:

```
qacli baseline -P <project_path> --list-snapshots --url <[protocol://]host[:port]> --token <tkn>
```

To download the baseline, use:

```
qacli baseline -P <project_path> --pull-baseline <snapshot> --url <[protocol://]host[:port]> --token <tkn>
```

Once the baseline has been downloaded, it can be applied using the command:

```
qacli baseline -P <project_path> --set-baseline --baseline-type UNIFIED
```

To turn the baseline off, use:

```
qacli baseline -P <project_path> --set-baseline --baseline-type OFF
```

## Setting a Version Control Baseline

This involves generating the baseline from the current analysis results, and then applying it. To generate the baseline:

```
qacli baseline -P <project_path> --generate-baseline --baseline-type VCS [-K <config-name>]
```

To apply the baseline:

```
qacli baseline -P <project_path> --set-baseline --baseline-type VCS [-K <config-name>]
```

To turn the baseline off:

```
qacli baseline -P <project_path> --set-baseline --baseline-type OFF [-K <config-name>]
```

## Setting a Local Baseline

Setting a local baseline requires that an analysis has previously been generated. After performing an analysis, a local baseline can be generated using the following command:

```
qacli baseline -P <project_path> --baseline-type LOCAL --generate-baseline [-K <config-name>]
```

The source code associated with the baseline must be stored, so that when the baseline is actually applied, diff-style techniques can be used to match baselined diagnostics with the diagnostics in the latest code. To store:

```
qacli baseline -P <project_path> --baseline-type LOCAL --set-baseline [--local-source <output-path>] [-K <config-name>]
```

**Note:**

The optional output-path is the directory containing details of the differences in the source code. The default

(prqa\configs\<config-name>\output) will suffice for most users, but maybe overridden if needed

To turn the baseline off, use:

```
qacli baseline -P <project_path> --baseline-type OFF --set-baseline [-K <config-name>]
```

## Downloading Dashboard Suppressions

The general functionality of Dashboard suppressions is described in [Downloading Suppressions](#).

**Note:** Dashboard suppressions are only available for Unified projects, which have links to projects within Dashboard.

The suppressions are downloaded from a Dashboard snapshot. The available snapshots for the Unified project can be listed using the command:

```
qacli baseline -P <project_path> --list-snapshots --url <[protocol://]host[:port]> --token <tkn>
```

To download the suppressions, use:

```
qacli baseline -P <project_path> --pull-suppressions <snapshot-name> --url <[protocol://]host[:port]> --token <tkn>
```

Once the suppressions have been downloaded, they will automatically be applied when the analysis results are viewed.

## Export

The QA-CLI 'export' subcommand allows you to export information from the Helix QAC Installation, Configuration or a Helix QAC project. At present you are only able to export CCTs. There is also a partner command import which allows the importing of data, this can be found in section [Import](#).

Verbose help on the command can be accessed as follows:

```
qacli export -h
```

The format for the 'export' command is as follows:

```
export [-P <directory>] [-K <config name>] [-u <file or directory>] -t <CCT> -C  
<PROJECT|LOCAL|INSTALL|USER> [--] [-h] <export-file>
```

- '-t' (or --type) specifies the export type, at present only CCT is supported.
- '-C' (or --cct-location) specifies the location of the CCT to export. Valid values are; PROJECT, LOCAL, INSTALL, USER.
- '-P' (or --qaf-project) specifies the Helix QAC project directory. This should be used when '-C' PROJECT is specified.
- '-K' (or --config) specifies the config in Helix QAC project directory to use.
- '-u' (or --user-defined) specifies the location of a file/directory to extract CCTs from. This should be used when '-C' USER is specified.
- export-file specifies the optional name of the output file.

## Exporting CCTs

It is now possible to export a CCT and all its associated files such as Script and Stub files and their directory structures, and to package them up into a single ZIP'd archive. This can then be moved between projects or used to aid support when CCT updates are needed.

CCTs are stored in several places and using the *export* (and *import*) commands will help to ensure these areas remain consistent. Using the '-C' (or --cct-location) specifies where the CCT will be extracted from. Values are:

- **INSTALL** : The CCT installation directory, eg

```
/home/bill/Helix-QAC-2019.1/config/cct
```

or

```
C:\Perforce\Helix-QAC-2019.1\config\cct
```

- **LOCAL** : The CCT directory in the local user data location, eg

```
/home/bill/.config/Perforce/Helix-QAC-2019.1/config/cct
```

or

```
C:\%LOCALAPPDATA%\Perforce\Helix-QAC-2019.1\config\cct
```

- **PROJECT** : The CCT directory of a project, eg

```
/home/bill/sample/configs/Initial/config/cct
```

or

```
C:\sample\configs\Initial\config\cct
```

- **USER** : Any file or directory specified by the user

The *export-file* specifies the name of the output file. This is optional and if not specified it is based upon the *cct-location* parameter:

- **INSTALL** : `INSTALL.zip_cct`
- **LOCAL** : `LOCAL.zip_cct`
- **USER** : `USER.zip_cct`
- **PROJECT** : `<parent-directory>.zip_cct` Notes:
- Only valid CCTs can be exported. If the CCTs to be exported are invalid (such as missing Stub/Script directories/files etc) then the resultant archive may be incomplete. A non zero return code will indicate this.

## Example Usage

To export the CCTs for a particular project then issue a command such as:

```
qacli export -P . -t CCT -C PROJECT
```

This assumes you are located in the same directory as the project and the resulting output file will be given a default name - based upon the name of the parent folder and ending in '.zip\_cct'.

For a project with multiple configurations:

```
qacli export -P /home/bill/sample -t CCT -C PROJECT -K other_config my_ccts.zip_cct
```

This version gives the absolute path to the project and exports the CCTs (from the 'other\_config') and outputs it all into the file 'my\_ccts.zip\_cct'.

To export alls the CCTs for your user data location then issue a command such as:

```
qacli export -t CCT -C LOCAL
```

Again, if not specified the output will be to a default name (LOCAL.zip\_cct)

You can also specify a location to extract the CCTs from (perhaps you have a directory that stores a family of CCTs specific to your setup):

```
qacli export -t CCT -C USER -u /home/bill/my_ccts
```

So all CCTs in that directory will be exported to a default name (USER.zip\_cct)

## Help

The QA-CLI 'help' subcommand provides information on *qacli* return codes and Helix QAC compiler support.

Verbose help on the command can be accessed as follows:

```
qacli help -h
```

To access a listing of supported compilers:

```
qacli help --compatibility
```

The shortcut for this switch is '-c'.

To access descriptions of qacli return codes:

```
qacli help --return-codes
```

The shortcut for this switch is '-r'

## Import

The QA-CLI 'import' subcommand allows you to import information to the Helix QAC Installation, Configuration or a Helix QAC project. At present you are only able to import CCTs. There is also a partner command export which allows the exporting of data, this can be found in section [Export](#).

Verbose help on the command can be accessed as follows:

```
qacli import -h
```

The format for the 'import' command is as follows:

```
import [-P <directory>] [-K <config name>] [-u <file or directory>] -t <CCT> -C  
<PROJECT|LOCAL|INSTALL|USER|ALL> [--] [-h] <import-file>
```

- '-t' (or --type) specifies the import type, at present only CCT is supported.
- '-C' (or --cct-location) specifies where to import the CCTs to. Valid values are; PROJECT, LOCAL, INSTALL, USER and ALL.
- '-P' (or --qaf-project) specifies the Helix QAC project directory. This should be used when '-C' PROJECT is specified.

- `'-K'` (or `--config`) specifies the config in Helix QAC project directory to use.
- `'-u'` (or `--user-defined`) specifies the location of a file/directory to import CCTs to. This should be used when `'-C'` USER is specified.
- *import-file* specifies the name of the input file.

## Importing CCTs

It is now possible to import an archive containing CCT(s) and their associated files such as Script and Stub directories and files. This can then be used to aid support when CCT updates are needed.

CCTs are stored in several places and using the *import* (and *export*) commands will help to ensure these areas remain consistent. Using the `'-C'` (or `--cct-location`) specifies where the CCT will be imported to. Values are:

- **INSTALL** : The CCT installation directory, eg  
`/home/bill/Helix-QAC-2019.1/config/cct`  
or  
`C:\Perforce\Helix-QAC-2019.1\config\cct`
- **LOCAL** : The CCT directory in the local user data location, eg  
`/home/bill/.config/Perforce/Helix-QAC-2019.1/config/cct`  
or  
`C:\%LOCALAPPDATA%\Perforce\Helix-QAC-2019.1\config\cct`
- **PROJECT** : The CCT directory of a project, eg  
`/home/bill/sample/configs/Initial/config/cct`  
or  
`C:\sample\configs\Initial\config\cct`
- **ALL** : Effectively an alias for INSTALL, LOCAL and PROJECT. It allows a CCT to fully replace an existing CCT.
- **USER** : Any file or directory specified by the user

The *import-file* specifies the name of the input file.

### Note:

- If you try to import to INSTALL either directly or indirectly (by using ALL) you must have the appropriate permissions to do so.
- When importing to a project you can only import the same number and names of CCTs i.e. you cannot try to change to a different CCT by using import.

## Example Usage

To import the CCTs for a particular project then issue a command such as:

```
qacli import -P . -t CCT -C PROJECT my_ccts.zip_cct
```

This assumes you are located in the same directory as the project.

For a project with multiple configurations:

```
qacli import -P /home/bill/sample -t CCT -C PROJECT -K other_config my_ccts.zip_cct
```

This version gives the absolute path to the project and imports the CCTs (to the 'other\_config').



To import all the CCTs for your user data location then issue a command such as:

```
qacli import -t CCT -C LOCAL my_ccts.zip_cct
```

You can also specify a location to import the CCTs to (perhaps you have a directory that stores a family of CCTs specific to your setup):

```
qacli import -t CCT -C USER -u /home/bill/my_ccts my_ccts.zip_cct
```

So all CCTs will be imported to that directory.

## Log

The QA-CLI 'log' subcommand allows you to configure where and how much content log files have. Verbose help on the command can be accessed as follows:

```
qacli log -h
```

The format for the 'log' command is as follows:

```
log {-p <enable|disable>|-l <NONE|ERROR|INFO|DEBUG|TRACE>} [-h]
```

By default logfiles are stored under the User Data Store (see [Setting the User Data Store](#)) in the "app/logs" sub-directory - all logging will go to that location. It is possible to direct logging for a project to be local to that project by setting the option '--project-logging' (-p for short) to 'enable', for example:

```
qacli log --project-logging enable
```

Now all project specific logging (i.e. commands with a -P or --qaf-directory) will be placed in the 'prqa/configs/<config-name>/logs' sub-directory. Commands that are not project specific will continue to be logged under the User Data Store in the "app/logs" sub-directory. To disable project logging, simply issue:

```
qacli log --project-logging disable
```

The size of the log files, and the information that they contain, are dependent on the log level that is set for the current user. By default, the debug level is set to ERROR. The following command can be used to change this:

```
qacli log --set-level <NONE|ERROR|INFO|DEBUG|TRACE>
```

The arguments are displayed in the order of the amount of data that is written to log files, with NONE specifying that no logs are to be written, and TRACE specifying that everything possible is to be logged.

### Note:

The above command needs to be set to DEBUG or TRACE in order for any console output from the Helix QAC for C and Helix QAC for C++ parser options (for example -settings+) to be redirected to the Helix QAC log files. These log files are sorted by time, and so console output from different processes running concurrently will become mixed. In this event, you can use the process ID following the time stamp on the log line to filter output for a specific component process. However, you should be aware that leaving the command set to DEBUG or TRACE for long periods will result in frequent large logs that are liable to take up a large amount of disk space very quickly and slow down all Helix QAC operations. Therefore, only use such logging if strictly necessary, or if requested by support engineer.

## Pprops (Project Properties)

The *pprops* command allows the configuration of Helix QAC Project Properties via QA-CLI.

Verbose help on the command can be accessed as follows:

```
qacli pprops -h
```

In the sections below, the common parameters *component\_key* and *toolchain* are shown when using the `qacli pprops --list-components` command. For example:

```
qacli pprops --list-components

rcma-1.5.0, target language - C_CPP
namecheck-1.0.2, target language - C
namecheck-1.0.2, target language - C++
qacpp-4.1.0, target language - C++
qac-9.3.0, target language - C
```

The *component\_key* is the value in the first column (for example, `qac-9.3.0`) and *toolchain* is the value in the final column (for example, `C`).

## Listing Components

1. To list all components available in the system:

```
qacli pprops --list-components
```

2. To list all components available in the system that support a given toolchain:

```
qacli pprops --list-components -T <toolchain>
```

- An invalid toolchain results in an error.

3. To list the components used in a particular project:

```
qacli pprops --list-components -P <path>
```

- An invalid project or path results in an error.

4. To list components used in a particular project that support a given toolchain:

```
qacli pprops --list-components -T <toolchain> -P <path>
```

- An invalid toolchain results in an error.
- An invalid project or path results in an error.

## Adding a component to a toolchain in a project

```
qacli pprops -c <component_key> --add -T <toolchain> -P <path>
```

- An invalid component results in an error.
- An invalid toolchain results in an error.

- Trying to assign a component to an incompatible toolchain (language) results in an error.
- Trying to assign a component to a toolchain that does not exist in the project results in an error.

## Upgrading a toolchain component in a project

To upgrade a component, simply follow the command above to add a component and the existing component will be upgraded.

This will **not**, however, upgrade your RCF. If the new component has a more recent RCF, you may wish to upgrade your RCF as well, using a command such as:

```
qacli admin --qaf-project-config --qaf-project <qaf-project> --rcf <rcf-path>
```

The additional `--update-method` option allows you to replace, update or merge the RCF. It takes one of three possible values:

- **Replace:** The file is replaced by the one specified. This is the default for RCFs, ACFs, VCFs and CCTs.
- **Update:** The existing file is updated, replacing the mappings for the existing components. RCFs only.
- **Merge:** The existing file is updated, merging rulesets from both files with the same component (in the case of a merge conflict where the new and old values differ, then the new values are used). RCFs only.

**Update** should be specified if you are already using a default RCF and you still require default mappings. **Merge** should be specified where you have customized the RCF and wish to retain the customizations (where possible).

## Removing a toolchain component from a project

```
qacli pprops -c <component_key> --remove -T <toolchain> -P <path>
```

- An invalid component results in an error.
- Trying to remove a primary analyzer is an error.
- An invalid toolchain results in an error.
- Trying to remove a component from a toolchain that does not exist in the project results in an error.

## Component Options

Important:

By default only "basic" component options are shown. If you need to view "advanced" component options then ensure the optional `--advanced` option is added to the command.

1. To list the options available for a given component:

```
qacli pprops -c <component_key> --list-options
```

- An invalid component results in an error.
- Ignores `-P` or `--qaf-project` switch with a warning.

2. To print component option information (name, shortcut, argument, argument syntax, cumulativity, default value, description):

```
qacli pprops -c <component_key> -o <option> -i
```

- An invalid component results in an error.
- An invalid option results in an error; prints available options.
- Ignores `-P` or `--qaf-project` switch with a warning.
- The name of the option does not have to be prefixed with a dash. For example, `-o enabledataflow` and `-o -enabledataflow` should both work.

3. To set a component option for a project (or to add to the options if cumulative):

```
qacli pprops -c <component_key> -o <option> --set <value> -P <path> [-K <config-name>]
```

- An invalid component results in an error.
- An invalid project or path results in an error.
- An invalid option results in an error; prints available options.
- An invalid value results in an error; prints option information.
- If the component is present in more than one toolchain in the particular project, disambiguation is required using the `-T` or `--toolchain` switch. Otherwise, an error will be generated.
- The name of the option does not have to be prefixed with a dash - that is to say, `-o enabledataflow` and `-o -enabledataflow` will work equally.

4. To set a component option only once for a project:

```
qacli pprops -c <component_key> -o <option> --set-once <value> -P <path> [-K <config-name>]
```

- If the component option is already set multiple times, then this option will not add it again - nor will it remove any existing instances (use `--unset` or `--reset` for this).

5. To reset a component option to its default value for a project:

```
qacli pprops -c <component_key> -o <option> --reset -P <path> [-K <config-name>]
```

- An invalid component results in an error.
- An invalid project or path results in an error.
- An invalid option results in an error; prints available options.
- If the component is present in more than one toolchain in the particular project, disambiguation is required using the `-T` or `--toolchain` switch. Otherwise, an error will be generated.
- The name of the option does not have to be prefixed with a dash - that is to say, `-o enabledataflow` and `-o -enabledataflow` will work equally.

6. To unset a component option value:

```
qacli pprops -c <component_key> -o <option> --unset <value> -P <path> [-K <config-name>]
```

- Unsetting may be used to remove a value from a cumulative option. On non-cumulative options, it has the same effect as `--reset`.
- An invalid component results in an error.

- An invalid project or path results in an error.
- An invalid option results in an error; prints available options.
- If the component is present in more than one toolchain in the particular project, disambiguation is required using the `-T` or `--toolchain` switch. Otherwise, an error will be generated.
- The name of the option does not have to be prefixed with a dash - that is to say, `-o enabledataflow` and `-o -enabledataflow` will work equally.

7. To view the values of all the options of a component in a project:

```
qaccli pprops -c <component_key> --view-values -P <path> [-K <config-name>]
```

- An invalid component results in an error.
- An invalid project or path results in an error.
- Does not list default values. These are not stored in the project and can be viewed in the option information.

8. To reset the values of all the options of a component in a project:

```
qaccli pprops -c <component_key> --reset-component -P <path> [-K <config-name>]
```

- An invalid component results in an error.
- An invalid project or path results in an error.
- Outputs options that are reset.

9. To set several component options from a file for a project (or to add to the options if cumulative):

```
qaccli pprops -c <component_key> -P <path> --options-file <opt-file> --set-options [-K <config-name>]
```

- An invalid component results in an error.
- An invalid project or path results in an error.
- An invalid file results in an error.
- The `opt-file` lists pairs of values, in the format `-<option> value value` (that is to say, in a similar format to a `.via` file).

10. To add components with dependent components

```
qaccli pprops --add -c [component] -f (--force-dependencies)
```

11. Passing the switch `--force-dependencies` automatically adds any dependencies that the component may have. If there are multiple versions of a dependency installed, the latest version is selected. The shortcut for this option is `'-f'`.

## Component Preset Groups

1. To list the component preset groups for a given component:

```
qaccli pprops -c <component_key> --list-preset-groups
```

- An invalid component results in an error.
- Ignores `-P` or `--qaf-project` switch with a warning.

2. To list the available presets in a group, for a given component:

```
qacli pprops -c <component_key> -g <name> -i
```

- An invalid component results in an error.
- Ignores `-P` or `--qaf-project` switch with a warning.

3. To enable a preset on a component in a project:

```
qacli pprops -c <component_key> -g <name> --set <presetname> -P <path> [-K <config-name>]
```

- An invalid component results in an error.
- An invalid project or path results in an error.
- An invalid preset group results in an error; prints available preset groups.
- An invalid preset name results in an error; prints available values.
- If the component is present in more than one toolchain in the particular project, disambiguation is required using the `-T` or `--toolchain` switch. Otherwise, an error will be generated.

## Sync Settings

Synchronisation will take place with configuration taken from the CCT header. This can be overridden by the user by using the CLI PPROPS command or the GUI. There are several different sync settings that can be modified, the list below shows which settings can be modified and typical values:

- `INCLUDE_PATH` : The symbols used to extract Include Paths from the command line, eg `-I`
- `DEFINE_SYMBOL` : The symbols used to extract Definitions from the command line, eg `-D`
- `FLAG_SCRIPT` : Path to a script that can be used to extract the Include Paths and Definitions from the command line, eg `DATA/Helix_Generic_Extract_C++/Script/extract_flags.py`
- `EXCLUDE_PROCESS_PATTERN` : Specify how processes are excluded from the sync; either by a CSV string match or Regular Expression match. eg `CSV` or `REGEXP2`
- `EXCLUDE_PROCESS` : Specify which processes to exclude from the sync (see above), eg for CSV it could be `mv`, `rm` or for REGEXP2 it could be `\b(rm|mv)\b`. This will remove from the sync `rm` or `mv` commands. The CSV variant only matches on the first word in the command line, the regex version uses the entire command line for a match.
- `FILE_FILTER` : Filter out individual files or folders from the sync. Simple string matching is used, eg `/home/project/subproject` will filter the folder `/home/project/subproject` as well as the file `/home/project/subproject1.cpp`
- `SUPPRESS_INCLUDE_PATHS` : Denotes if Includes Paths are suppressed during sync. Valid values are `+` and `-`

1. To list the sync settings for a project:

```
qacli pprops --view-sync-settings -P <path>
```

- An invalid project or path results in an error.

2. To set a sync value in a project:

```
qacli pprops -P <project-path> --sync-setting <setting> --set <value>
```

- Where `<setting>` is `INCLUDE_PATH`, `DEFINE_SYMBOL`, `FLAG_SCRIPT`, `EXCLUDE_PROCESS`, `EXCLUDE_PROCESS_PATTERN`, `FILE_FILTER` or `SUPPRESS_INCLUDE_PATHS`.

- An invalid project or path results in an error.

```
qaccli pprops -P <project-path> --sync-setting EXCLUDE_PROCESS_PATTERN --set <value>
```

- Where `<value>` is CSV (by default) or REGEXP2.
- Setting a new exclusion pattern will clear the `EXCLUDE_PROCESS` strings.
- An invalid string results in an error.
- Multiple CSV separated values may be used to set values for `DEFINE_SYMBOL`, `FILE_FILTER` and `INCLUDE_PATH`:

```
qaccli pprops -P . --sync-settings FILE_FILTER --set stan.c, cartman.cpp, eric.c
```

### 3. To remove a sync value from a project:

```
qaccli pprops -P <project-path> --sync-setting <setting> --unset <value>
```

- Where `<setting>` is `INCLUDE_PATH`, `DEFINE_SYMBOL`, `FLAG_SCRIPT`, `EXCLUDE_PROCESS`, `EXCLUDE_PROCESS_PATTERN`, `FILE_FILTER` or `SUPPRESS_INCLUDE_PATHS`.
- An invalid project or path results in an error.
- Multiple CSV separated values may be used to unset values for `DEFINE_SYMBOL`, `FILE_FILTER` and `INCLUDE_PATH`:

```
qaccli pprops -P . --sync-settings FILE_FILTER --unset stan.c, eric.c
```

### 4. To reset a sync setting in a project:

```
qaccli pprops -P <project-path> --sync-setting <setting> --reset
```

- Where `<setting>` is `INCLUDE_PATH`, `DEFINE_SYMBOL`, `FLAG_SCRIPT`, `EXCLUDE_PROCESS`, `EXCLUDE_PROCESS_PATTERN`, `FILE_FILTER` or `SUPPRESS_INCLUDE_PATHS`.
- An invalid project or path results in an error.

## CIP Settings

There are several different CIP settings that can be modified: `DISABLE_CIP_GENERATION` and `IGNORE_CIP_ERROR`. They are all flag fields that can take either '-' or '+' as a value.

### 1. To list the CIP settings for a project:

```
qaccli pprops --view-cip-settings -P <path>
```

- An invalid project or path results in an error.

### 2. To set a CIP value in a project:

```
qaccli pprops -P <project-path> --cip-setting <setting> --set <value> --cct-name <cct>
```

- Where `<setting>` is `DISABLE_CIP_GENERATION` and `IGNORE_CIP_ERROR`.
- An invalid project, path or unknown CCT results in an error.

3. To remove a CIP value from a project:

```
qacli pprops -P <project-path> --cip-setting <setting> --unset <value> --cct-name <cct>
```

- Where <setting> is DISABLE\_CIP\_GENERATION or IGNORE\_CIP\_ERROR.
- An invalid project, path or unknown CCT results in an error.

4. To reset a CIP setting in a project:

```
qacli pprops -P <project-path> --cip-setting <setting> --reset --cct-name <cct-name>
```

- Where <setting> is DISABLE\_CIP\_GENERATION or IGNORE\_CIP\_ERROR.
- An invalid project, path or unknown CCT results in an error.

## Miscellaneous Settings

There are several miscellaneous settings that can be modified:

INTER\_TU\_ANALYSIS. This is a flag field that can take either '-' or '+' as a value.

LOG\_LOCATION\_TYPE. This determines the location of the logfiles - it overrides any user defined setting (see [Log](#)). It can be either:

- DEFAULT : Whatever the global user configuration is i.e. Project or UDL. UDL is the default.
- PROJECT : Log data to the prqa/configs/<name>/logs sub directory
- UDL : Log data to the User Data Location (%localappdata%\Perforce\Helix-QAC-<version>\app\logs or ~/.config/Perforce/Helix-QAC-<version>/app/logs/)

To list the miscellaneous settings for a project

```
qacli pprops --view-misc-settings -P <path>
```

- An invalid project or path results in an error.

To set a value in a project:

```
qacli pprops -P <project-path> --misc-setting <setting> --set <value>
```

- Where <setting> is INTER\_TU\_ANALYSIS.
- An invalid project, path or value results in an error.

To remove a CIP value from a project.



```
qacli pprops -P <project-path> --misc-setting <setting> --unset <value>
```

- Where <setting> is INTER\_TU\_ANALYSIS.
- An invalid project, path or value results in an error.

To reset a CIP setting in a project:

```
qacli pprops -P <project-path> --misc-setting <setting> --unset <value>
```

- Where <setting> is INTER\_TU\_ANALYSIS.
- An invalid project, path or value results in an error.

## Report

The QA-CLI 'report' subcommand allows you to generate analysis reports based on the analysis results of a Helix QAC project.

Verbose help on the command can be accessed as follows:

```
qacli report -h
```

### Note:

The analysis results for a Helix QAC project will include any CMA results relating to the specified project.

For the report types available, and output locations, refer to Standard Report Types.

To generate a report, issue the following command, where acronym is one of the report types (that is to say, CRR, HMR, MDR, RCR or SUR):

```
qacli report -P <project-path> --type <acronym> [-K <config-name>]
```

The shortcut for the `--type` option is `-t`.

To specify a report to be generated for a specific set of source files within a Helix QAC project, the path to each source file must be placed in a filelist, which is a text file made up of one path per line. To generate the report:

```
qacli report -P <project-path> -t <acronym> --files <path-to-filelist> [-K <config-name>]
```

The shortcut for this option is `-F`. The option will be ignored if `--type` is set to 'RCR'.

If you wish to skip dependency checking so that reports are generated even for those files for which analyses remain out of date, for example because RCMA is in the toolchain, then use the following command:

```
qacli report -P . -t <report_type> --ignore
```

### Note:

If you do use the `--ignore` (or `-i`) option given above, then you should be aware that, because the

dependency checking has not been run (and subsequent analysis has also not been run), the reports may not be *final*: that is to say, they may not be 100% complete. For this reason, all reports generated using this option are marked as *Draft* in the HTML/XML.

By default all reports go to a fixed location (`prqa/configs/<config-name>/reports`), this can be overridden by specifying `--output-path` (or `-o`)

```
qacli report -P . -t <report_type> --output-path my_dir
```

That will create a report with a default name. If you wish to specify a name then use the `--output-name` (or `-n`) option:

```
qacli report -P . -t <report_type> --output-path my_dir --output-name my_reports
```

**Note:**

The appropriate file extension should not be specified - that will automatically be added.

**Information:**

You must have write access to this directory. If the directory does not exist it will be created.

Various stages in the report generation process can be done in parallel, which may reduce the time needed to produce the report. The `-j` (or `--jobs`) option specifies the number of simultaneous jobs to run in parallel. Some experimentation may be needed to determine the optimal value - using an excessively large value is counter-productive and may increase the report generation time. If not specified, then a default value of four is used.

## Sync

The QA-CLI `sync` subcommand allows you to synchronize files automatically in a Helix QAC project.

Verbose help on the command can be accessed as follows:

```
qacli sync -h
```

The format for the 'sync' command is as follows:

```
qacli sync [-P <directory>] -t <BUILD_LOG|JSON|MONITOR|MSVS> [-o <output-path>] [-x] [-Z] [--] [-h] <file|script>
```

where:

- `'-P'` (or `--qaf-project`) specifies the Helix QAC project directory.
- `'-Z'` (or `--optimize`) is an optional switch that optimizes the project after synchronization. For more information, refer to [Optimizing a Project](#).
- `'-t'` (or `--type`) specifies the build type.
- `'-x'` (or `--sync-exit-code`) Do not return a standard Helix QAC exit code - return whatever the sync/build system returned. This option is only really useful for MONITOR synchronisation.
- `'-o'` (or `--output-progress`) specifies the file that Helix QAC synchronisation progress will be piped to. No output from Helix QAC will be sent to the console. Any output originally coming from the underlying build system will continue to be output on the console.

The other parameter depends on the build type, and the descriptions of the build types can be found in the following sub-sections

## MONITOR

Using this option with an executable script or command as a parameter will start the build process. The process will be monitored, and any source files and dependent information will be extracted and added to the specified Helix QAC project.

If your build command contains more than one word, wrap it in quotes. If your build command is complex, you are advised to place it in an executable script, and pass the path of this script as the argument.

Any existing files in the project will be removed if they are not in the current build - this is useful for keeping Helix QAC projects synchronized with your build projects.

```
qaccli sync --type MONITOR make
```

This will issue the Linux/UNIX 'make' command in the current directory and synchronize all build files and dependencies into the Helix QAC project within the current directory.

**Note:** It is important that all files are built, so your build should be 'cleaned' first. The MONITOR command replaces the qaccli admin --build-command, which is deprecated. Refer to Section [Deprecated Commands](#).

## BUILD LOG

This synchronization type allows you to add data using a build log. If your build system is able to generate a build log that abides by the following constraints:

- The build log is newline-separated for each source file that was compiled.
- The lines that contain a path to a source file also contain the Include paths and macro Definitions needed to compile the file.

then the resultant log file can be used to add the data to a Helix QAC project using the following command:

```
qaccli sync -P <project-path> --type BUILD_LOG <build-log-path>
```

If your build system does not support the creation of a build log in this format, then, for most IDEs and build systems, a log can be created by using a script that wraps your compiler. The following is an example using the *Bash* variant:

```
#!/bin/bash
echo "$@" >> build.log
exec /usr/bin/gcc "$@"
```

In this example, the script must be named 'gcc', made executable, and placed on the PATH prior to 'gcc'. During the build process, the build system will call the wrapper script instead of 'gcc' and a build log will be dropped in each directory from which the wrapper is executed. These build logs will now be suitable for using with the BUILD\_LOG command. You just need to ensure that you use an absolute path for the 'project-path' argument and issue the command from the same directory as the one containing the log file.

**Note:**

- The same method can be used for the Windows platform, using a Windows version of the above script example.
- The BUILD\_LOG command replaces qaccli admin --add-files, which is deprecated and may be removed in a future release (refer to [Deprecated Commands](#)).

## JSON

Synchronization can also be carried out using a file that is a JSON compilation database - refer to <http://clang.llvm.org/docs/JSONCompilationDatabase.html> for further details on this format.

A typical JSON example might be:

```
qacli sync -t JSON -P ~/samples/enum-eg compile_commands.json
```

where *compile\_commands.json* is:

```
[
{
    "directory": "/home/steve/Temp",
    "command": "g++ -std=c++11 -I . enum.cpp",
    "file": "/home/steve/Temp/enum.cpp"
}
```

Support has also been added for the arguments form of the command (which is produced by the Clang compiler), for example:

```
[
{
    "directory": "/home/steve/Temp",
    "arguments": ["g++", "-std=c++11", "-I", "diff.h", "-D", "Wibble"], "file": "File C.c"
}
```

**Note:** This command replaces `qacli admin --parse-json-db`, which is deprecated. Refer to [Deprecated Commands](#).

## MSVS

Synchronization with the MS (Microsoft) build system is possible using the plug-in provided that sits between the MS build system and the QA·CLI "sync" command. The plug-in allows VS (Visual Studio) projects to be synchronized into Helix QAC projects.

For example:

```
qacli sync -t MSVS -P . "--vsproj C:\Temp\Win32ProjectOne.vcxproj -p
C:\Temp\Win32ProjectTwo
```

**Note:**

Details of the options for the MS Build plug-in can be found in the QA Visual Studio manual.

## Upload

The QA-CLI 'upload' subcommand allows you to upload analysis results from a Helix QAC project to an external application. At present, supported external applications are Structure101 and Dashboard.

Verbose help on the command can be accessed as follows:

```
qacli upload -h
```

The full syntax is as follows:

```
qacli upload {-q|-s|-g <output-file>} [-P <directory>] [-K <config name>] [--
username <username>] [-p <password>] [-t <token>] [-r <directory>] [-v <file>] [--
upload-project <project>] [-u <directory>] [--upload-source <NONE|ALL|NOT_IN_VCS>]
[-a <ROOT|RELATIVE|ABSOLUTE>] [-S <snapshot>] [-U <[protocol://]host[:port]>] [-F
<file>] [--] [-h] <source-file>
```

**Note:**

- The analysis results for a Helix QAC project will include any CMA results relating to the specified project.
- -- (or --ignore-rest) is not used by this command and may be removed in a future release.

## Upload to Helix QAC Dashboard

The following command can be used to upload the analysis results of a Helix QAC project to a Helix QAC Dashboard project:

```
qacli upload -P <qaf-project> --qav-upload [-K <config name>] --url
<protocol://host:port>
--username <username> --password <password>] --password-file <file>
--token <token> --repository <directory> -vcf <file> --jobs <n>
--upload-project <project> --upload-source <NONE|ALL|NOT_IN_VCS>
--path-format <ROOT|RELATIVE|ABSOLUTE>
--snapshot-parent <parent> --snapshot-name <snapshot>
```

where:

- `qaf-project` gives the location of the Helix QAC project. It defaults to '.' if not specified. The shortcut for this option is '-P'.

- `qav-upload` flags that this upload is going to Helix QAC Dashboard. The shortcut for this option is `'-q'`.
- `upload-project` is the name of the Helix QAC Dashboard project that should receive the results (which may or may not be a Unified one). If the project does not exist, it will be created.
- `snapshot-name` is the optional name that will be given to the Helix QAC Dashboard snapshot created to hold the uploaded results. If no name is specified, then a default will be used based on the current date/time, in the format YYYYMMDDHHMMSS. The shortcut for this option is `'-s'`.
- `upload-source` can be set to one of: NONE, ALL, NOT\_IN\_VCS. This determines which code files are uploaded to Helix QAC Dashboard.
- `url` is the address of the Helix QAC Dashboard server. The shortcut for this option is `'-U'`.
- `username` is the Helix QAC Dashboard identifier for the user who is uploading. The shortcut for this option is `'-u'`.
- `password` is the user's Helix QAC Dashboard password. The shortcut for this option is `'-p'`.
- `password-file` is a file containing the user's Helix QAC Dashboard password. The shortcut for this option is `'-w'`.
- `token` specifies that an authentication token is to be used in preference to user credentials (`--username` and `--password`). Refer to [Specifying Helix QAC Dashboard Credentials](#) for token creation details. The shortcut for this option is `'-t'`.
- `path-format` denotes how paths are passed to Helix QAC Dashboard. Options are: ROOT (default), RELATIVE and ABSOLUTE. The shortcut for this option is `'-a'`.
- `repository` specifies the version control repository path passed to Helix QAC Dashboard.
- `vcf` is the file location of a local VCF to be uploaded to Helix QAC Dashboard.
- `snapshot-parent` is the optional name of the parent snapshot, to which the new snapshot will be directly connected. If this entry is not provided or this parent snapshot is not found, then the last uploaded snapshot is chosen as parent.
- `jobs` is the optional number of jobs to run in parallel for the Dashboard upload. If not supplied or if 0 is supplied, then `qaimport` will determine the value.

## Upload to Structure101

The following command can be used to upload the analysis results of a Helix QAC project to a Structure101 project:

```
qaccli upload -P <project-path> --s101-upload --upload-location <directory>
```

The shortcut for the `--s101-upload` option is `'-s'` and the shortcut for the `--upload-location` option is `'-u'`.

The following constraints exist for this option:

- A Structure101 license must be available from the license server.
- The project must be a C or C++ project.
- The project must have been previously analyzed and the analysis results must be available.
- The argument to the `--upload-location` option must specify an existing directory path.

## Upload of Selected Files

The following command can be used to upload selected Dashboard results from a Helix QAC project:

```
qacli upload -P <project-path> -q <source-file> --username <uname> --password <pwd> --  
upload-project <proj-name> --upload-source <NONE|ALL|NOT_IN_VCS> -U <server-url>
```

The following command can be used to upload selected Structure101 results from a Helix QAC project:

```
qacli upload -P <project-path> -s <source-file>
```

To specify the upload for a specific set of source files within a Helix QAC project, the path to each source file must be placed in a filelist, which is a text file made up of one path per line.

To upload a Dashboard filelist:

```
qacli upload -P <project-path> -q --files <path-to-file-list> --username <uname> --  
password <pwd> --upload-project <proj-name> --upload-source <NONE|ALL|NOT_IN_VCS> -U  
<server-url>
```

To upload a Structure101 filelist:

```
qacli upload -P <project-path> -s --files <file>
```

The above options can also be combined in order to upload the sum of the filelist and source-file.

To upload combined Dashboard results:

```
qacli upload -P <project-path> -q --files <file> <source-file>
```

To upload combined Structure101 results:

```
qacli upload -P <project-path> -s --files <file> <source-file>
```

When uploading to Dashboard several threads are created to enhance performance. The number of threads is based upon the number of cores (see qaimport). You can override the default number of threads by specifying the optional parameter `--jobs` option (`-j` for short):

```
qacli upload -P <project-path> -q <source-file> --username <uname> --password <pwd> --  
upload-project <proj-name> --upload-source <NONE|ALL|NOT_IN_VCS> -U <server-url> --jobs  
3
```

**Note:**

This parameter is generally not needed and should only be used when directed by support. Increasing the job count excessively will likely result in slower uploads.

## Generate Dashboard Configuration

It is possible to generate a Dashboard client configuration file for a specified Helix QAC project. This file can then be used for advanced Dashboard configurations. The option is primarily for use by support consultants and experienced users only.

The following command can be used to generate the file:

```
qacli upload -P <project-path> --generate-qav-config <output-file>
```

The shortcut for the `--generate-qav-config` option is `'-g'`.

## View

The QA-CLI 'view' subcommand allows you to output the diagnostics from the analysis results of a Helix QAC project.

Verbose help on the command can be accessed as follows:

```
qacli view -h
```

**Note:**

All features in this subcommand are currently restricted by license. The original source remains **Read Only** and, as such, it will never be modified or have its modification time changed under any circumstances, regardless of the commands invoked.

This command can output in several different formats:

**HTML** - For each source file, output results to an HTML file.

**STDERR** - Output results to standard error.

**STDOUT** - Output results to standard output.

**XML** - For each source file, output results to an XML file.

For those who possess a completely unrestricted license, it is possible to view results on the command line or output them to a file.

**Note:**

Diagnostics that are suppressed by a baseline will be presented as if they have not been suppressed.

There are several different styles of output available and can be chosen using the `--type` (`-t` for short). It can have the following values:

- **DIAGLST** : Outputs a list of diagnostics for each file (containing diagnostics).
- **ANNSRC** : Embeds the diagnostics output into each source file.
- **SUMMARY** : Outputs a summary of all diagnostics for the project into a single file.

If this option is omitted, then `-t DIAGLST` is assumed i.e. in a diagnostic list style.

Several options are common to all output styles and these are described below. Options that are specific to a particular style are described in subsequent sections.

To output the results for a Helix QAC project to standard output, issue the following command:

```
qacli view -P <project-path> --medium STDOUT [-K <config-name>]
```

The shortcut for this option is `'-m'`. To output to standard error, replace `STDOUT` with `STDERR` in the above command.

When the argument to the `'--medium'` option is set to `'HTML'` or `'XML'`, a directory can be specified. Files will be output to this directory:

```
qacli view -P <project-path> -m HTML --output-path <directory> [-K <config-name>]
```

The directory must exist and the shortcut for this option is `'-o'`. The output will mirror the directory structure of the source files. To flatten the output into a single folder then add `--collate-output` (`-c` for short). Duplicate files will not be overwritten and their name will have a unique number appended, for example:

```
foo.txt
foo(1).txt
```



foo(2).txt

Datasets can be quite large and difficult to navigate on the command line, and so there is a series of switches that hide or show various subsets of data, as described below.

To show rule information with each primary diagnostic (but not sub-diagnostics):

```
qaccli view -P <project-path> -m STDOUT --rules [-K <config-name>]
```

The shortcut for this switch is '-r'.

**Note:**

The --rules argument requires either %r or %v to be present in the --format string parameter (refer to [Default Format Strings](#)).

To include suppressed messages in the output:

```
qaccli view -P <project-path> -m STDOUT --suppressed-messages [-K <config-name>]
```

The shortcut for this switch is '-s'.

**Note:**

Diagnostics that are suppressed by baselining will always be presented, regardless of this switch.

To prevent diagnostics from header files from being displayed:

```
qaccli view -P <project-path> -m STDOUT --no-header-messages [-K <config-name>]
```

The shortcut for this switch is '-n'.

The amended default for every output mode is to display line numbers in the annotated source, although the --no-line-numbers argument (-b for short) has been made available to remove the display of line numbers where required, for example:

```
qaccli view -P <project-path> -m STDOUT -ab [-K <config-name>]
```

To display results for a specific source file within a Helix QAC project, the path to the source file can be provided:

```
qaccli view -P <project-path> -m STDOUT <path-to-file> [-K <config-name>]
```

To display results for a specific set of source files within a Helix QAC project, the path to each source file must be placed in a filelist, which is a text file made up of one path per line. To view this, use the following command:

```
qaccli view -P <project-path> -m STDOUT --files <path-to-filelist> [-K <config-name>]
```

The shortcut for this option is '-F'.

To list multi-homed messages (that is to say, messages that are not generated for a specific project file) to standard output:

```
qaccli view -P <project-path> -m STDOUT --multi-homed-messages [-K <config-name>]
```

The shortcut for this option is '-M'.

**Note:**

Switch shortcuts can be chained, as shown in the example below:

```
qaccli view -P <project-path> -m STDOUT -rs [-K <config-name>]
```

To report source file paths relative to the project rather than as absolute paths, use the `--relative-paths` option:

```
qaccli view -p <project_path> -m STDOUT --relative_paths
```

The shortcut for this option is `'-R'`

## Annotated Source

To display diagnostics as annotations in the original source code (both source and headers) you should output in annotated source style:

```
qaccli view -P <project-path> -m HTML --type ANNSRC [-K <config-name>]
```

You can add the `--one-line-only` argument (`-1` for short) to output only those lines from the source code that have diagnostics associated with them:

```
qaccli view -P . -m STDOUT -t ANNSRC -1
```

Rather than having individual entries for each source file and header separately, it's possible to embed the headers into the annotated source listing using `--embed-header-messages` argument (`-e` for short). This form of output is not supported for multi-project CMA solutions.

```
qaccli view -P . -m STDOUT -t ANNSRC -e
```

The diagnostic output for included files will only show the top level diagnostic (no subdiagnostics will be output).

It is also possible to specify how any included diagnostics are formatted using their own formatting string; `--header-format-string` argument (`-H` for short). The content is identical to the `'--format'` option (refer to [Formatting the Diagnostic String](#)). If no value is provided and a value for `'--format'` is supplied then this same value will be used as the header format. If both a header and `'--format'` are not supplied then the following default values will be used:

For output to STDOUT and STDERR:

```
%?u==0% (%?h% (Err%:Msg%) %:-->%F:%l:%c %) (%N) %R(%u, ) %t%?v% (\n?v%)
```

For output to HTML:

```
%?u==0% (%?h% (Err%:Msg%) %:-->%F:%l:%c %) (<a href=\"%H\">%p-%N</a>) %R(%u, ) %t%?v% (<br/>%v%)
```

For output to XML:

```
%l%c%n%t
```

These values are similar to the defaults for `'--format'`, but with leading `%%C% (%^%)` removed.

## SUMMARY

To output a concise summary of a project use `--type SUMMARY`, for example

```
qaccli view -P <project-path> -m STDOUT --type SUMMARY [-K <config-name>]
```

This will output the summary to the console, where it can be redirected to a file if needed.

Naturally, you can also output to a HTML or XML file. By default, the file will be called `severity_summary.xml` (or `.html`) and be in the current working directory. To output to another directory, use the `--output-path`, for example:

```
qaccli view -P <project-path> -m XML --type SUMMARY --output-path ../frodo
```

**Note:**

For XML output an additional schema that describes the file format will be output. It will be called: `cli_view_severity_summary.xsd`

For multi-project CMA solutions one file per underlying project will be created. The name will be `severity_summary.xml` (or `.html`), then `severity_summary(1).xml` etc

## Viewing CMA Projects

Viewing a CMA project follows the same format as a Helix QAC project. Issue the following command to output the results for a CMA project to standard output:

```
qacli view -C <cma-project-name> --medium STDOUT
```

The Helix QAC projects are listed in the order in which they are defined in the CMA project. If no Helix QAC project is specified (as above), then the default behavior is to use CMA data from the first in the list of Helix QAC projects in the CMA project.

Viewing a CMA project in HTML will result in the output going to a sub-directory of the specified directory. The sub-directory will be the Helix QAC project name. So if, for example, you have a CMA project called 'CMA1', with modules 'PRQA1' and 'PRQA2', a command such as:

```
qacli view -C CMA1 --medium HTML -o \tmp
```

will result in output going to `/tmp/PRQA1/` and `/tmp/PRQA2/` (Linux) or `C:\tmp\PRQA1\` and `C:\tmp\PRQA2\` (Windows).

## Filtering Messages

To reduce the volume of messages, you can filter the output using several criteria. Each of the filters is combined to dictate the final output.

To display only the specified warning messages, you can use the `--only ('-O')` option and specify the messages you want to see. Message numbers can be specified individually, as a list, as a range, or a combination of these, for example:

- `--only 1234`
- `--only 1234,1236,1240`
- `--only 2111,2200-2300`

To display only messages 1234 and 5678, you could issue the following command:

```
qacli view -P . --medium STDOUT --only 1234,5678
```

**Note:**

You should avoid whitespace adjacent to the punctuation characters. If you need to use whitespace, then enclose the values in quotes, for example `--only "1234, 5678"`.

To display only messages that do not contain the specified messages, you can use the `--nomsg ('-N')` option and specify the messages you want to filter. To display all messages except 1234 and 5678, you could issue the following command:

```
qacli view -P . --medium STDOUT --nomsg 1234,5678
```

To display only messages of a certain severity (or higher), use the `--min-severity` ('-s') option and specify the severity level. Valid values are in the range 0-9: 0 is acceptable, but will display everything. Values greater than 9 will filter everything.

To display high severity messages, a command such as the following might be appropriate:

```
qaccli view -P . --medium STDOUT --min-severity 7
```

The following example combines the filters and displays only high severity messages, ignoring message 1234:

```
qaccli view -P . --medium STDOUT -S 7 -N 1234
```

To limit the number of times that any message is displayed, you can use the `--max-count` (-X) option. The default of 0 shows no limit is applied.

```
qaccli view -P . --medium STDOUT --max-count 2
```

This command displays only the first 2 unique messages in each file.

## Formatting the Diagnostic String

To modify the format of the diagnostic string, the following option can be used:

```
qaccli view -P <project-path> -m STDOUT --format <format-string>
```

where `format-string` is a quoted string in which the following specifiers are recognized:

Specifier	Description
%B	Base file name
%c	Column number
%C	Column number less 1 (left column is 0)
%e	Message extension information, e.g. message reference text (requires <message_extension> tag in the RCF file)
%f	File name
%F	File name (absolute, including path)
%g	Message level
%G	Rule group(s) (comma separated)
%H	Message help file (absolute path)
%j	Suppression justification
%l	Line number
%m	Suppression Macro Name
%n	Message number (raw integer format)

Specifier	Description
%N	Message number (zero padded to four digits)
%p	Component for which the diagnostics are being produced (for example "qacpp-4.1")
%r	Rule(s) (comma separated)
%S	Suppression type
%t	Message text
%u	Context message depth
%v	Verbose message text
%Y	Severity
%Z	Rule help file(s) (absolute path, comma separated)
%^	if the column is not zero and the message is not a submessage for different line or filename, update last location, show the caret and insert a newline.

**Note:**

The suppression type is a single digit decimal bitmap that denotes the types of suppression that are active for a diagnostic. A value of '0' indicates that no suppressions are active. The following bits may be set:

## Table of suppression types

Bit	Suppression Type
0	Comment Suppression
1	Pragma Suppression
2	Baseline Suppression
3	Macro Suppression
4	Dashboard Suppression

## Conditional Formatting

Message formatting can also be configured so as to apply only in certain conditions. This allows you to specialize message display, for example according to the message level. You can enter a conditional statement in the format:

```
%?<condition> %(<true condition> [%:<false condition>] %)
```

The false branch is optional and, if it is not supplied, nothing is displayed. The condition field consists of a letter, and optionally, a conditional operator and value.

The following conditional variables can hold any legitimate value, and must be tested using a relational operator:

c	Column number
g	Message level
l	Line number
n	Message number
S	Suppression Type
u	Context message depth
Y	Severity level

The possible values for Message Level (%g) are:

0	Information
1	Warning
2	Error
3	User Message

For example, to display Err for Level 2 messages and Msg for all others, the conditional statement would look like:

```
%?g==2%(Err%:Msg%)
```

Primary messages have a context message depth (u) of zero, and context messages are level 1 or greater. Typical conditional processing of context messages is:

```
%?u==0%(primary message %: context message %)
```

**Note:** The above Message Levels are not to be confused with the RCF (Rule Configuration File) Message Levels documented in the Helix QAC for C and Helix QAC for C++ component manuals, and with the Severity levels documented in [Viewing and Filtering the Diagnostics](#).

The following conditional variables can only be tested for a change in their values:

C	File name, line and column number
F	File name
L	File name and line number

For example, a simple check for a file name is:

```
%?F%(\n%f%)
```

In this example, if the file name has changed (F), the new file name will be displayed on a new line (\n%f). The conditional variable C can be used to determine the positioning of the warning location caret (^). In the default message format, the caret is omitted for messages with the same location as the previous message. To replicate this, use the following conditional statement:

```
%?C% (%^%)
```

This statement will compare the file name, line and column number (C) of the current message with the previous message. If the location is different, the statement is considered True and the caret is displayed, followed by a new line, which is added implicitly.

Finally, properties of a message can be queried with the following Boolean conditional variables:

h	Is message a hard error (level 9)
j	Are suppressions present?
v	Is verbose text present?

For example, the following statement:

```
%t%?v% (\n%v%)
```

will print the message text (%t) followed by the verbose text on a new line (\n%v), if it exists (%?v).

**Note:** The %j conditional will only be triggered when the suppression message is not empty.

## Message Text Control

There is another set of format specifiers that control how text is handled within warning messages. These should appear at the beginning of the format string.

%i(str)	Indent wrapped text: when text is wrapped, it is indented with <str>.
%q	Save the location for the next message (used for %F, %L, %C tests).
%Q	Reset the location for the next message. <div><b>Note:</b> %F, %L, %C variables will be set as true for the next message. %Q takes precedence over %q.</div>
%R(n,c)	Print "n" times the character specified by "c".
%s(n)	Limit message size: text is truncated to n characters.
%w(n)	Character wrap: message text wraps at column n.
%W(n)	Word wrap: message text wraps after the last word before column n.

To limit the size of message output to 80 characters per line without word breaks, include the specifier %W(80) in the format string.

The %q and %Q entries can be used to override the default behavior of %, or to implement similar functionality from scratch. As an example of %R(num,char) usage:

```
%?c>0% (%?C% (%R (%C, .) ^\n%) %)
```

will cause the following display of caret lines:

```
.....^
```

## An Example Message Format

The following is a typical message format that displays the most relevant information about a warning message:

```
%W(80)%i( )%?C%(%^%)%f(%l) ++ %?h%(ERROR%:WARNING%) ++: <=%g=(%n) %t
```

The components of this string are:

%W(80)	Message text wraps after the last word before column 80.
%i( )	Each hanging indent is indented by three columns.
%?C%(%^%)	A caret is displayed if the location of the error is different from that of the previous error.
%f(%l) ++	The file name and line number (in brackets) is displayed, followed by '++'.
%?h%(ERROR%:WARNING%) ++:	If the message is a hard error, ERROR is displayed. Otherwise, WARNING is displayed, followed by '++:'.
<=%g=(%n)	The message level is displayed (preceded by '<='). This is followed by the message number (preceded by '=').
%t	The message text is displayed.

A warning message with the above format would look like:

```
int i;
^
test.c (1) ++ WARNING ++: <=2=(3408) 'i' is externally visible. Functions and
variables with external
```

## Default Format Strings

The default format string, which is compatible with Eclipse and GNU Emacs C++ mode (that is to say, it will link to the correct file, line and column), is:

```
%F(%l,%c): %?u==0% (%?h%(Err%:Msg%)%:-->%) (%G:%N) %R(%u, )%t%?v%(\n%v%)
```

The default format string for annotated source (STDOUT and STDERR) is:

```
%?C%(%^%)%?u==0% (%?h%(Err%:Msg%)%:-->%F:%l:%c %) (%N) %R(%u, )%t%?v%(\n%v%)
```



The default format string for annotated source (HTML) is:

```
%?C% (%^%)%?u==0% (%?h% (Err%:Msg%)%:-->%F:%l:%c %) (<a href=\"file://%H\"> %p-%N</a>) %R  
(%u, ) %t%?v% (<br/>%v%)
```

## Formatting in Windows command files (.bat)

If specifying one of the format commands (`--format`, `--header-format` or `--xml-format`) within a Windows command file (.bat) the % will indicate a variable name will follow, to avoid this then specify the % character twice. For example, an interactive user could use a command such as:

```
qaccli view -P . -m XML --xml-format "%l%c%n%t%G%r"
```

However, if you wanted this command in a Windows command file it should look like:

```
qaccli view -P . -m XML --xml-format "%l%%c%n%%t%%G%%r"
```

## XML Output

Output to an XML file is possible when the '`--medium`' option is set to 'XML'. The most basic command to do this is:

```
qaccli view -P . -m XML
```

This will output the entire project, which is based in the current directory. As no output directory is specified a directory called '`_SOURCE_ROOT`' will be created in the current directory and the files will be located underneath in a tree structure mirroring the source directories. The files will be named similarly to the original source files but with the extension '`-diaglist.xml`'. If annotated source format is requested then the name will end with '`-annsrc.xml`', for example:

```
qaccli view -P . -m XML -a
```

Each file contains a standard set of elements which are described in detail below. Additionally, the content can be enhanced by using the following options; '`--xml-format`', '`-format`' and '`--annotated-source`'.

## XML Indent Size

It is possible to customize the number of spaces or tabs that will be output for each level of indentation (XML only) using the option `--xml-indent` (`-i` for short). Positive numbers indicate spaces, negative numbers tabs. The default is -1 i.e. a single tab character.

Information:

To retain compatibility with releases earlier than 2019.1, set this value to 4

## XML File Format

Each file will start with the following header giving creation date, time, version information and the command that was used during the creation.

```
<?xml version="1.0" encoding="UTF-8"?>
<Header>
  <Title>Helix QAC XML Output</Title>
  <Date>07/09/2018</Date> <Time>09:19:38</Time>
  <CreationCommand>/home/steve/Perforce/Helix-QAC-2019.1/common/lib/qacli view -P .
-m
  <Version>2.4.0.9593-qax</Version>
</Header>
```

Then a section will detail the file being processed and its full absolute filename.

```
<File>
  <Name>/home/steve/.config/Perforce/Helix-QAC-2019.1/samples/
    sample_cgicc_diff-2.4.0/src/cgic
```

If annotated source output has been requested then the code will be output with the corresponding line number:

```
<File>
  <Source line="1">#include <iostream></Source>
```

Then, there will be a separate element for each diagnostic that is produced. The content of the diagnostic is controlled by the '--xml-format' and '--format' options:

```
<Diag id="1">
  <Line>0</Line>
  <Column>0</Column>
  <MsgNum>999</MsgNum>
  <MsgText> QAC++ Deep Flow Static Analyser 4.3.0-41497</MsgText>
</Diag>
```

A diagnostic may contain 0 or more sub-diagnostics. If any are present they will be denoted with the SubDiag element and contain an attribute with an incrementing id, for example:

```
<Diag id="94">
  <Line>119</Line>
  <Column>15</Column>
  <MsgNum>2647</MsgNum>
  <MsgText>This class declares some special member functions
    </MsgText>
  <SubDiag id="1">
    <Line>128</Line>
    <Column>3</Column>
    <MsgNum>1594</MsgNum>
```

```

        <MsgText>'MultipartHeader (::cgicc::MultipartHeader
            const &)' declared here.</Msg>
    </SubDiag>
    <SubDiag id="2">
        <Line>129</Line>
        <Column>4</Column>
        <MsgNum>1594</MsgNum>
        <MsgText>'~MultipartHeader()' declared here.</MsgText>
    </SubDiag>
</Diag>

```

At the end of the file will be two elements indicating if there were any analysis errors and the return code of the analysis process. Any non zero value should be investigated.

```
<AnalysisErrorCount>0</AnalysisErrorCount>
```

```
<AnalysisExitStatus>0</AnalysisExitStatus>
```

Finally, an optional element may be output if there are no analysis diagnostics for a file:

```
<NoDiagsOK>true</NoDiagsOK>
```

A value of 'true' indicates the file has been analysed and there are no diagnostics. A value of 'false' indicates that no analysis has been done on the file.

## XML Content

The content of the 'Diag' element is controlled by the '--xml-format' and '--format' options. The '--xml-format' option is a quoted string in which the following specifiers are recognised. If present, the corresponding element will be present in the diagnostic output:

## Specifier to XML mapping

Specifier	XML Tag	Description
%B	<BaseName>	Base file name
%c	<Column>	Column number
%C	<ZeroColumn>	Column number less 1 (left column is 0)
%e	<MsgExtText>	Message extension information
%f	<FileName>	File name
%F	<FilePath>	File name (absolute, including path)
%g	<MsgLevel>	Message level
%G	<RuleGroup>	Rule group(s) (comma separated)

Specifier	XML Tag	Description
%H	<HelpPath>	Message help file (absolute path)
%j	<SuppJust>	Suppression justification
%l	<Line>	Line number
%m	<SuppMacroName>	Suppression Macro Name
%n	<Msgnum>	Message number (raw integer format)
%N	<PadMsgNum>	Message number (zero padded to four digits)
%p	<Producer>	Producer component (e.g. qacpp-3.1)
%r	<RuleNum>	Rule(s) (Ccomma separated)
%S	<SuppMask>	Suppression type
%t	<MsgText>	Message text
%u	<Depth>	Context message depth
%v	<VerboseText>	Verbose message text
%Y	<Severity>	Severity
%z	<RuleHelpPath>	Rule help file(s) (absolute path, comma separated)

If any other specifiers are requested (such as Conditional Variables or Message Text Control variables) these will have no effect and will be silently ignored.

If no xml-format string is specified then the following will be used:

```
%l%c%n%t
```

The '`--format`' option (refer to [Formatting the Diagnostic String](#)) produces formatted output. This formatted text can also be output in the XML by specifying a non empty '`--format`' string. In the XML it will be shown with the '`<Formatted>`' element.

## XML Schema

A schema is provided to describe the XML output, it covers all three output formats; Diagnostic List, Annotated Source and Embedded Annotated Source. It is located in the installation directory:

```
<install>\Helix-QAC-2019.1\common\schemata\cli_view.xsd
```

This file will also be copied to the output directory when any XML files are generated.

## XML Examples

Provide basic XML output with no embedded source code and no custom formatting:

```
qaccli view -P . -m XML --format ""
```

This will produce XML with diagnostics looking like:

```
<Diag id="419">
  <Line>489</Line>
  <Column>1</Column>
  <MsgNum>1090</MsgNum>
  <MsgText> Tab found in source.</MsgText>
</Diag>
```

To give basic XML output with no embedded source code and default custom formatting:

```
qaccli view -P . -m XML
```

This will produce XML with diagnostics looking like:

```
<Diag id="419">
  <Formatted>/home/steve/.config/Perforce/Helix-QAC-2019.1/samples/sample_cgicc_
diff-2.4.0</Formatted>
  <Line>489</Line>
  <Column>1</Column>
  <MsgNum>1090</MsgNum>
  <MsgText> Tab found in source.</MsgText>
</Diag>
```

To give XML output with only the Line, Column and Message Number:

```
qaccli view -P . -m XML --format "" --xml-format "%l%c%n"
```

This will produce XML with diagnostics looking like:

```
<Diag id="419">
  <Line>489</Line>
  <Column>1</Column>
  <MsgNum>1090</MsgNum>
</Diag>
```

## Deprecated Commands

The following table gives a list of deprecated commands, including the version in which they were deprecated, and, if applicable, the version of the software from which they were removed ("tbc" = "to be confirmed"). Any item marked as deprecated that has not yet been removed, may be removed in a future release.

## Table of deprecated commands

Command	Deprecated	Removed	Notes
<code>--reuse-db</code>	2020.1	tbc	Option now no longer needed.
<code>--use disk-storage</code>	2020.1	tbc	Option now no longer needed.
<code>admin --debug-level</code>	2019.2	tbc	See <code>qacli log --set-level</code>
<code>--ignore_rest</code>	2019.2	tbc	Replaced with <code>--ignore-rest</code>
<code>--reuse_db</code>	2019.2	tbc	Replaced with <code>--reuse-db</code>
<code>sync --type GRADLE</code>	v2.4.0	tbc	Java component has been deprecated.
<code>sync --type MAVEN</code>	v2.4.0	tbc	Java component has been deprecated.
<code>analyze --repeat &lt;n&gt;</code>	v2.2.0	tbc	The <code>--repeat &lt;n&gt;</code> option has been replaced with the <code>--inter-tu-dataflow</code> option.
<code>admin --add-files &lt;script&gt;</code>	v2.1.3	tbc	See <code>qacli sync --type BUILD_LOG</code>
<code>admin --build-command &lt;script&gt;</code>	v2.1.3	tbc	See <code>qacli sync --type MONITOR</code>
<code>admin --parse-json-db &lt;script&gt;</code>	v2.1.3	tbc	See <code>qacli sync --type JSON</code>
<code>admin --ncf &lt;ncf-file&gt;</code>	v2.0.0	tbc	Now specified using the Namecheck component.

## 13 | Upgrading Helix QAC

This section covers the upgrading of Helix QAC and its components.

### User Settings

If you have one or more existing versions of Helix QAC configured with a license server, then, after installing a new version of Helix QAC, an Import Configuration dialog box will be displayed on initial startup.

This gives you the option to import the user settings from one of the other Helix QAC installations. If you select "Don't import anything", then Helix QAC will be initialized with the default settings and will require you to specify license server details.

If, however, you opt to import an existing configuration, then the following will be copied from the selected Helix QAC to the newly installed version:

- Helix QAC License Server List.
- The recently used Helix QAC Dashboard server addresses and user names.
- The component search path.
- The number of threads to use for analysis.
- The language to display.
- The desktop location and size of the GUI Window.
- The GUI frame layout.
- The maximum number of recently opened projects.
- The list of recent projects.

### Upgrading Projects

When upgrading to a newer version of Helix QAC, you are able to continue using Helix QAC projects created from previous versions of Helix QAC. The settings in the projects will be updated so that you can carry on working with them.

The upgrade process will create an additional folder in your Helix QAC Project called "upgrade". This folder will contain an upgrade log and a backup of any files prior to being modified.

### Missing Component Dialog Box

When upgrading a project, you may see a Project Warning Dialog.

This will occur when a Helix QAC project references an old component that is not installed on the newer version of Helix QAC. In this case, you can do one of two things:

1. Update the **Analysis Toolchain**.
2. Install the missing component.

## Updating the Analysis Toolchain

The **Analysis Toolchain** can be updated from the **Project Properties** dialog box. The Helix QAC for C component in the **Analysis Toolchain** is greyed out, meaning that it is not available. There is a Helix QAC for C component in the "Available Components" that can be selected to replace the missing component.

## Installing the Missing Component

You may install the missing component if the package exists for it. Please contact Programming Research Ltd to determine if there is an associated installer.



## 14 | Adding Translations to Helix QAC

Helix QAC translations for both the GUI and CLI interfaces can now be added for any language of your choice, using the third party Qt Linguist tool<sup>1</sup>. These translations will then appear in both the GUI and CLI interfaces.

The following steps describe the process of adding a translation to Helix QAC, using the German language as an example.

- You need to find the `default.ts` in the translations subdirectory of the Helix QAC installation directory.

Assuming that Helix QAC 2019.1 is installed, on Windows this location would be:

```
C:\perforce\Perforce-Helix QAC-2021.1\common\translations
```

On Linux, assuming that Helix QAC was installed in the home folder of user bloggs, this location would be:

```
/home/bloggs/Perforce-Helix QAC-2021.1/common/translations/
```

- The file `default.ts` must be copied to create a new file, and this new file given a suitable name. Because, in this case, German translations are to be added, it is recommended that the copied file is renamed to `de.ts`, since `de` is the two-character language code for the German language. A link to a table containing the ISO list of language codes is provided in the links at the end of this section.
- The newly created file can then be opened using the **Qt Linguist** tool. On opening the file, **Qt Linguist** will prompt you to select the source and target languages. The source language must be left unchanged but you can select your target language. For this example, German must be selected.

**Note:** Links to both the **Qt Linguist** tool and manual can be found at the end of this section.

- Translations may then be added using **Qt Linguist**.
- Once the translation file has been updated and saved, the translations can be 'released' to generate a `.qm` file. This can be achieved by using either **Qt Linguist** (select **File : Release**), or the `terminal/command prompt`.

To generate the `.qm` file in a `terminal/command prompt`, you need to navigate to the `translations` subdirectory and run the `lrelease` command as follows:

On Linux:

```
cd /home/bloggs/Perforce-Helix QAC-2021.1/common/translations/  
lrelease de.ts
```

On Windows:

```
cd C:\Perforce\Perforce-Helix QAC-2021.1\common\translations\ lrelease.exe de.ts
```

The file that is generated in our example is called `de.qm`, which is the file used by Helix QAC for translation. This new language will appear in the **Admin : Languages** menu of the GUI and can also be used when setting the language using the CLI.

**Note:**

- For messages containing strings of the type `"%1"`, `"%2"`, and so on, these strings must also be

---

<sup>1</sup>This is a commercial tool and there may be a license overhead.

present in the translation. Removal of these strings will cause unexpected behavior in Helix QAC.

For example, considering the following source:

*Open files %1 and %2*

A valid German translation would be:

*Dateien %1 und %2 öffnen*

- If a *.ts* file is modified, the release step must be repeated to generate a new *.qm* file
- It is not necessary to translate all strings before running the release step. Unfinished translations are simply ignored by Helix QAC.
- The above procedure can be repeated several times to add translations for multiple languages
- Using the two-character language codes to name the *.qm* files is recommended since using this convention causes the localized name of the language to be displayed in the **Admin : Languages** menu. Considering the above example, the additional language would appear as **Deutsch** and not **German**.

#### Useful links:

- [List of ISO Language Codes](#)
- [Qt Tools Download Link](#)
- [Qt Linguist Manual](#)

#### Note:

It is recommended that you download the Qt4 rather than Qt5 package, as it will then match the version used by Helix QAC.

## 15 | Glossary

### Glossary of Terms

Below is a table on common terms that are found in this and other Helix QAC manuals.

#### Table of Helix QAC Terms

Term	Description
Baseline	To take a snapshot of the current diagnostics issues in your code for comparison with later changes.
File Based Analysis	To analyse files of a project without considering diagnostic issues from other code files.
Local Baseline	A Baseline stored on your local computer.
Message	A diagnostic message indicating a potential issue in your code.
Helix QAC Project	This contains all the information that Helix QAC requires in order to analyze your code.
Raw Source Analysis	To analyze a file independently, for example, to analyze a preprocessed file in order to debug configuration issues.
Rule	A combination of messages that are checked to determine whether or not a particular restriction has been broken.
Synchronize Analysis	This feature enables a project to be built in its own environment and monitored for information that is required for analysis in Helix QAC.
Toolchain	The components that will be used to analyse your code
Unified Project	Project definitions which are stored in Helix QAC Dashboard.
VCS Baseline	A Baseline that is stored via your Version Control System.

### Glossary of Acronyms

Below is a table on common Acronyms that are found in this and other Helix QAC manuals.

Acronym	Description
ACF	Analysis Configuration File
CCT	Compiler Compatibility Template

Acronym	Description
CIP	Compiler Include Paths
CLI	Command Line Interface
CMA	Cross Module Analysis
CRR	Code Review Report
GUI	Graphical User Interface
HICPP	High Integrity C++
ITU	Inter Translational Unit
MDR	Metrics Data Report
MISRA	Motor Industry Software Reliability Association
MTR	Multi-threading and Resource
M2CM	MISRA C 2004 Compliance Module
M3CM	MISRA C 2012 Compliance Module
RCF	Rule Configuration File
RCMA	Relational Cross Module Analysis
RCR	Rule Compliance Report
SUR	Suppression Report
TU	Translation Unit
UMF	User Messages File
VCF	Version Control File
VCS	Version Control System

## A | QA-CLI Return Codes

The following table contains details of the return codes that could be returned by QA-CLI:

Table of QACLI Return Codes

Code	Definition
0	Command Success
1	Command Line Parse Failure
2	Command Processing Failure
3	Command Success, with some warnings
8	Report did not complete successfully
9	Component did not complete successfully
10	Mutually Exclusive Operation Encountered on the Specified Project
11	Filesystem Read Failure
12	Filesystem Write Failure
13	Unknown Filesystem Permissions Failure
14	Illegal Component Command
15	Server Connection Failure
16	Missing Configuration
17	Empty Helix QAC Project
18	Helix QAC License Failure
19	File(s) Not Found Failure
20	Helix QAC Configuration File Exception
21	Helix QAC Deployment Exception
22	Filesystem Exception
23	Helix QAC Project Exception
24	Unknown Exception
110	Configuration error calling component.

Code	Definition
111	Unable to acquire license for component.
118	System failure in component processing analysis data.
119	Fatal exception in component.
121	Component unable to create the intermediate DB in the stated output location.
122	Component unable to read intermediate DB structure.
123	Incorrect user credentials supplied to component.
125	Unable to initiate communication with the server, or mismatched versions between client and server.
126	Communication unexpectedly closed (client started and then server disconnected).
127	Server config failure.
128	Server port failure.
129	Server DB failure.
130	Specified qav::prqavcs parameter file does not exist.
131	Specified project not ready to receive this snapshot - processing earlier snapshot.
132	Generated output folder does not contain any entity data (badly formed).

## B | Keyboard Shortcuts

The following table contains the global keyboard shortcuts for Helix QAC:

Table of Global Helix QAC Shortcut Keys

Keyboard Shortcut	Description
Ctrl + N	New Project
Ctrl + Alt + O	Open Project
Ctrl + Alt + C	Close Project
Alt + A + B	Synchronize Project
Ctrl + Alt + P	Open Project Properties
Ctrl + Alt + E	Exit Helix QAC
Ctrl + Alt + A	Open License Server Management Dialog Box
Alt + F + P	File-Based Analysis of the Project
Alt + C + P	Clean the Project
Alt + P + P	Run CMA Analysis
Alt + A + R	Open Raw Source Analysis Dialog Box
Alt + A + S	Open Analysis Settings Dialog Box
Alt + R + P	Generate a Report for the Project
Alt + R + C	Generate a Report for the selected files
Alt + U + P	Upload results of project to Dashboard
Alt + U + C	Upload results for selected files to Dashboard
Alt + W + D	Toggle On/Off Analysis Results/Diagnostics Panel
Alt + W + E	Toggle On/Off Whole Project Analysis Errors Panel
Alt + W + C	Toggle On/Off Message Levels Panel
Alt + W + R	Toggle On/Off Rule Groups Panel
Alt + W + F	Toggle On/Off Files Panel
Alt + W + P	Toggle On/Off Recent Projects Panel

Keyboard Shortcut	Description
Ctrl + Alt + T	Toggle the Toolbar On and Off
Ctrl + Alt + R	Reset Layout
Ctrl + Shift + W	Close all Source code tabs in the Editor
Ctrl + Shift + S	Save Changes to all Source code tabs in the Editor
Alt + Shift + W	Open the Welcome Page
F1	Open the Helix QAC Help

The following table contains the keyboard shortcuts when the input focus is on a source file within the Editor:

Table of Source Code Editor Shortcut Keys

Shortcut Key	Description
Ctrl + S	Save Changes in the source file
Ctrl + Z	Undo Changes in the source file
Ctrl + Y	Redo Changes in the source file
Ctrl + X	Cut text in the source file
Ctrl + C	Copy text in the source file
Ctrl + V	Paste text in the source file
Ctrl + F	Proceed to next item as specified in the search text box
Ctrl + B	Proceed to previous item as specified in the search text box
Ctrl + W	Closes the current source file tab

The following table contains the Keyboard Shortcut Keys when the input focus is on the Analysis/Results Diagnostics panel within Helix QAC:

Table of Analysis/Results Diagnostics Panel Shortcut Keys

Shortcut Key	Description
Alt + Up Arrow	Jump to the first page of results
Alt + Left Arrow	Move to the previous page of results
Alt + Right Arrow	Move to the next page of results
Alt + Down Arrow	Jump to the last page of results



## C | Validation

Various parameters may be entered by the user and these will be validated when appropriate. The approach is to allow anything except items that are known to cause issues either with the application or the underlying OS.

- [Filenames](#)
- [Pathnames](#)

### Filenames

Any item that can be used to create a file or directory will have validation applied to it. This includes the Helix QAC Project Directory, Multi-Configuration Name, Project Name, CMA Project Name, Unified Project Name, Upload Snapshot Name, & Upload Project Name (for the additional restriction on this, please refer to the final note below).

The following restrictions are applied to these items:

1. The name must be no more than 100 characters. This reduces the chance of a breach of the 260-character limit imposed by most versions of Windows on file paths.
2. The following characters are not permitted:
  - < (less than)
  - > (greater than)
  - : (colon)
  - " (double quote)
  - / (forward slash)
  - \ (backslash)
  - | (vertical bar or pipe)
  - ? (question mark)
  - \* (asterisk)
  - ' (single quote)
3. The following names are not allowed; CON, PRN, AUX, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, and LPT9. You should also avoid these names followed immediately by an extension (for example, NUL.txt).
4. The name must not end with a "." or consist of only "."s, for example ".", "..", "..." etc.
5. The name must not start or end with a space.

**Note:**

That Upload Project Name is more restrictive. Full details are in the Dashboard support material.

## Pathnames

On Windows based systems, which have not enabled long file support (the default behavior) pathnames are limited to 260 characters. Warnings will be given when the path exceeds 180 characters. An error will be given when the path is greater than 259 characters.

There are no length restrictions imposed on other platforms.

## D | Environment Variables

The following environment variables can be set (generally to any value) and will effect some aspect of operation. They should generally only be set when you are experiencing a problem. The performance related variables are more useful when used in a qaccli environment for fast commands. You are unlikely to notice any difference when used with the graphical tools or when doing long analysis, viewing or report runs.

HELIX\_QAC\_DISABLE\_SCHEMA\_CHECKS

Setting this to any value will disable all schema checking (principally for the ACF, RCF and `prqaproject.xml`). Any ill-formed XML file will not be detected and incorrect/incomplete data may be read. You should only use this if these exact XML files have previously been checked and are known to be OK.

HELIX\_QAC\_DISABLE\_SAMPLE\_COPY

On startup for any Helix QAC command (such as `qaccli`, `qagui` etc) the software checks and copies any modified sample projects from the installation directory into the local user data location. Setting this variable will disable this checking and copying.

HELIX\_QAC\_DISABLE\_CONFIG\_COPY

On startup for any Helix QAC command (such as `qaccli`, `qagui` etc) the software checks and copies any modified file in the 'config' folder from the installation directory into the local user data location. Setting this variable will disable this checking and copying.

HELIX\_QAC\_DISABLE\_CCT\_CHECKS

Previous version of Helix QAC did not checking for a valid CCT header. This is now checked so CCTs that previously loaded may now fail to load. Set this variable to disable the CCT checking during loading.

HELIX\_QAC\_COUT

Enabling this to one of `NONE`, `INFO`, `ERROR`, `DEBUG`, `TRACE` sends the logging information at the requested level to the console. This is the same information that is also sent to the logfile.

HELIX\_QAC\_QUIT\_QAX\_ON\_EXIT

Setting this to any value will force `qaxd` to immediately quit when the `qaccli` command completes. Setting this may increase the time for subsequent `qaccli` commands to complete.

HELIX\_QAC\_DISABLE\_VERSION\_LOGGING

Setting this to any value will stop Helix QAC logging to the logfile the exact versions of all components used during analysis.

HELIX\_QAC\_ENABLE\_BROADCAST

Setting this to any value will ensure the local subnet is searched for licence servers (if no licence server could be found at the configured addresses)

MIMALLOC\_DISABLE\_REDIRECT

The default memory allocator used by Helix QAC (on windows only) has been changed to `mimalloc` (<https://github.com/microsoft/mimalloc>). This allocator has been shown to address some rare non-deterministic failures. If analysis finishes with "Unknown Error" (todate, only seen on Windows 7 systems), then you can set the environment variable `MIMALLOC_DISABLE_REDIRECT` to 1 and retry. This disables the `DLL` injection and should resolve the issue.

This page Intentionally left blank

This page Intentionally left blank