PRQA
Programming Research

QUICK START

PRQA-FRAMEWORK 2.0.1

February, 2016

IMPORTANT NOTICE

DISCLAIMER OF WARRANTY

COPYRIGHT NOTICE

TRADEMARKS

CONTACTING PROGRAMMING RESEARCH LTD

For technical support, contact your nearest Programming Research Ltd authorized distributor or you can contact Programming Research's head office:

by telephone on	+44 (0) 1932 888 080
by fax on	+44 (0) 1932 888 081
or by webpage:	www.programmingresearch.com/services/contact-support/

Contents

List of Figures

List of Tables

1 Introduction

This document assumes that the PRQA Framework software has been successfully installed on the target machine and licensing has been correctly set up. The PRQA Framework software will request license server details the first time it starts; incorrect settings will not stop the PRQA Framework software running, but analysis of code will fail with clear indication that a license check has failed. License server details can be set in the "Admin → License Server" menu; for help setting up a license server, please contact PRQA.

Important note for users of Microsoft Visual Studio and Eclipse based IDEs; PRQA offers a plug-in for these IDEs and that plug-in should be used in preference to the PRQA Framework software. The plug-in can be installed using the Visual Studio Plug-in Package in the

```
QA-Framework-x.y.z\ide_plugins\qa_visual_studio
```

directory, in the PRQA Framework software installation directory [1]. Instructions on installing the Visual Studio plug-in is below for those users, but this documentation describes the PRQA Framework GUI; NOT the Visual Studios plug-in.

This guide focuses on the GUI aspect of the PRQA Framework software. In following this guide, the user will be guided through a simple analysis of a software project, and then introduced to using the GUI to apply the PRQA Framework software to their own software projects.

Throughout this document, text that is a typed command, filename or a directory structure is presented in a monospace font; for example,

```
prqaproject.xml.
```

The PRQA Framework software exists principally to present the user with a convenient means to trigger analysis of chosen source code files, and to examine the analysis results. The PRQA Framework program runs separately and independently to the code under examination, using existing build environment information to inform itself about the user's code.

It is intended that the user take the results from PRQA Framework and, in their existing code development environment, make changes based upon those results. Whilst PRQA Framework does have a basic text file editing capability, PRQA Framework is not intended to be used as a direct development tool to work on the code project files.

It is possible to operate the PRQA Framework in automated circumstances, such as applying analysis routinely to sets of code on a build server; such operation is beyond the scope of this guide.

[1]Where x.y.z is the version number of the QA·Framework software.

The PRQA Framework software is designed to fit around an existing code project, reading the existing code build configuration information to gather the information; that is, source files, include paths, environment setting and other such detail. As such, PRQA Framework cannot be simply "pointed at" a single source file and asked to examine it, although of course it is perfectly possible to have a project containing a single file (one of the sample projects, inspect_c, is just such a project).

The PRQA Framework software stores information about a project by adding two files (*prqaproject.xml* and *prqaproject.xml.stamp*) and a directory (*prqa*) to the project directory. These should not be altered by the user.

2 PRQA Framework Graphical User Interface - Quick Start

2.1 PRQA Framework Graphical User Interface

The PRQA Framework Graphical User Interface can be started on Windows through the start menu, and on all systems by running the

```
qagui
```

2.2 Visual Studio Plug-in

Whilst this guide discusses the GUI aspect of the PRQA Framework software, for convenience instructions on installing the Visual Studio plug-in are presented here also. The Plug-ins are delivered as part of the PRQA Framework package in the form of an installer.

2.2.1 Visual Studio 2008 Installation

- Close all instances of Visual Studio 2008 IDE.
- Click on .msi file. Follow the steps from the setup wizard.
- The product will install by default to the C:\PRQA folder.

2.2.2 Visual Studio 2010 Installation

- Close all instances of Visual Studio 2010 IDE.
- Click on .vsix file. Follow the steps from the setup wizard.
- The product will install by default to the AppData\Local\Microsoft\VisualStudio\10.0\Extensions folder.

2.2.3 Visual Studio 2012 Installation

- Close all instances of Visual Studio 2012 IDE.
- Click on .vsix file. Follow the steps from the setup wizard.
- The product will install by default to the AppData\Local\Microsoft\VisualStudio\11.0\Extensions folder.

2.2.4 Visual Studio 2013 Installation

- Close all instances of Visual Studio 2013 IDE.
- Click on .vsix file. Follow the steps from the setup wizard.
- The product will install by default to the AppData\Local\Microsoft\VisualStudio\12.0\Extensions folder.

2.2.5 After the Installation

- Open a solution with Visual Studios projects.
- A PRQA Framework menu will be visible and a QA·Visual Studios toolbar will appear also.
- To create a prqa project, click on "Create/Sync Project" option from main menu or toolbar.

2.3 Eclipse Plug-in

Please see Installation Notes for QA·Eclipse document.

3 Getting to Work - Analyzing the Example Projects

The PRQA Framework software comes with a number of sample projects; beginning by analyzing these sample projects will give the user an intuitive introduction to the PRQA Framework software.

Upon starting the software, the user is presented with the following screen (there will be minor cosmetic differences between the Windows, Linux and Solaris versions, but layout and functionality are identical).

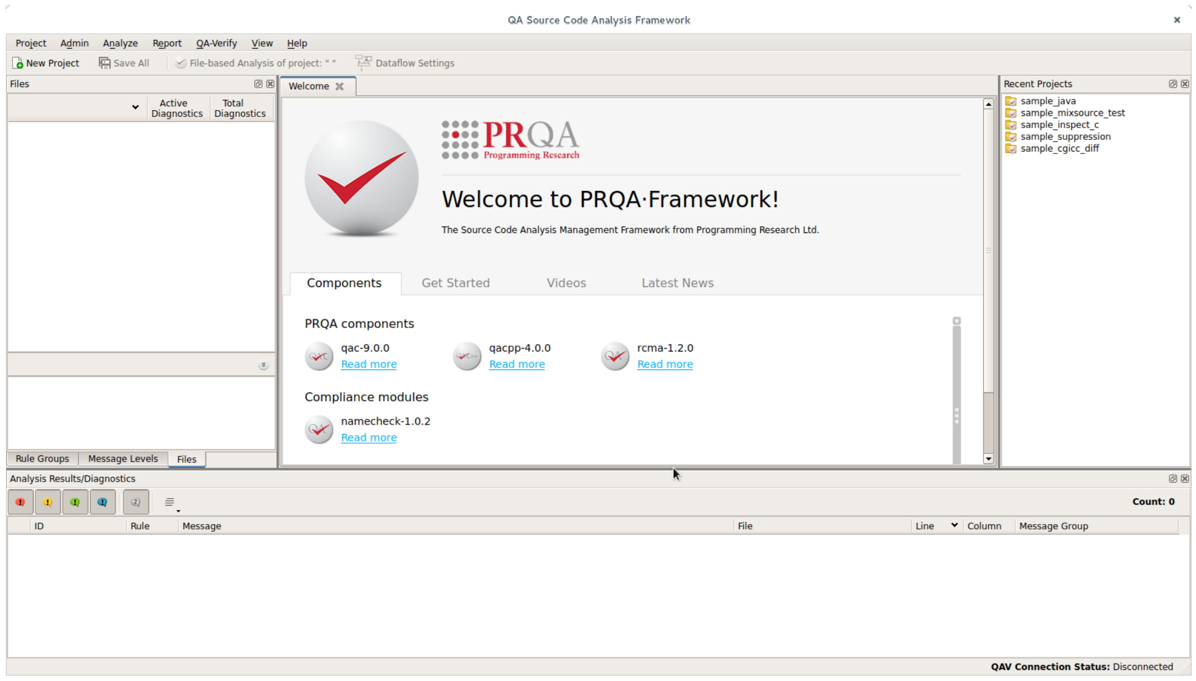
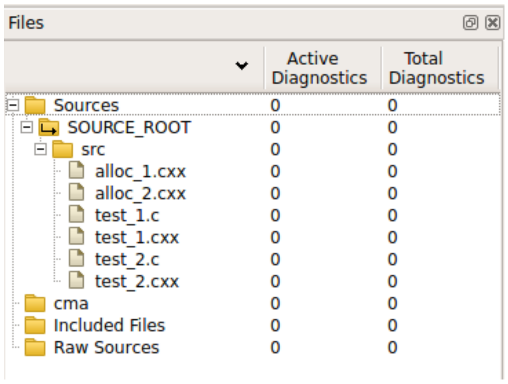


Figure 3.1: Analyzing the Example Projects

The provided example PRQA Framework projects are listed in the right-hand panel; if compliance modules, for example, a MISRA compliance module, have been installed, their sample projects will also be listed. After the first use of the PRQA Framework GUI, this list will not be updated with the sample projects, but will instead fill from the top with the user's recent projects.

A project can be selected by double-clicking on it in the "Recent Projects" panel, or by right-clicking and using the presented context menu. The display will change; the left-hand panel File browser tab will populate. Note that the default layout presents the File browser, Message Levels and Rule Groups dockable windows in a tabbed arrangement; this is because the most common workflow involves identifying which files to see the diagnostics of, and then using the Message Levels or Rule Groups tabs to see broad results. The windows are dockable and can be moved or detached entirely in the conventional mouse-dragging manner.

The user can use the File browser dockable window this to browse the project directories and see the files contained within the project:



Files

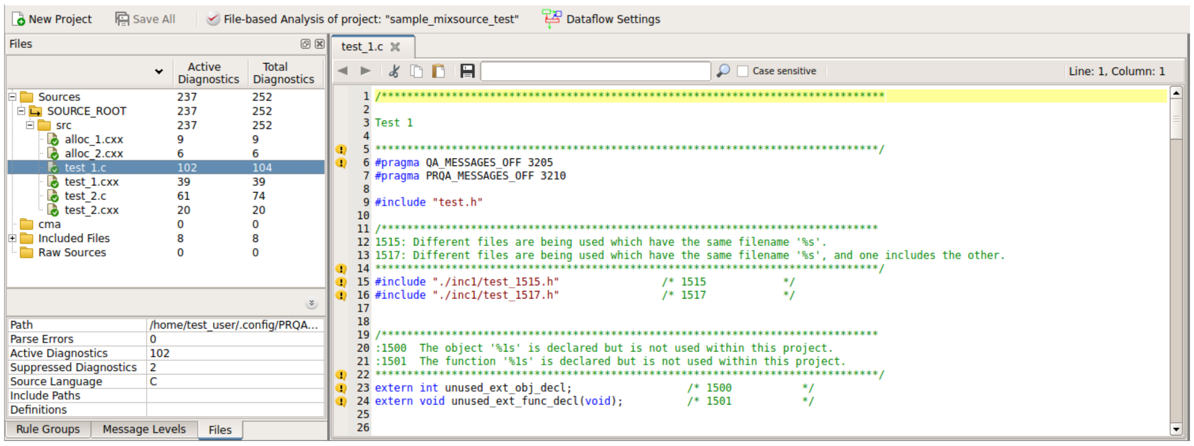
	Active Diagnostics	Total Diagnostics
Sources	0	0
SOURCE_ROOT	0	0
src	0	0
alloc_1.cxx	0	0
alloc_2.cxx	0	0
test_1.c	0	0
test_1.cxx	0	0
test_2.c	0	0
test_2.cxx	0	0
cma	0	0
Included Files	0	0
Raw Sources	0	0

Figure 3.2: Source Files

A file is selected by left-clicking the file with the mouse. Multiple files can be selected in the standard way, by holding down the CTRL key on the keyboard whilst left-clicking each file, or by holding the SHIFT key to select a range of files between two points. Also, selecting or deselecting a directory will select or deselect all files within that directory and any sub-directories. The effect of selecting a file is to have that file's analysis results presented in the "Analysis Results/Diagnostics" window at the bottom of the window; selecting more than one file will present the results from all selected files. At this stage in this example, there are no analysis results, as no analysis has been run.

When a single file has been selected, it will also be opened in the central area with any diagnostics marked. Note that if a file is open in the central area, and the user then deselects that file, its diagnostics will vanish as they are no longer part of the user's chosen set of diagnostics for viewing.

Files are best opened by right-clicking and selecting "Open" (a file can also be opened by double-clicking, *but this functionality is deprecated and should not be relied on for future versions; the first click of the double-click action will select that file, which may be an unintended consequence, so the user is advised to use right-click and "Open"*); the file will open in the central area:



	Active Diagnostics	Total Diagnostics
Sources	237	252
SOURCE_ROOT	237	252
src	237	252
alloc_1.cxx	9	9
alloc_2.cxx	6	6
test_1.c	102	104
test_1.cxx	39	39
test_2.c	61	74
test_2.cxx	20	20
cma	0	0
Included Files	8	8
Raw Sources	0	0

Path	/home/test_user/.config/PRQA...
Parse Errors	0
Active Diagnostics	102
Suppressed Diagnostics	2
Source Language	C
Include Paths	
Definitions	

```
/******************************************************************************

Test 1

******************************************************************************/
#pragma QA_MESSAGES_OFF 3205
#pragma PRQA_MESSAGES_OFF 3210

#include "test.h"

/******************************************************************************
1515: Different files are being used which have the same filename '%s'.
1517: Different files are being used which have the same filename '%s', and one includes the other.
******************************************************************************/
#include "./incl/test_1515.h"                /* 1515           */
#include "./incl/test_1517.h"                /* 1517           */


/******************************************************************************
:1500  The object '%1s' is declared but is not used within this project.
:1501  The function '%1s' is declared but is not used within this project.
******************************************************************************/
extern int unused_ext_obj_decl;                 /* 1500          */
extern void unused_ext_func_decl(void);         /* 1501          */


```

Figure 3.3: Source File Contents

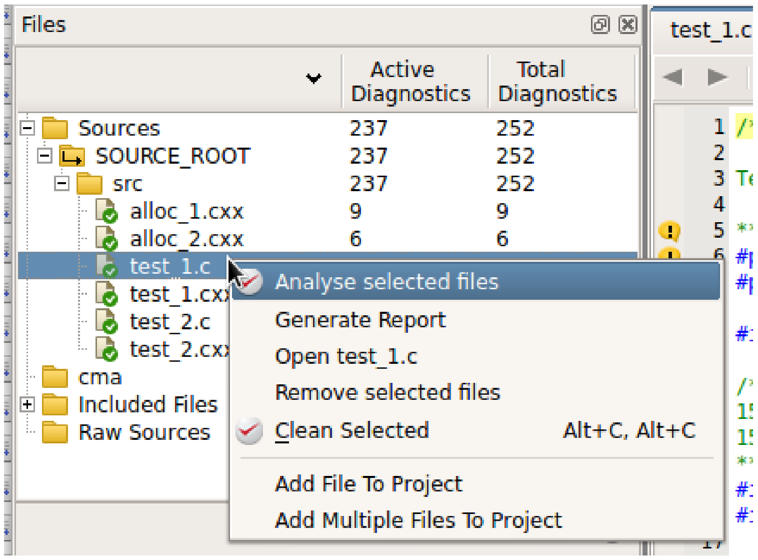
A file can also be opened for viewing by double-clicking on a relevant diagnostic, see figure 3.5.

The contents of the file can be edited and saved; note that, as stated previously, the PRQA Framework software is not intended to be a code development environment and the user should not use it as such.

The source files can be analyzed individually, or in a group. 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. Note that cma analysis is a superset of file based analysis, and requires file based analysis to have been conducted already.

This example conducts file based analysis; cma analysis can be conducted through the menus in a similar fashion to file-based analysis.

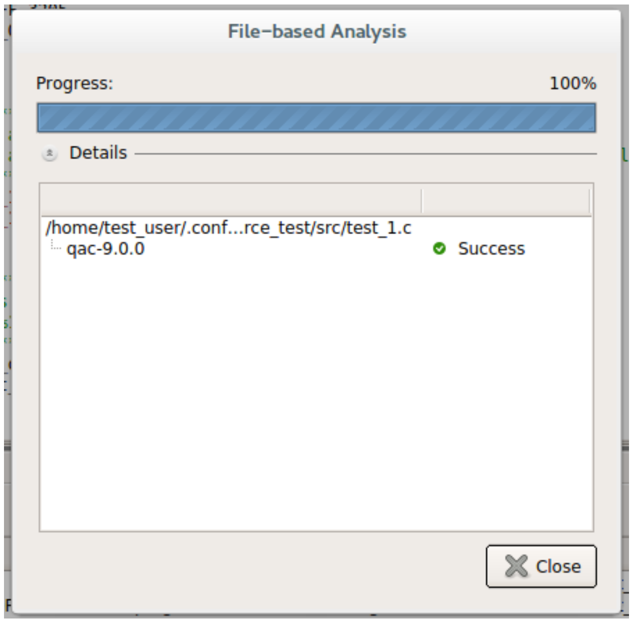
To analyze a single file, the user can select it in the Files browser window using right-click, and then selecting "Analyze":



	Active Diagnostics	Total Diagnostics
Sources	237	252
SOURCE_ROOT	237	252
src	237	252
alloc_1.cxx	9	9
alloc_2.cxx	6	6
test_1.c		
test_1.cxx		
test_2.c		
test_2.cxx		
cma		
Included Files		
Raw Sources		

Figure 3.4: Analyzing a Single File

A progress window will open, informing the user of the current state of the analysis. Upon completion of the single file analysis, a progress window will be displayed as shown:

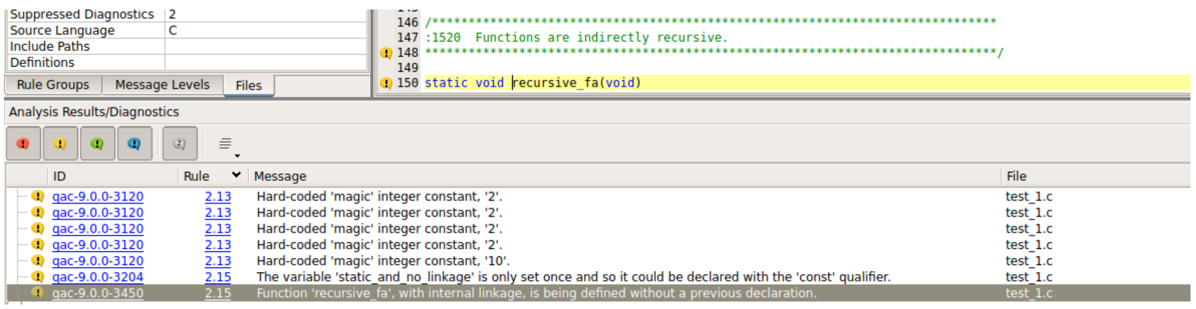


```
/home/test_user/.conf...rce_test/src/test_1.c
  qac-9.0.0                                   Success
```

Figure 3.5: Analyzing a Single File Results

Note that the "Analysis Results/Diagnostics" frame at the bottom of the screen will now be populated now populated, as the file with results is selected, and that the code displayed in the central frame will now have analysis results indicators at its left. The results of analysis for this file can now be explored. The results can be sorted by clicking column headings in the standard way. Double-clicking an individual result (in any column other than the "ID" or "Rule" column) will cause the code window to display the affected line of code, highlighted; the relevant file shall be opened if necessary.

This is shown in the following diagram; a message regarding a function has been double-clicked, and that line of code has been highlighted in yellow:



Suppressed Diagnostics	2
Source Language	C
Include Paths	
Definitions	

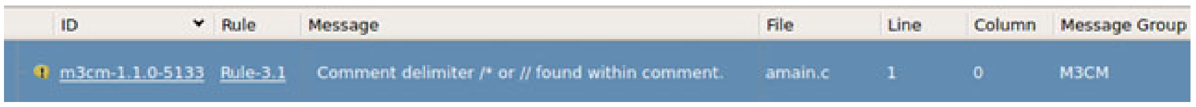
```
/*****************************************************************************
:1520  Functions are indirectly recursive.
*****************************************************************************/

static void recursive_fa(void)
```

Analysis Results/Diagnostics

ID	Rule	Message	File
qac-9.0.0-3120	2.13	Hard-coded 'magic' integer constant, '2'.	test_1.c
qac-9.0.0-3120	2.13	Hard-coded 'magic' integer constant, '2'.	test_1.c
qac-9.0.0-3120	2.13	Hard-coded 'magic' integer constant, '2'.	test_1.c
qac-9.0.0-3120	2.13	Hard-coded 'magic' integer constant, '2'.	test_1.c
qac-9.0.0-3120	2.13	Hard-coded 'magic' integer constant, '10'.	test_1.c
qac-9.0.0-3204	2.15	The variable 'static_and_no_linkage' is only set once and so it could be declared with the 'const' qualifier.	test_1.c
qac-9.0.0-3450	2.15	Function 'recursive_fa', with internal linkage, is being defined without a previous declaration.	test_1.c

Figure 3.6: Message / Code Line Association

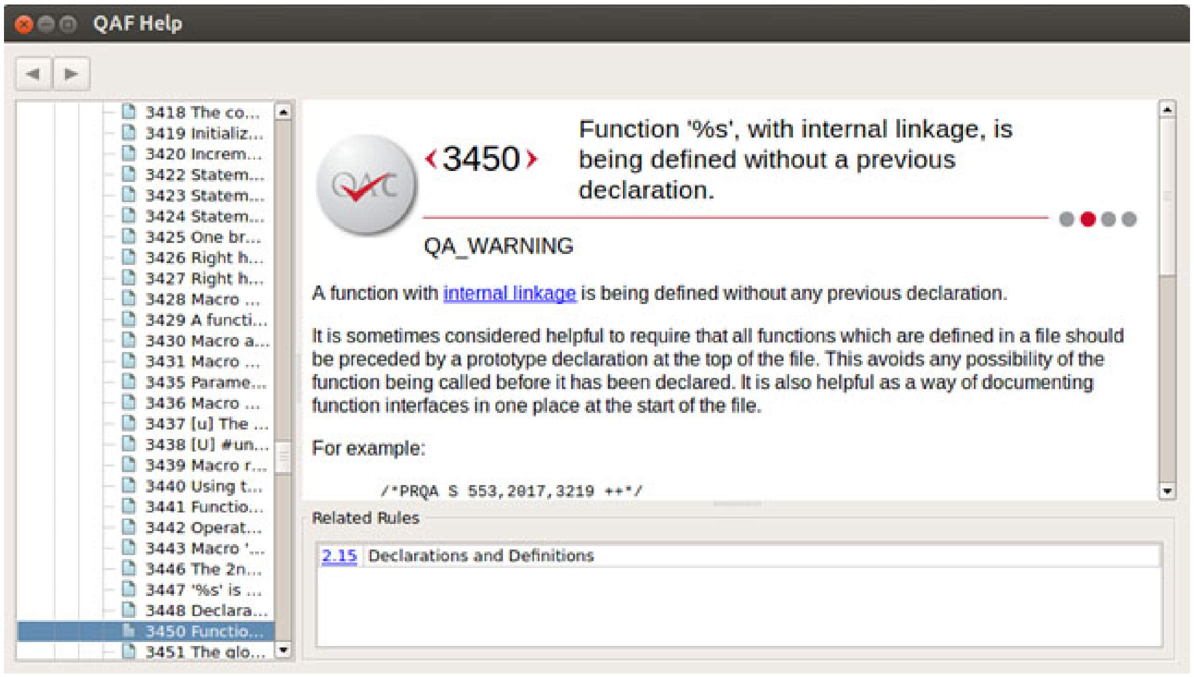
If the user is using a compliance module, the "Rule" column will directly identify which of that compliance modules rules are being violated; for example, this image shows a diagnostic from a MISRA compliance module in which Rule-3.1 of the "MISRA C:2012" rules is being violated:



ID	Rule	Message	File	Line	Column	Message Group
m3cm-1.1.0-5133	Rule-3.1	Comment delimiter /* or // found within comment.	amain.c	1	0	M3CM

Figure 3.7: Compliance Module Rule Violation

Clicking the blue highlighted "ID" column will open the PRQA Framework help in a separate window, at the relevant information, as shown:



‹3450›

Function '%s', with internal linkage, is being defined without a previous declaration.

QA_WARNING

A function with internal linkage is being defined without any previous declaration.

It is sometimes considered helpful to require that all functions which are defined in a file should be preceded by a prototype declaration at the top of the file. This avoids any possibility of the function being called before it has been declared. It is also helpful as a way of documenting function interfaces in one place at the start of the file.

For example:

```
        /*PRQA S 553,2017,3219 ++*/
```

Related Rules

2.15	Declarations and Definitions

Figure 3.8: Message Help Details

Clicking other blue highlighted "ID" or "Rule" links in the results will cause the help window to update; alternatively, depressing the SHIFT key whilst clicking will create a new help window, allowing the user to see multiple help files at the same time.

Individual files can also be selected for analysis through the menu bar, following "Analyze → File-Based Analysis → File-based Analysis of files to be specified", as shown:

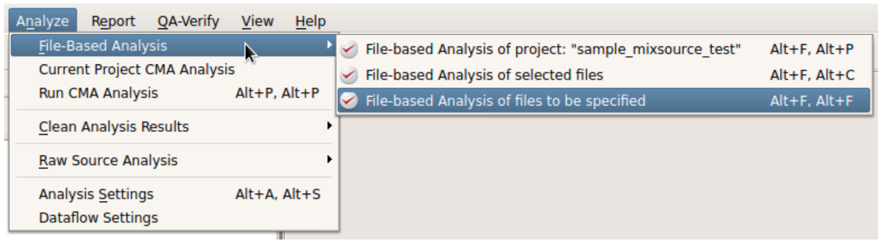

Analyze
Report
QA-Verify
View
Help
File-Based Analysis
Current Project CMA Analysis
Run CMA Analysis Alt+P, Alt+P
Clean Analysis Results
Raw Source Analysis
Analysis Settings Alt+A, Alt+S
Dataflow Settings
File-based Analysis of project: "sample_mixsource_test" Alt+F, Alt+P
File-based Analysis of selected files Alt+F, Alt+C
File-based Analysis of files to be specified Alt+F, Alt+F

Figure 3.9: Analysis of specified files

This allows the user to select any combination of files for analysis using a standard, intuitive tree-view:

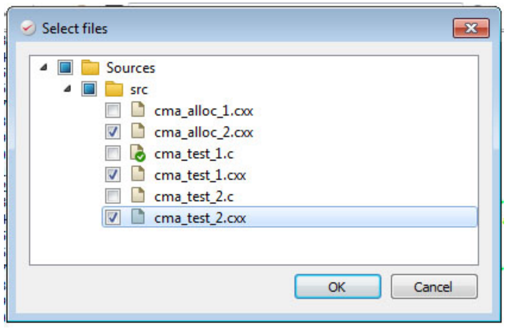
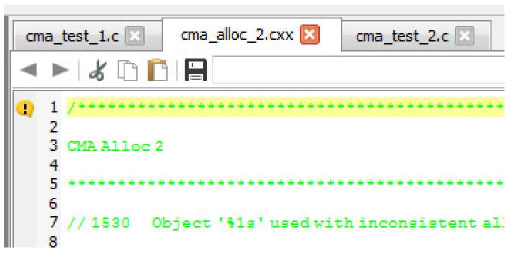


Figure 3.10: User Determined File Selection

Having chosen the files for analysis, they will be immediately analyzed as before. Now that multiple files have been analyzed, the user must select which file's results to view.

The PRQA Framework software will display the results of whichever files are currently selected in the "Files" tab at left. It is not necessary to open the file to view its results, although having a file open in the editor window whilst its results are being shown will provide icons to the left of lines containing diagnostics (open a file in an editor using the right mouse button on the file, and select from the menu). Each open file will be arranged in the central frame using a standard tabbed view:

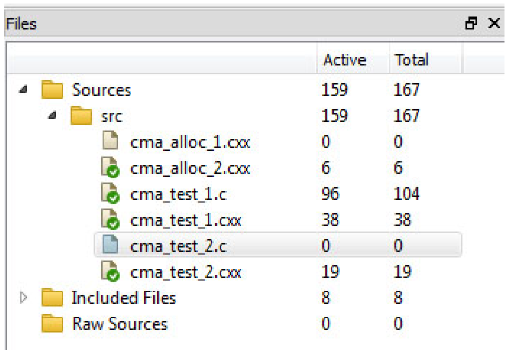


```
/*********************************************

CMA Alloc 2

**********************************************

// 1530   Object '%1s' used with inconsistent al
```

Figure 3.11: File Detailed Analysis

Each file can be brought to the fore by clicking its tab.

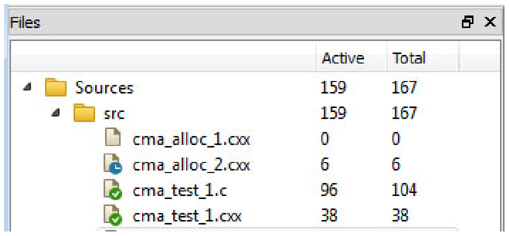
As files are analyzed, the file browser window at left will indicate the current state of analysis for any given file:



Files	Active	Total
Sources	159	167
src	159	167
cma_alloc_1.cxx	0	0
✓ cma_alloc_2.cxx	6	6
✓ cma_test_1.c	96	104
✓ cma_test_1.cxx	38	38
cma_test_2.c	0	0
✓ cma_test_2.cxx	19	19
Included Files	8	8
Raw Sources	0	0

Figure 3.12: Current State of Analysis

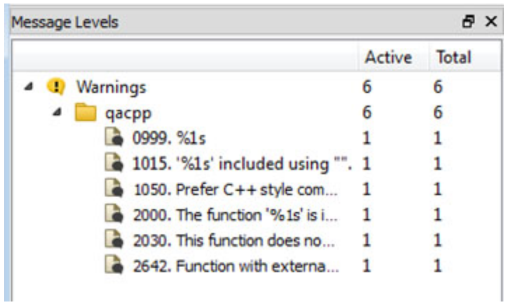
The "green tick" icon indicates that the analysis completed successfully and is available for viewing. Editing a file and/or saving it, will be recognized by the PRQA Framework software. A visual warning will be presented in the form of a "blue clock" icon, which indicates that analysis results for the file are out-of-date (this will also be done by changing the project's properties significantly, such that a new analysis needs to be conducted under the new settings). In the following diagram, the file *cma_alloc_2.cxx* has been edited and saved, and now is marked as out-of-date:



Files	Active	Total
Sources	159	167
src	159	167
cma_alloc_1.cxx	0	0
cma_alloc_2.cxx	6	6
✓ cma_test_1.c	96	104
✓ cma_test_1.cxx	38	38

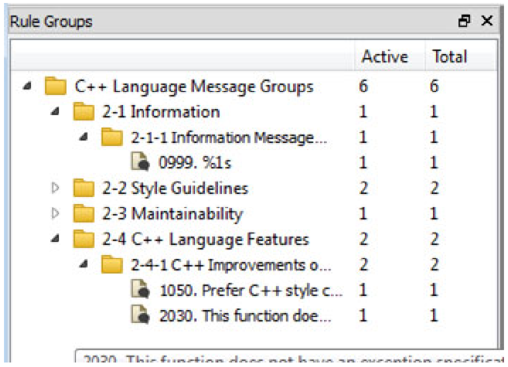
Figure 3.13: Out of Date Analysis

The left hand panel contains two other tabs, alongside the "Files" tab. "Message Levels", which provides the user with a tree-view and count, sorted by the message level, of the various different analysis results for the currently selected files, and "Rule Groups", which provides the same information sorted by rule group.


Message Levels

	Active	Total
Warnings	6	6
qacpp	6	6
0999. %1s	1	1
1015. '%1s' included using "".	1	1
1050. Prefer C++ style com...	1	1
2000. The function '%1s' is i...	1	1
2030. This function does no...	1	1
2642. Function with externa...	1	1

Figure 3.14: Message Levels


Rule Groups

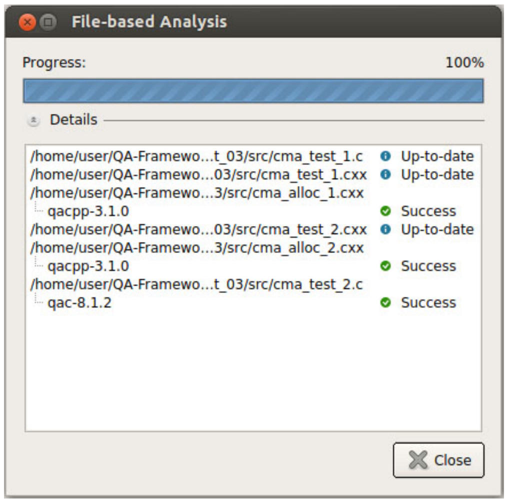
	Active	Total
C++ Language Message Groups	6	6
2-1 Information	1	1
2-1-1 Information Message...	1	1
0999. %1s	1	1
2-2 Style Guidelines	2	2
2-3 Maintainability	1	1
2-4 C++ Language Features	2	2
2-4-1 C++ Improvements o...	2	2
1050. Prefer C++ style c...	1	1
2030. This function doe...	1	1

Figure 3.15: Unique Diagnostic Counts

Each level of the Rule Groups 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 Rule Groups tab is intended to show users which diagnostics violate which rules, but ultimately counts unique diagnostics only; the number of rules violated may be greater than the number of diagnostics if a diagnostic violates more than one rule.

Right-clicking on a message or rule gives the user the option to select a single message or all messages for a given rule group or warning level. This will then be applied to the diagnostics shown in the diagnostics window. By this means, a user could choose, for example, to show all instances of a single message for the entire project, or all messages from a single rule group for a given subset of files.

The user can also choose to simply analyze all files in the project at once, through the "Analyze → File-Based Analysis → File-based Analysis of Project <Project Name>". This will trigger an analysis of every file in the project that is not up-to-date. The progress window indicates which files specifically have been analyzed:



```
/home/user/QA-Framewo...t_03/src/cma_test_1.c       Up-to-date
/home/user/QA-Framewo...03/src/cma_test_1.cxx       Up-to-date
/home/user/QA-Framewo...3/src/cma_alloc_1.cxx
  qacpp-3.1.0                                       Success
/home/user/QA-Framewo...03/src/cma_test_2.cxx       Up-to-date
/home/user/QA-Framewo...3/src/cma_alloc_2.cxx
  qacpp-3.1.0                                       Success
/home/user/QA-Framewo...t_03/src/cma_test_2.c
  qac-8.1.2                                         Success
```

Figure 3.16: File-Based Analysis of Project Progress Window

Results are available in the same way as previously discussed.

4 Creating Your Own Projects - C&C++

As alluded to previously, the PRQA Framework software is designed to fit around an existing code project, reading the existing code build configuration information to gather the information it needs; that is, source files, include paths, environment setting and other such detail.

The PRQA Framework software gathers the information it needs by executing a build of the user's code project. The PRQA Framework software requires a single build command, that can be run from anywhere on the system, that will build the user's code project. This functionality exists only for C&C++ projects. Java and C# projects must have their files indicated manually by the user; see section 5.

This is best explained through simple examples. Examples are presented for Linux and Windows. Much of the discussion is applicable to both, so even if the user intends to work solely on one system, it will be informative to read all the examples.

4.1 Use Case - Analyze a simple code project built using an executable script on a Linux PC

A user has simple code project, consisting of a few source code files, arranged in a directory structure as shown, on a standard developer Linux installation with the GCC compiler suite and associated standard development tools.

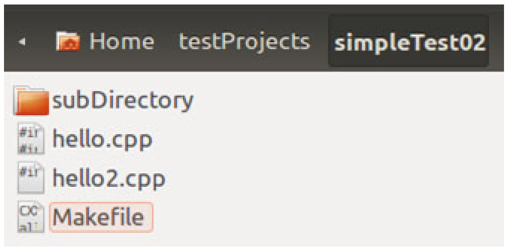


Figure 4.1: Analyze a Simple Code Project

The user currently builds this code project by running the command *make* on the command line, where the *Makefile* contents are as shown:

```
CXX = g++
all:
    $(CXX) -o outputFile hello.cpp hello2.cpp
    ./subDirectory/code.cpp
clean:
```

This simple situation is easily turned into a PRQA Framework project. When a user wishes to apply the PRQA Framework software to a code project, the procedure is conducted in two steps; creating a new PRQA Framework project, and then informing the PRQA Framework software of the files to be analyzed (this is done by building the code project as the PRQA Framework software observes). This means that creating a PRQA Framework project presents the user with an *empty* project, which must then be populated with the relevant files.

4.1.1 Creating a New PRQA Framework Project

Select "New Project":

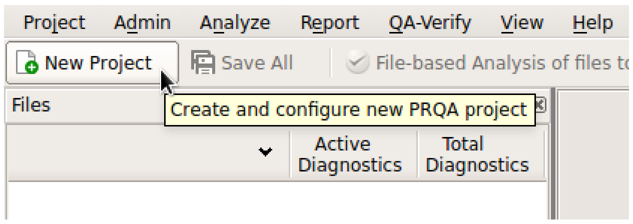
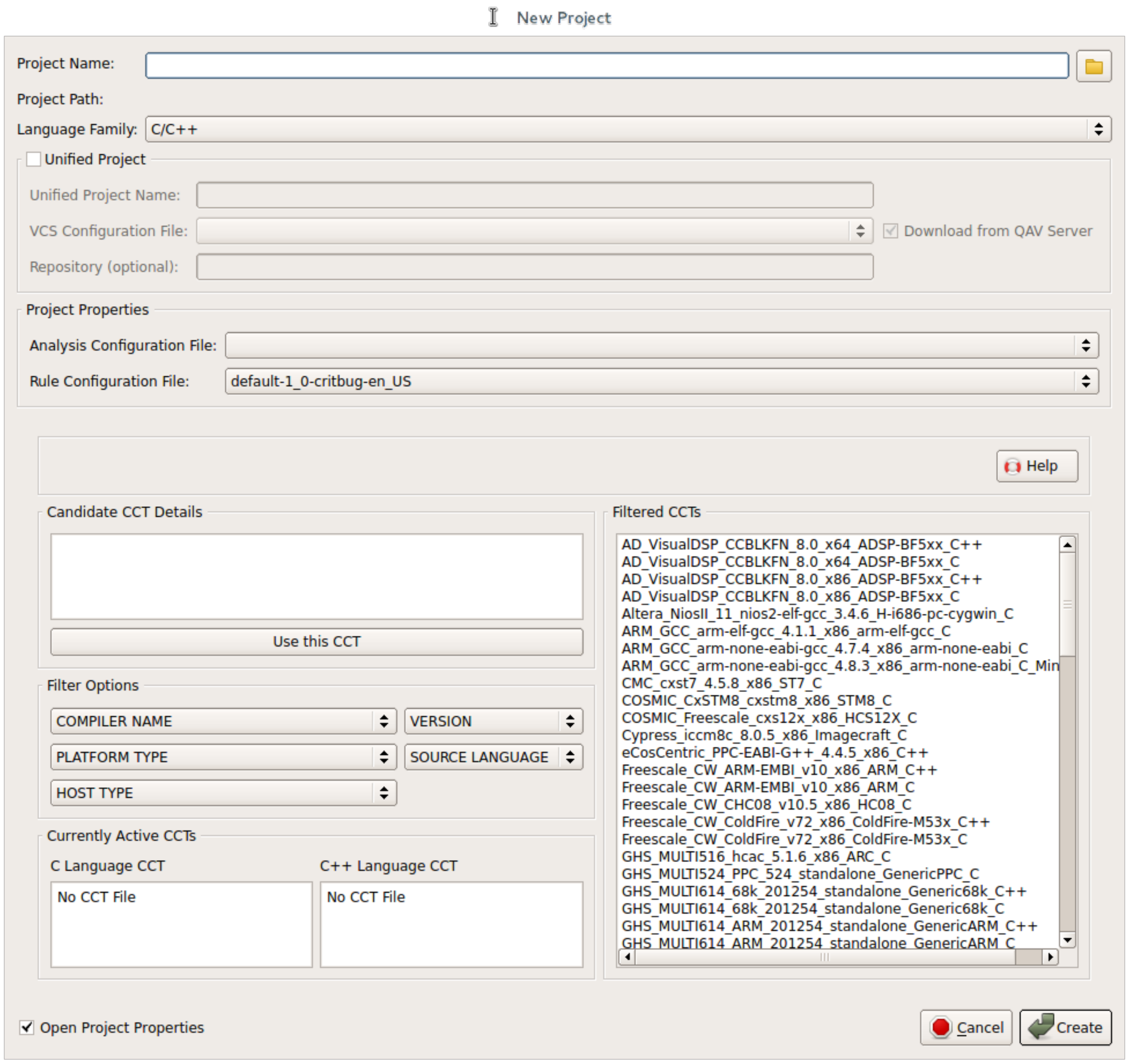


Figure 4.2: Create a New PRQA Framework Project

Select the directory containing the code project of interest, and also the appropriate compiler from the list:



New Project

Project Name:

Project Path:

Language Family: C/C++

☐ Unified Project

Unified Project Name:

VCS Configuration File:

☑ Download from QAV Server

Repository (optional):

Project Properties

Analysis Configuration File:

Rule Configuration File: default-1_0-critbug-en_US

Help

Candidate CCT Details

Use this CCT

Filtered CCTs

```
AD_VisualDSP_CCBLKFN_8.0_x64_ADSP-BF5xx_C++
AD_VisualDSP_CCBLKFN_8.0_x64_ADSP-BF5xx_C
AD_VisualDSP_CCBLKFN_8.0_x86_ADSP-BF5xx_C++
AD_VisualDSP_CCBLKFN_8.0_x86_ADSP-BF5xx_C
Altera_NiosII_11_nios2-elf-gcc_3.4.6_H-i686-pc-cygwin_C
ARM_GCC_arm-elf-gcc_4.1.1_x86_arm-elf-gcc_C
ARM_GCC_arm-none-eabi-gcc_4.7.4_x86_arm-none-eabi_C
ARM_GCC_arm-none-eabi-gcc_4.8.3_x86_arm-none-eabi_C_Min
CMC_cxst7_4.5.8_x86_ST7_C
COSMIC_CxSTM8_cxstm8_x86_STM8_C
COSMIC_Freescale_cxs12x_x86_HCS12X_C
Cypress_iccm8c_8.0.5_x86_Imagecraft_C
eCosCentric_PPC-EABI-G++_4.4.5_x86_C++
Freescale_CW_ARM-EMBI_v10_x86_ARM_C++
Freescale_CW_ARM-EMBI_v10_x86_ARM_C
Freescale_CW_CHC08_v10.5_x86_HC08_C
Freescale_CW_ColdFire_v72_x86_ColdFire-M53x_C++
Freescale_CW_ColdFire_v72_x86_ColdFire-M53x_C
GHS_MULTI516_hcac_5.1.6_x86_ARC_C
GHS_MULTI524_PPC_524_standalone_GenericPPC_C
GHS_MULTI614_68k_201254_standalone_Generic68k_C++
GHS_MULTI614_68k_201254_standalone_Generic68k_C
GHS_MULTI614_ARM_201254_standalone_GenericARM_C++
GHS_MULTI614_ARM_201254_standalone_GenericARM_C
```

Filter Options

COMPILER NAME | VERSION
PLATFORM TYPE | SOURCE LANGUAGE
HOST TYPE

Currently Active CCTs

C Language CCT: No CCT File

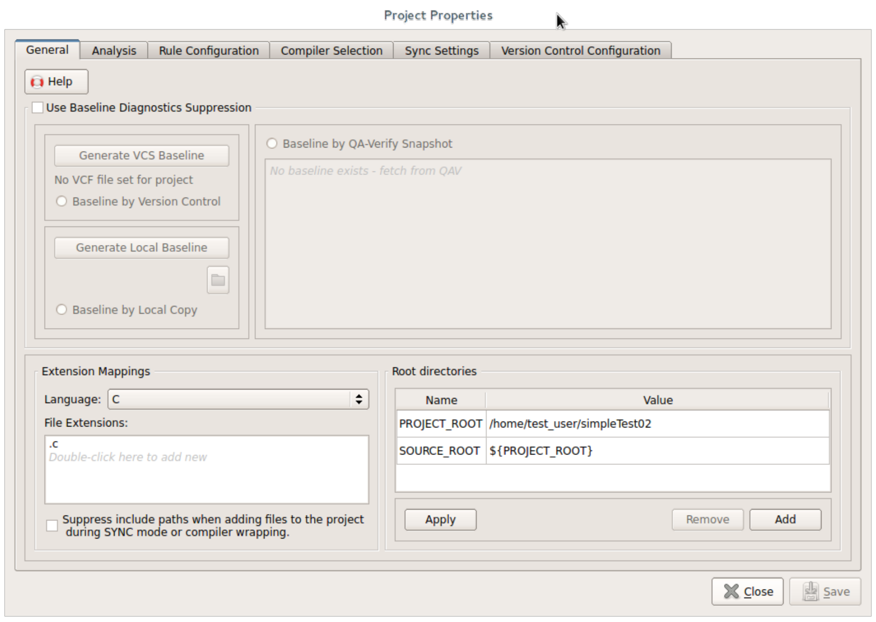
C++ Language CCT: No CCT File

☑ Open Project Properties

Cancel | Create

Figure 4.3: New PRQA Framework Project Window

Selecting "Create" will cause the Project Properties window to open:


Project Properties

General | Analysis | Rule Configuration | Compiler Selection | Sync Settings | Version Control Configuration

Help

☐ Use Baseline Diagnostics Suppression

Generate VCS Baseline

No VCF file set for project

○ Baseline by Version Control

Generate Local Baseline

○ Baseline by Local Copy

○ Baseline by QA-Verify Snapshot

No baseline exists - fetch from QAV

Extension Mappings

Language: C

File Extensions:

.c

Double-click here to add new

☐ Suppress include paths when adding files to the project during SYNC mode or compiler wrapping.

Root directories

Name	Value
PROJECT_ROOT	/home/test_user/simpleTest02
SOURCE_ROOT	${PROJECT_ROOT}

Apply | Remove | Add

Close | Save

Figure 4.4: New PRQA Framework Project Properties Window

For this simple example, this is all that is needed. Other options can be safely ignored at this stage and the Project properties window can be closed without making any changes.

The project should now be open; the "File" window will show zero files. This has created a new PRQA Framework project; the next step is to populate the PRQA Framework project with files for analysis.

4.1.2 Informing the PRQA Framework Software of the Files to be Analyzed - C&C++ Projects

The PRQA Framework software is intended to infer the source code contents of a code project by observing a build process. This functionality can be accessed via the "Project $\rightarrow$ Synchronize" menu.

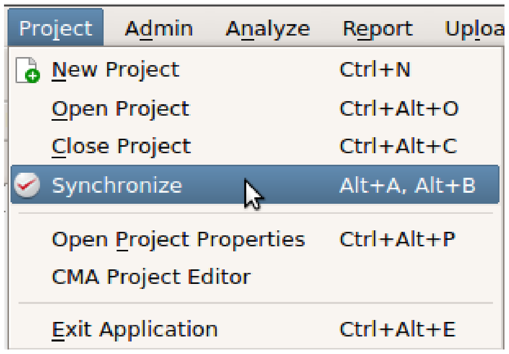
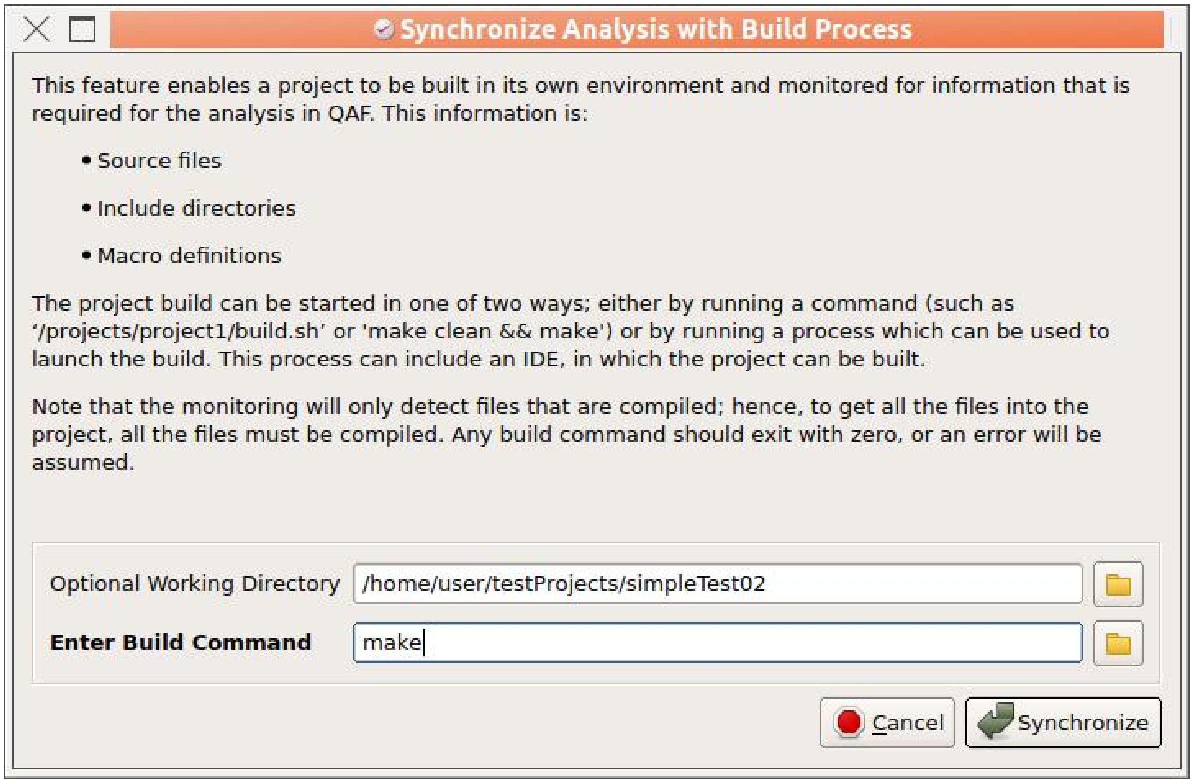


Figure 4.5: Synchronize QA·F project with source

This image below shows how one would enter the build script name and location into the PRQA Framework software:


Synchronize Analysis with Build Process

This feature enables a project to be built in its own environment and monitored for information that is required for the analysis in QAF. This information is:

- Source files
- Include directories
- Macro definitions

The project build can be started in one of two ways; either by running a command (such as '/projects/project1/build.sh' or 'make clean && make') or by running a process which can be used to launch the build. This process can include an IDE, in which the project can be built.

Note that the monitoring will only detect files that are compiled; hence, to get all the files into the project, all the files must be compiled. Any build command should exit with zero, or an error will be assumed.

Optional Working Directory: /home/user/testProjects/simpleTest02

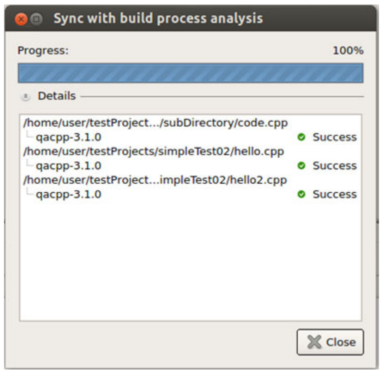
Enter Build Command: make

Figure 4.6: Build Script Name and Location

Pressing "Synchronize" will cause the PRQA Framework software to execute the script and monitor the subsequent build process, identifying the source code files of interest.

Note that because the PRQA Framework software informs itself by watching the build process, if the build process does not build something (for example, because the code project has been recently built and nothing needs to be recompiled), the PRQA Framework software will gather no information. Accordingly, it is recommended to clean the build first, or (as inferred in the image above) to have the scripted build process conducts such a cleaning operation).

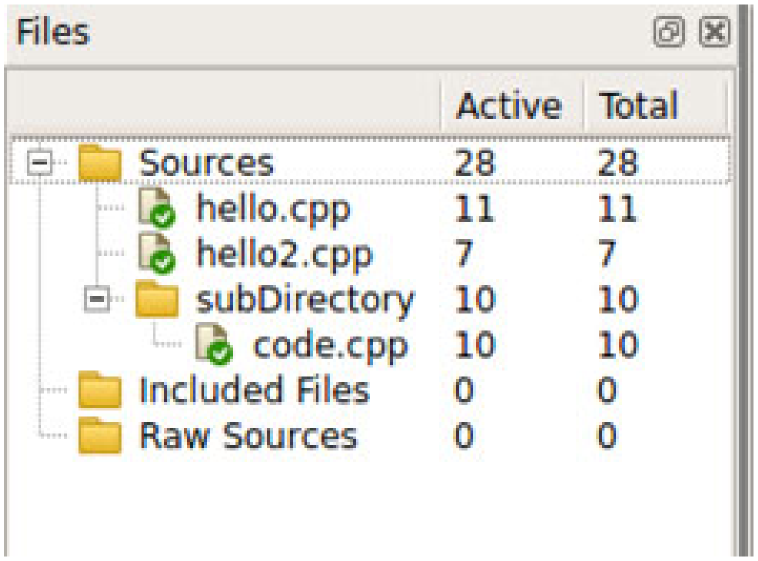
The progress window shows files being identified and analyzed as the build process is observed.



```
/home/user/testProject.../subDirectory/code.cpp
  qacpp-3.1.0                                     Success
/home/user/testProjects/simpleTest02/hello.cpp
  qacpp-3.1.0                                     Success
/home/user/testProject...impleTest02/hello2.cpp
  qacpp-3.1.0                                     Success
```

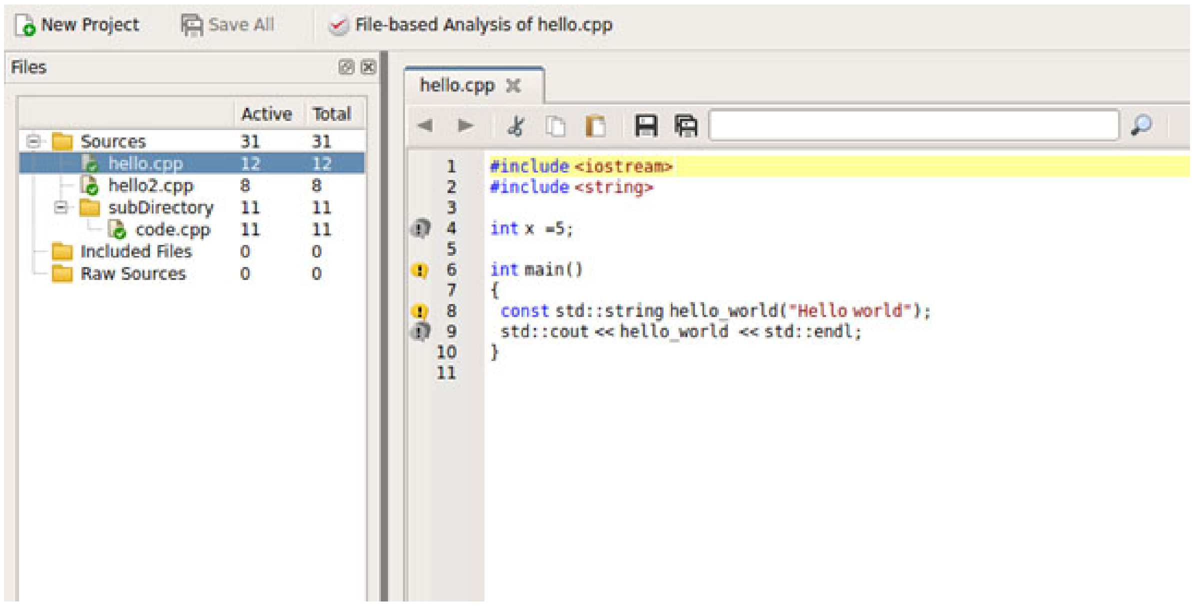
Figure 4.7: Build Process Progress Window

The identified files are available for examination and selection as before:



	Active	Total
Sources	28	28
hello.cpp	11	11
hello2.cpp	7	7
subDirectory	10	10
code.cpp	10	10
Included Files	0	0
Raw Sources	0	0

Figure 4.8: Built Source File Selection



	Active	Total
Sources	31	31
hello.cpp	12	12
hello2.cpp	8	8
subDirectory	11	11
code.cpp	11	11
Included Files	0	0
Raw Sources	0	0

```
#include <iostream>
#include <string>

int x =5;

int main()
{
 const std::string hello_world("Hello world");
 std::cout << hello_world << std::endl;
}
```

Figure 4.9: Built Source File Selection Details

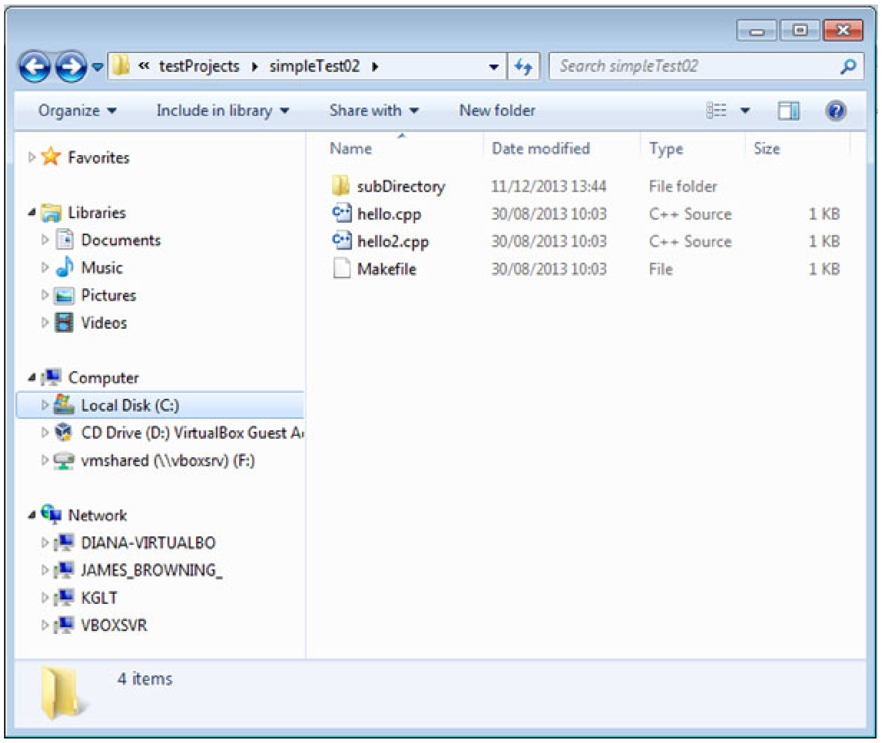
Larger, more complex code projects can be analyzed by the PRQA Framework software in the same way.

4.2 Use Case - Analyze a Code Project Built Using an Executable Script on Windows

The typical windows software development environment is an IDE; most common are Microsoft's Visual Studio or Eclipse-based IDEs. Plug-ins are available for these IDEs and a user should use them rather than the PRQA Framework GUI.

This example assumes that a user is not using one of these IDEs but instead is using a build method analogous to the previous example; the user has chosen to use the command-line based development system provided by the Cygwin tools.

The user has a simple code project with a structure shown as in the image:



Name	Date modified	Type	Size
subDirectory	11/12/2013 13:44	File folder	
hello.cpp	30/08/2013 10:03	C++ Source	1 KB
hello2.cpp	30/08/2013 10:03	C++ Source	1 KB
Makefile	30/08/2013 10:03	File	1 KB

Figure 4.10: Project Built Using an Executable Script

Using their command-line build tools, the user currently builds this project through typing the command:

```
g++ -o outputFile hello.cpp hello2.cpp ./subDirectory/code.cpp
```

The g++ (i.e. the GCC C++) compiler is part of the installed Cygwin tools the user has chosen to work with in this example.

Alternatively, the user could have a simple *Makefile*, which they run through the command *make*.

```
CXX = g++
all:
  $(CXX) -o outputFile hello.cpp hello2.cpp \
                       ./subDirectory/code.cpp
clean:
```

The code project will not need to be altered in any way; the PRQA Framework software can analyze it without any changes as long as the user can provide a single command to be run that will work from any directory, as in the previous Linux example.

In Windows systems, it is common to use a *.bat (windows batch) file for simple scripting; a single executable script that will run the code build is needed by the QA GUI; in this example, the user has created a file build.bat, which they can execute to conduct the build.

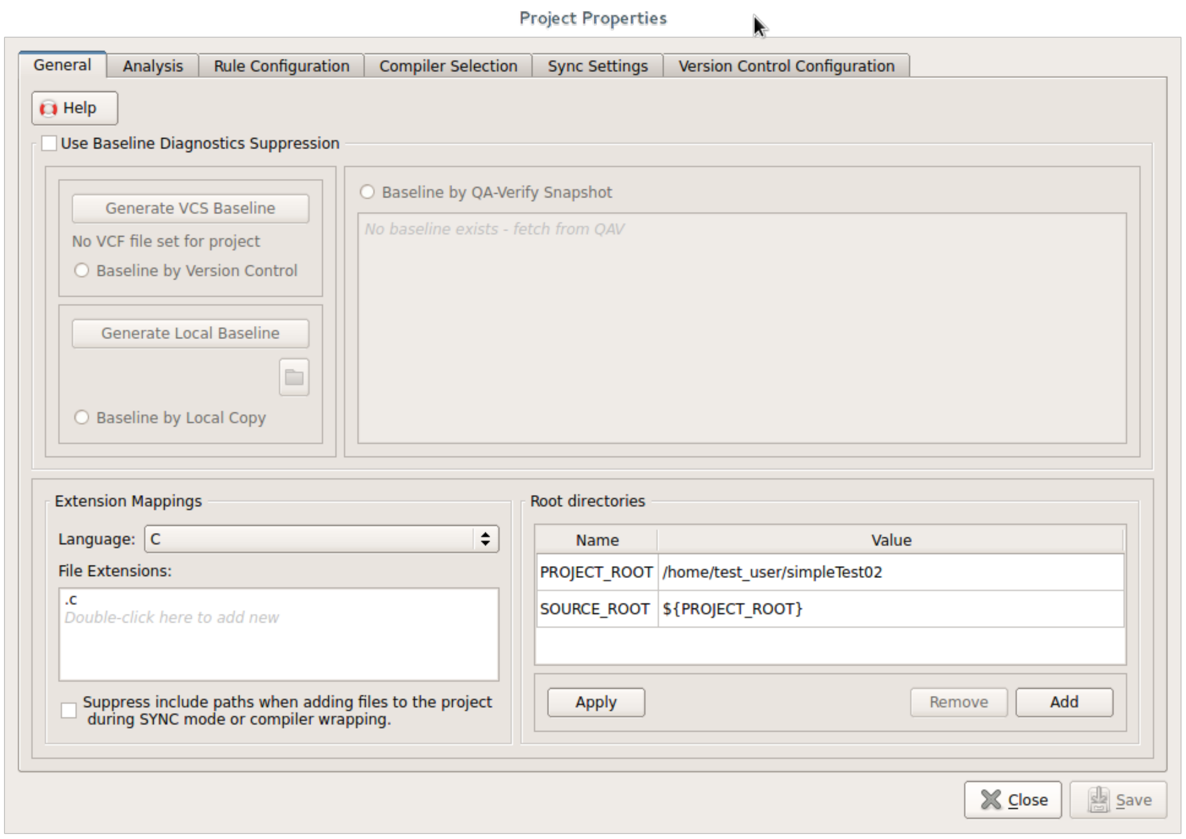
```
    make -C C:\Users\Public\testProjects\simpleTest02
```

Given this existing code project, the user now wishes to analyze the code using the PRQA Framework GUI. Upon starting the PRQA Framework GUI, the user is presented with the standard start page. The user must now create a new project; this can be done using the "New Project" button towards the top-left of the PRQA Framework GUI, just as in the previous example.

The user is now presented with a dialog box; the user specifies the "Project Name". The project name is simply the name of the directory containing the code project; in this example,

```
    simpleTest02
```

The user must also specify the compiler being used; a large number of options are presented and it is important the user select correctly. In this case, the user would chose the appropriate Cygwin compiler. This is all that is needed for this simple project; the user presses "Create" and is presented with the "Project Properties" dialog:



Project Properties

General | Analysis | Rule Configuration | Compiler Selection | Sync Settings | Version Control Configuration

Help

Use Baseline Diagnostics Suppression

Generate VCS Baseline

No VCF file set for project

Baseline by Version Control

Generate Local Baseline

Baseline by Local Copy

Baseline by QA-Verify Snapshot

No baseline exists - fetch from QAV

Extension Mappings

Language: C

File Extensions:

.c

Double-click here to add new

Suppress include paths when adding files to the project during SYNC mode or compiler wrapping.

Root directories

Name	Value
PROJECT_ROOT	/home/test_user/simpleTest02
SOURCE_ROOT	${PROJECT_ROOT}

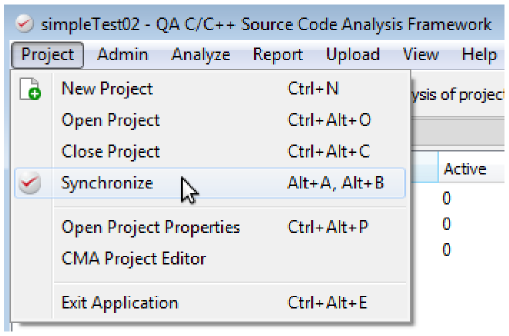
Apply | Remove | Add

Close | Save

Figure 4.11: Create a New Project Properties

This simple case requires no change from the defaults; the user simply closes this dialog.

The PRQA Framework software will now monitor a build process to gather the information it needs to identify the source files and other configuration information it requires. The user must identify to the PRQA Framework GUI the single script to run to conduct the software build; as discussed previously, the user has created a Windows batch file to do this. The user selects "Project → Synchronize":



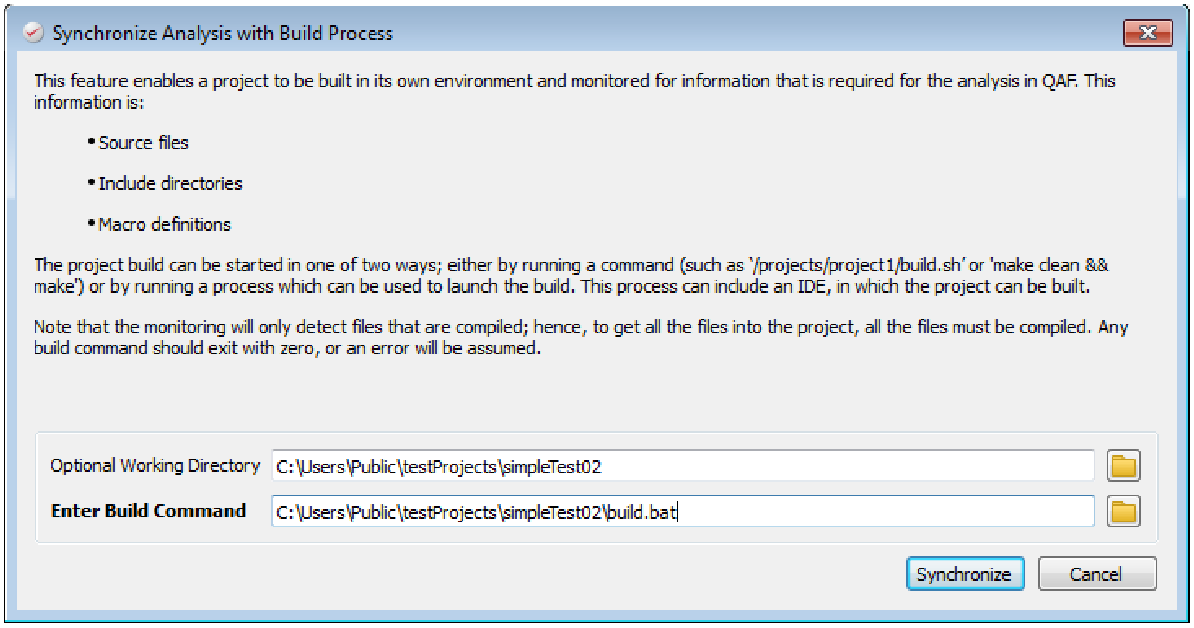
simpleTest02 - QA C/C++ Source Code Analysis Framework

Project | Admin | Analyze | Report | Upload | View | Help

New Project	Ctrl+N
Open Project	Ctrl+Alt+O
Close Project	Ctrl+Alt+C
Synchronize	Alt+A, Alt+B
Open Project Properties	Ctrl+Alt+P
CMA Project Editor	
Exit Application	Ctrl+Alt+E

Figure 4.12: File-Based Analysis of Build Process

and is presented with the opportunity to identify the batch file:


Synchronize Analysis with Build Process

This feature enables a project to be built in its own environment and monitored for information that is required for the analysis in QAF. This information is:

- Source files
- Include directories
- Macro definitions

The project build can be started in one of two ways; either by running a command (such as '/projects/project1/build.sh' or 'make clean && make') or by running a process which can be used to launch the build. This process can include an IDE, in which the project can be built.

Note that the monitoring will only detect files that are compiled; hence, to get all the files into the project, all the files must be compiled. Any build command should exit with zero, or an error will be assumed.

Optional Working Directory: C:\Users\Public\testProjects\simpleTest02

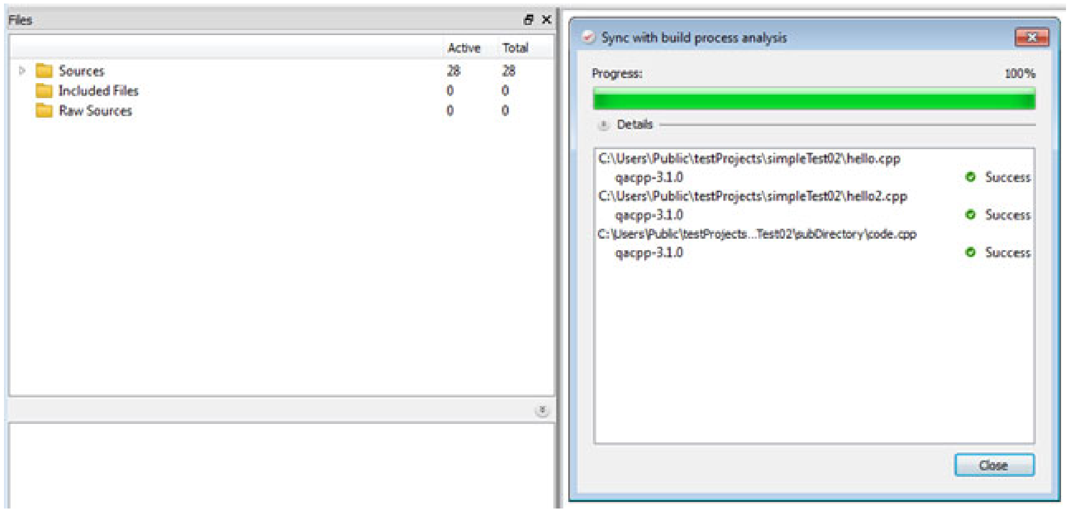
Enter Build Command: C:\Users\Public\testProjects\simpleTest02\build.bat

Figure 4.13: File-Based Analysis of Build Process Batch File

Alternatively, the user might choose to discard the batch file and simply enter the complete command within that batch file, if the command is very simple; for example, as in this simple example case,

```
make -C C:\Users\Public\testProjects\simpleTest02
```

When the user presses "Synchronize", the build begins with the PRQA Framework software monitoring it. The user will watch the progression, until the screen appears as follows; note that the number of diagnostics listed in the left window has changed, and that the feedback indicates successful identification and scan of the code project files.

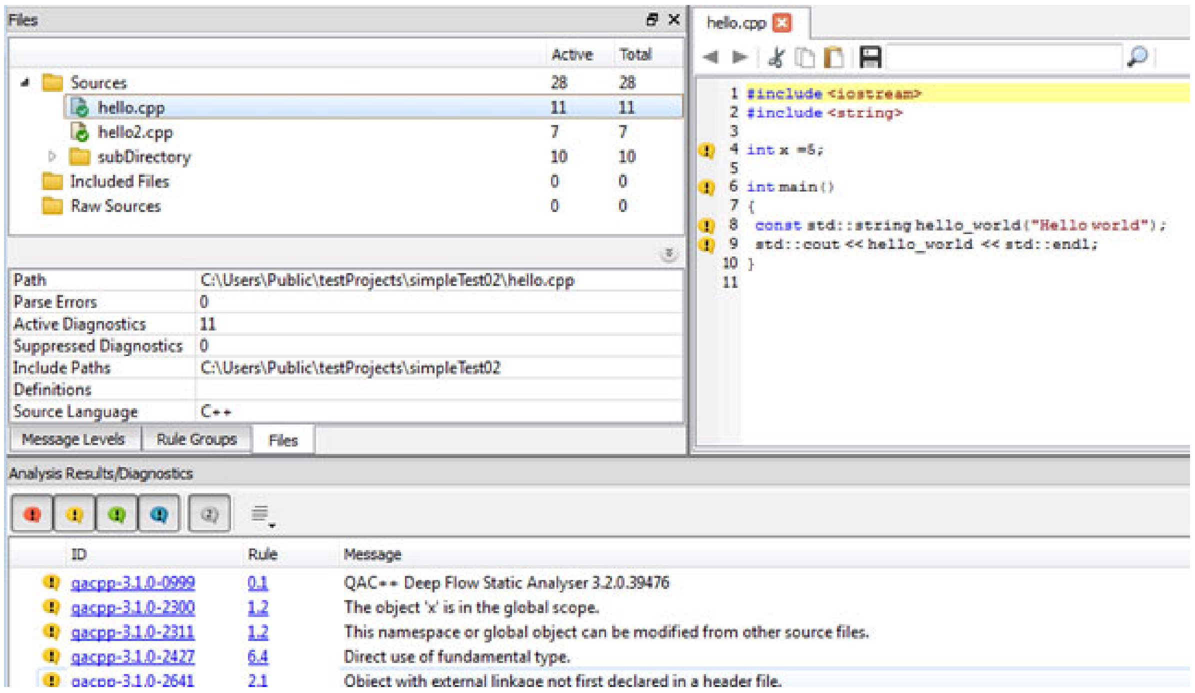


	Active	Total
Sources	28	28
Included Files	0	0
Raw Sources	0	0

```
C:\Users\Public\testProjects\simpleTest02\hello.cpp
    qacpp-3.1.0                                              Success
C:\Users\Public\testProjects\simpleTest02\hello2.cpp
    qacpp-3.1.0                                              Success
C:\Users\Public\testProjects...Test02\subDirectory\code.cpp
    qacpp-3.1.0                                              Success
```

Figure 4.14: Build Process Analysis

The results are available for examination as described previously (in the exploration of the sample project):



	Active	Total
Sources	28	28
hello.cpp	11	11
hello2.cpp	7	7
subDirectory	10	10
Included Files	0	0
Raw Sources	0	0

Path	C:\Users\Public\testProjects\simpleTest02\hello.cpp
Parse Errors	0
Active Diagnostics	11
Suppressed Diagnostics	0
Include Paths	C:\Users\Public\testProjects\simpleTest02
Definitions	
Source Language	C++

```
#include <iostream>
#include <string>

int x =5;

int main()
{
 const std::string hello_world("Hello world");
 std::cout << hello_world << std::endl;
}
```

ID	Rule	Message
qacpp-3.1.0-0999	0.1	QAC++ Deep Flow Static Analyser 3.2.0.39476
qacpp-3.1.0-2300	1.2	The object 'x' is in the global scope.
qacpp-3.1.0-2311	1.2	This namespace or global object can be modified from other source files.
qacpp-3.1.0-2427	6.4	Direct use of fundamental type.
qacpp-3.1.0-2641	2.1	Object with external linkage not first declared in a header file.

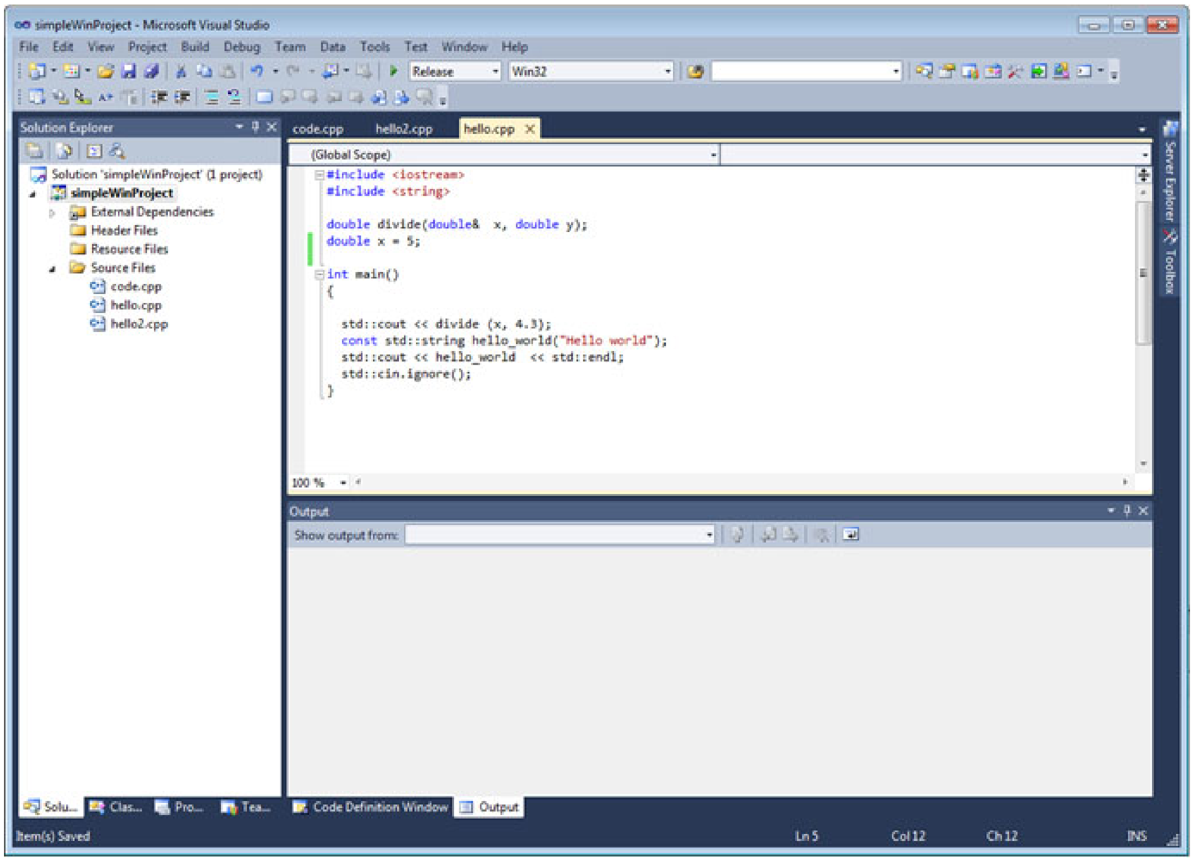
Figure 4.15: Build Source File Selection Details

4.3 Use Case - Analyze a Microsoft Visual Studio Project Without Using the Plug-in

It is intended that users of Microsoft's Visual Studio use the plug-in module to analyze their code. However, should a user choose not to use the plug-in, the PRQA Framework GUI can still be used to analyze their code.

In this example, the user has a directory layout and source code files identical to the previous example, but they have built this code into a Microsoft Visual Studio project, named

```
simpleWinProject
```



```
#include <iostream>
#include <string>

double divide(double&  x, double y);
double x = 5;

int main()
{

  std::cout << divide (x, 4.3);
  const std::string hello_world("Hello world");
  std::cout << hello_world  << std::endl;
  std::cin.ignore();
}
```

Figure 4.16: Analyze a Microsoft Visual Studio Project Without Using the Plug-in

The user currently builds their project using the Microsoft Visual Studio GUI, but wishes to have their code analyzed, without using the available Microsoft Visual Studio plug-in. The steps are almost identical to the previous example.

After starting the PRQA Framework GUI, the user must now create a new PRQA Framework Project, using the "New Project" button:

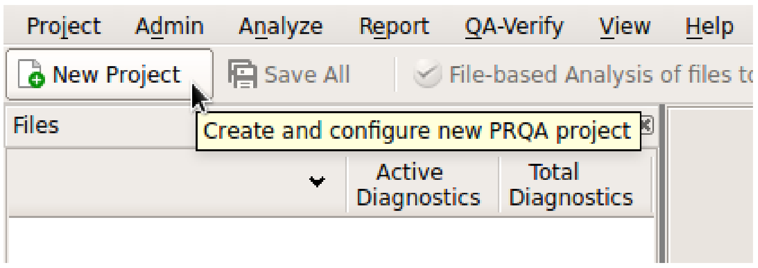
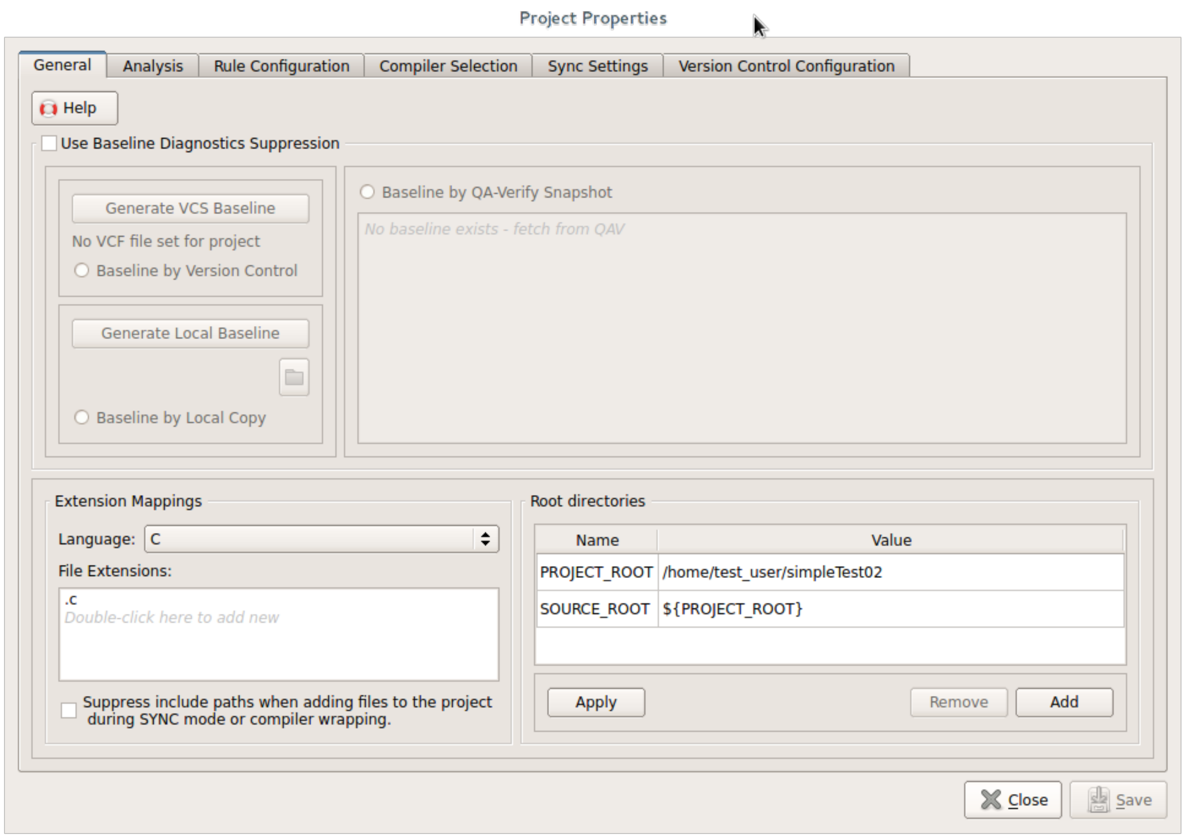


Figure 4.17: Create a New PRQA Framework Project

The user is now presented with a dialog box; the user specifies the Project Name. The project name is simply the name of the directory containing the code project; in this example,

```
simpleWinProject
```

The user also chooses the appropriate compiler, which in this case is the Microsoft Visual Studio 2010 compiler. The user is now presented with project options; in this simple case, the default values are suitable and the user need do no more than close this window:


Project Properties

General | Analysis | Rule Configuration | Compiler Selection | Sync Settings | Version Control Configuration

Help

Use Baseline Diagnostics Suppression

Generate VCS Baseline

No VCF file set for project

Baseline by Version Control

Generate Local Baseline

Baseline by Local Copy

Baseline by QA-Verify Snapshot

No baseline exists - fetch from QAV

Extension Mappings

Language: C

File Extensions:

.c
Double-click here to add new

Suppress include paths when adding files to the project during SYNC mode or compiler wrapping.

Root directories

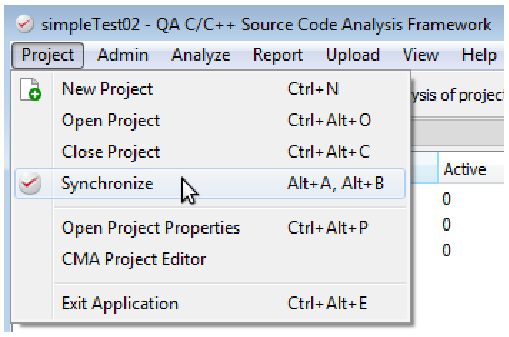
Name	Value
PROJECT_ROOT	/home/test_user/simpleTest02
SOURCE_ROOT	${PROJECT_ROOT}

Apply | Remove | Add

Close | Save

Figure 4.18: New PRQA Framework Project Properties Window

Now, the QAF software must observe a build to infer the correct source files for analysis, as in previous examples. To do so, it must be given a command to run on the command line that will drive a build. In Microsoft's Visual Studio, the build process can be driven using the devenv.exe program. The user selects "Project → Synchronize":

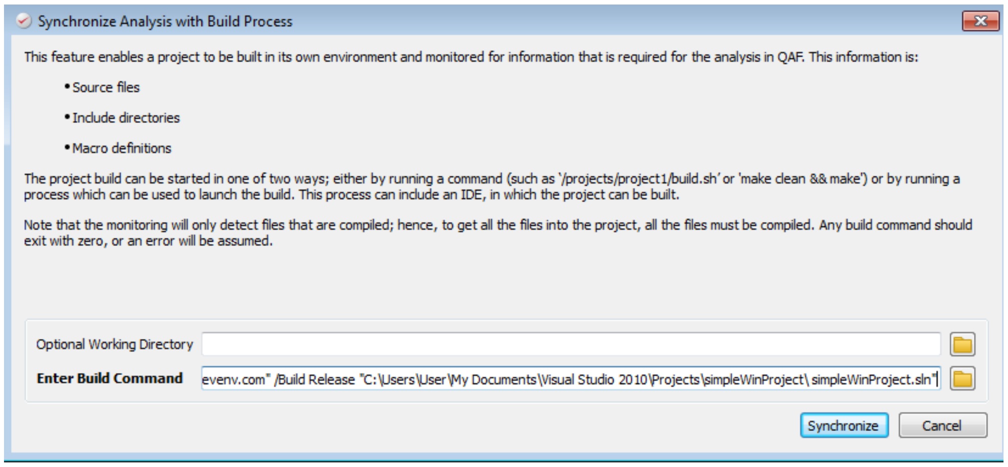


simpleTest02 - QA C/C++ Source Code Analysis Framework
Project Admin Analyze Report Upload View Help
New Project Ctrl+N
Open Project Ctrl+Alt+O
Close Project Ctrl+Alt+C
Synchronize Alt+A, Alt+B
Open Project Properties Ctrl+Alt+P
CMA Project Editor
Exit Application Ctrl+Alt+E
ysis of projec
Active
0
0
0

Figure 4.19: Synchronizing the new QAF project with the source code

and is presented with opportunity to enter the build command, which in this case is the lengthy single line:

```
"C:\Program Files (x86)\Microsoft Visual Studio 10.0\
Common7\IDE\devenv.com" /Build Release "C:\Users\User\
My Documents\Visual Studio 2010\Projects\simpleWinProject\
simpleWinProject.sln"
```



Synchronize Analysis with Build Process

This feature enables a project to be built in its own environment and monitored for information that is required for the analysis in QAF. This information is:

- Source files
- Include directories
- Macro definitions

The project build can be started in one of two ways; either by running a command (such as '/projects/project1/build.sh' or 'make clean && make') or by running a process which can be used to launch the build. This process can include an IDE, in which the project can be built.

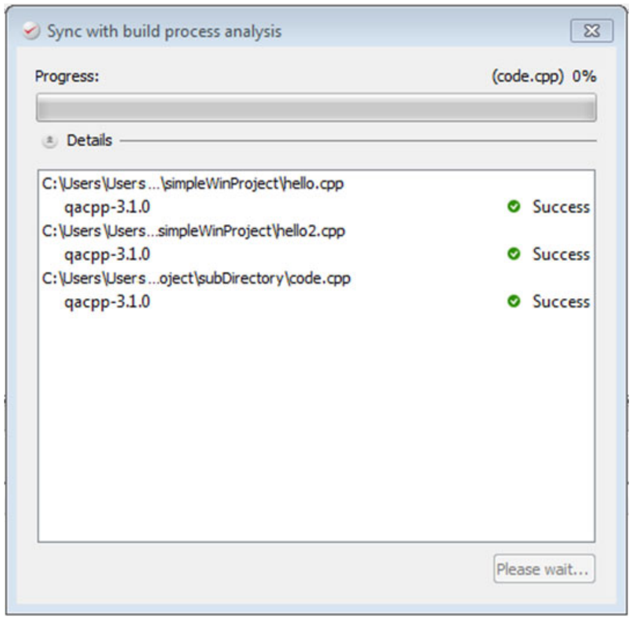
Note that the monitoring will only detect files that are compiled; hence, to get all the files into the project, all the files must be compiled. Any build command should exit with zero, or an error will be assumed.

Optional Working Directory:

Enter Build Command: evenv.com" /Build Release "C:\Users\User\My Documents\Visual Studio 2010\Projects\simpleWinProject\ simpleWinProject.sln"

Figure 4.20: Synchronization options

Upon selecting "Synchronize", the build is triggered and observed, with detected files being displayed in the progress window:



Sync with build process analysis

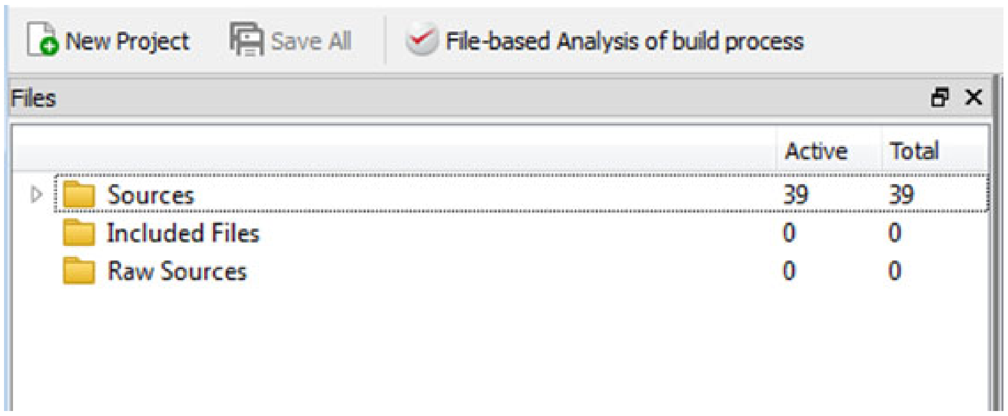
Progress: (code.cpp) 0%

Details

```
C:\Users\Users...\simpleWinProject\hello.cpp
    qacpp-3.1.0                                Success
C:\Users\Users...simpleWinProject\hello2.cpp
    qacpp-3.1.0                                Success
C:\Users\Users...oject\subDirectory\code.cpp
    qacpp-3.1.0                                Success
```

Figure 4.21: Build Process Analysis

and the files are available for examination in the file browser at left:



	Active	Total
Sources	39	39
Included Files	0	0
Raw Sources	0	0

Figure 4.22: Build Source File Selection

4.4 Use Case - Create a PRQAF Project Manually By Selecting Files

The preferred means of creating a PRQAF project is by the synchronization method as demonstrated in other use cases. However, there are two common situations in which a user might choose to manually identify files to be added to a PRQAF project. One of the key advantages of using the synchronization method will be lost in such cases; information about the specifics of the build (for example, compiler settings) will not automatically be part of the PRQAF project, and the user may have to manually add those details as well.

The first such situation occurs when a user simply cannot build their code project using the synchronization method. For example, perhaps the code is only built on a remote server on which the user cannot run PRQAF.

The second such situation involves minor changes to an existing PRQAF, such as adding a new source file, and the user does not wish to go through a complete rebuild of their code project. In such cases, it is possible to manually add files to a PRQAF project.

Firstly, the user must create a new PRQAF project, using the "New Project" button:

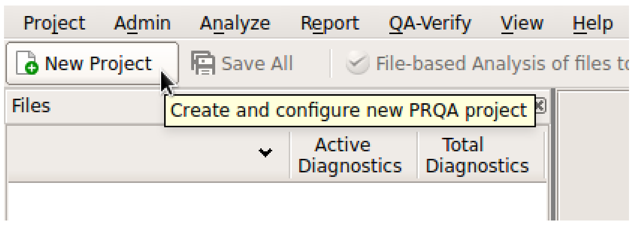
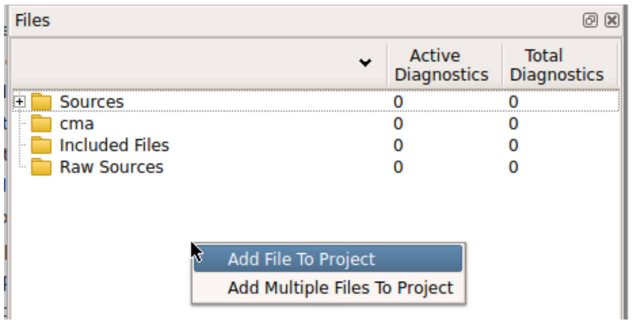


Figure 4.23: User selects New Project

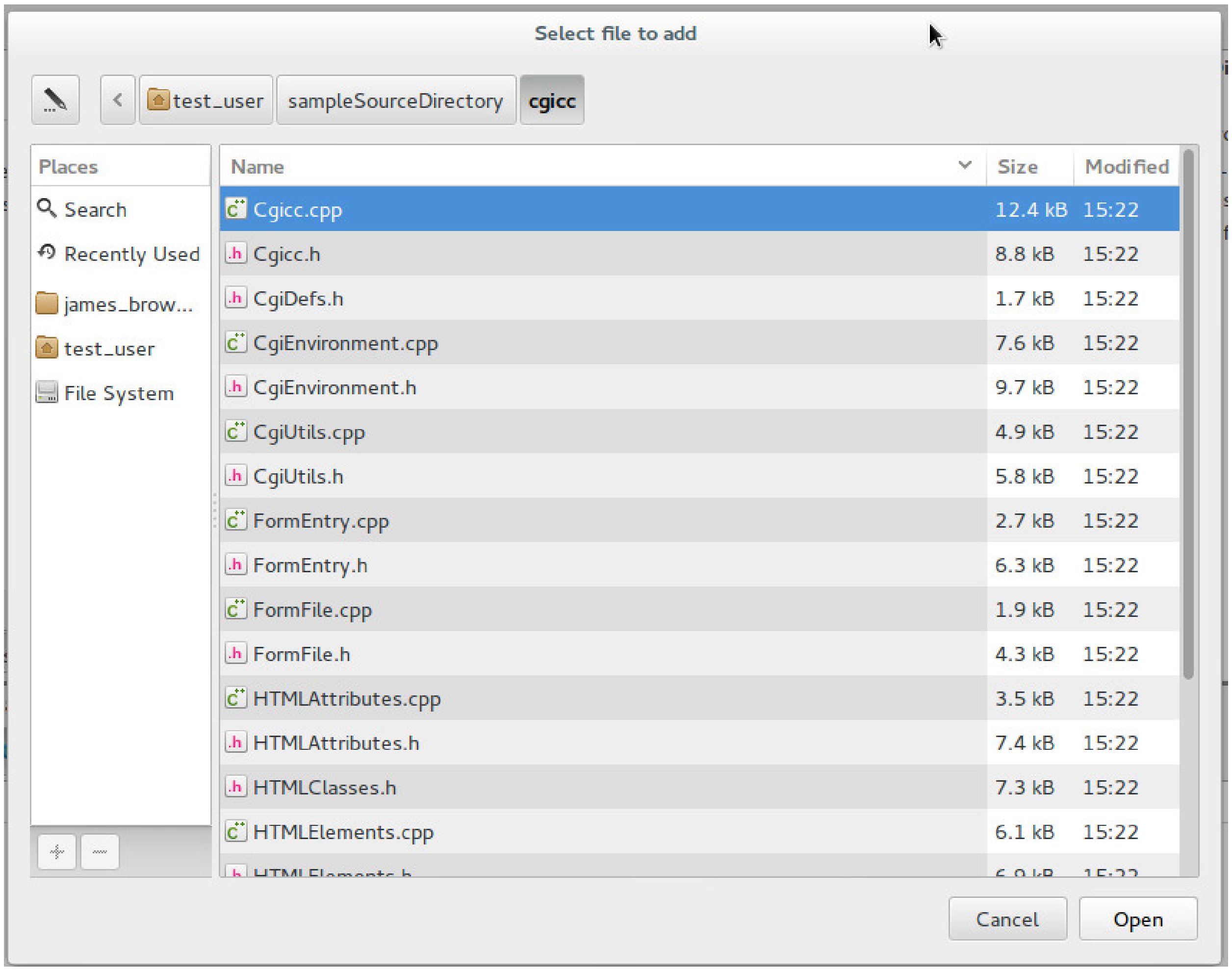
As in the other use cases, the user now selects the directory in which the PRQAF project is to reside, which would typically be a top-level source directory. An appropriate compiler is selected from the list, and the project is then created. The user is now presented with an empty project. Ideally, the next step would be to add files via the synchronization method as in other use cases. However, in this case, the user right-clicks on the project file explorer window and then selects the presented “Add File To Project” menu option:



	Active Diagnostics	Total Diagnostics
Sources	0	0
cma	0	0
Included Files	0	0
Raw Sources	0	0

Figure 4.24: The "Add File To Project" menu option

Selecting this menu option will present the user with a typical file selection dialog:



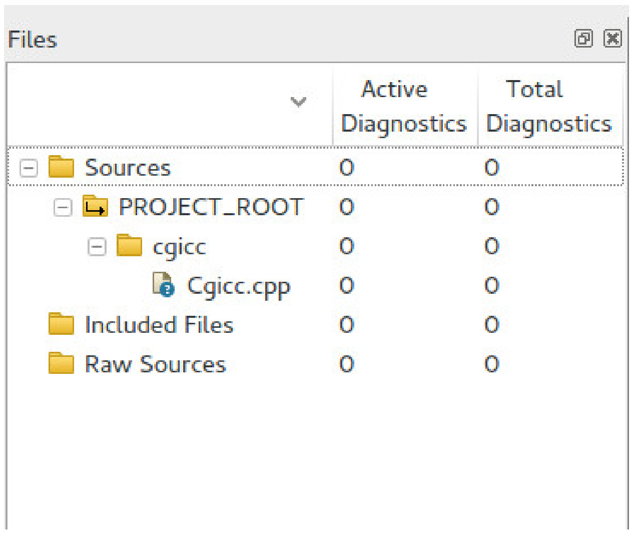
Name	Size	Modified
Cgicc.cpp	12.4 kB	15:22
Cgicc.h	8.8 kB	15:22
CgiDefs.h	1.7 kB	15:22
CgiEnvironment.cpp	7.6 kB	15:22
CgiEnvironment.h	9.7 kB	15:22
CgiUtils.cpp	4.9 kB	15:22
CgiUtils.h	5.8 kB	15:22
FormEntry.cpp	2.7 kB	15:22
FormEntry.h	6.3 kB	15:22
FormFile.cpp	1.9 kB	15:22
FormFile.h	4.3 kB	15:22
HTMLAttributes.cpp	3.5 kB	15:22
HTMLAttributes.h	7.4 kB	15:22
HTMLClasses.h	7.3 kB	15:22
HTMLElements.cpp	6.1 kB	15:22

Figure 4.25: The "Selecting a Single File To Add" menu option

In this use case, the user adds a single file;

```
Cgicc.cpp
```

Following selection, the file appears in the project files explorer window:

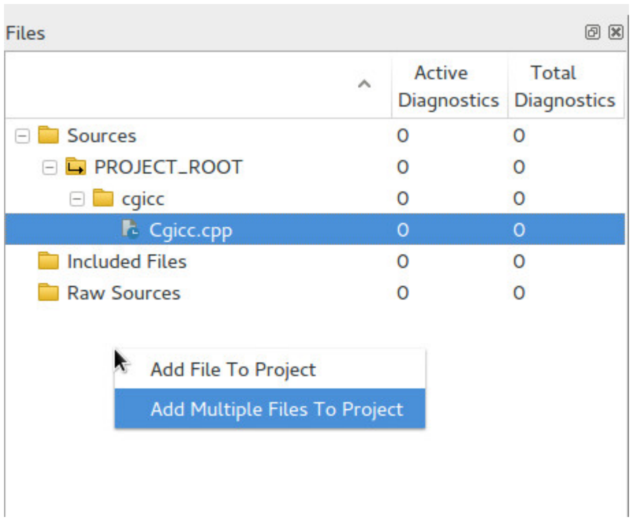


Files

	Active Diagnostics	Total Diagnostics
Sources	0	0
PROJECT_ROOT	0	0
cgicc	0	0
Cgicc.cpp	0	0
Included Files	0	0
Raw Sources	0	0

Figure 4.26: Selected File Appears in Project Files Explorer

The project (containing just this single file) can then be analyzed as usual. Alternatively, the user can select multiple files to add simultaneously:

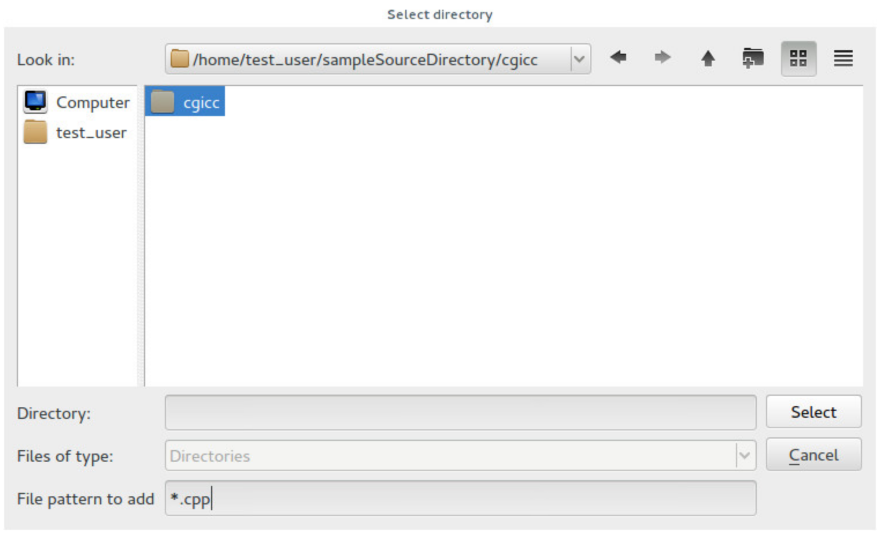


Files

	Active Diagnostics	Total Diagnostics
Sources	0	0
PROJECT_ROOT	0	0
cgicc	0	0
Cgicc.cpp	0	0
Included Files	0	0
Raw Sources	0	0

Figure 4.27: The “Add Multiple Files To Project” menu option

This will present a directory browser, with which the user browses to a directory and enters a wildcard filter to identify by extension files to add:



Select directory

Look in: /home/test_user/sampleSourceDirectory/cgicc

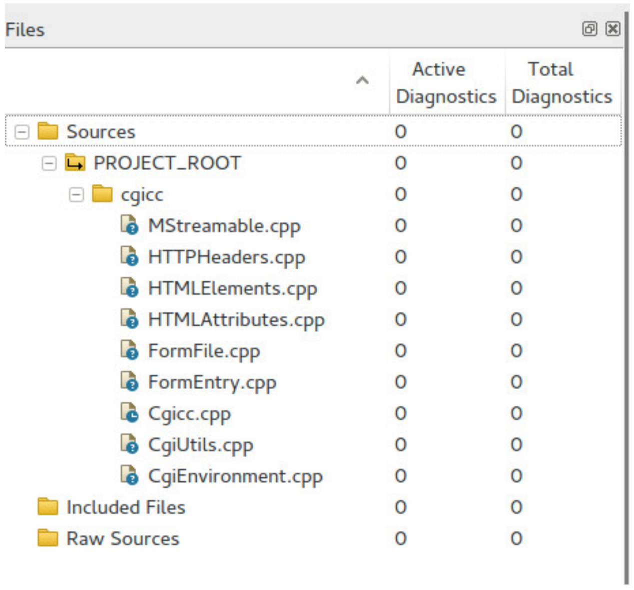
Directory:

Files of type: Directories

File pattern to add *.cpp

Figure 4.28: The user selects a directory and enters a wildcard filter in the “File Pattern To Add” field

In this case, the directory contains a number of files matching the chosen pattern, and they are added to the project and appear in the project files explorer window:



Files

	Active Diagnostics	Total Diagnostics
Sources	0	0
PROJECT_ROOT	0	0
cgicc	0	0
MStreamable.cpp	0	0
HTTPHeaders.cpp	0	0
HTMLElements.cpp	0	0
HTMLAttributes.cpp	0	0
FormFile.cpp	0	0
FormEntry.cpp	0	0
Cgicc.cpp	0	0
CgiUtils.cpp	0	0
CgiEnvironment.cpp	0	0
Included Files	0	0
Raw Sources	0	0

Figure 4.29: Many files have been added simultaneously to the project

5 Creating Your Own Projects - Java and C#

Java and C# projects cannot make use of the automatic project synchronisation method, and the user must create an empty PRQAF project and then indicate manually which source files are to be added.

5.1 Use Case - Analyze a Java or C# Project

Firstly, the user must create a new PRQAF project, using the "New Project" button:

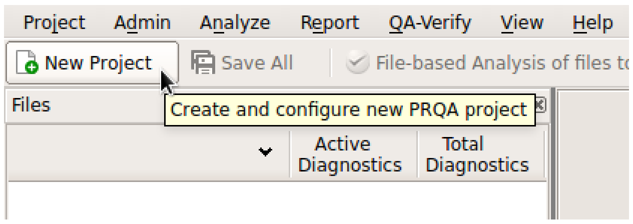
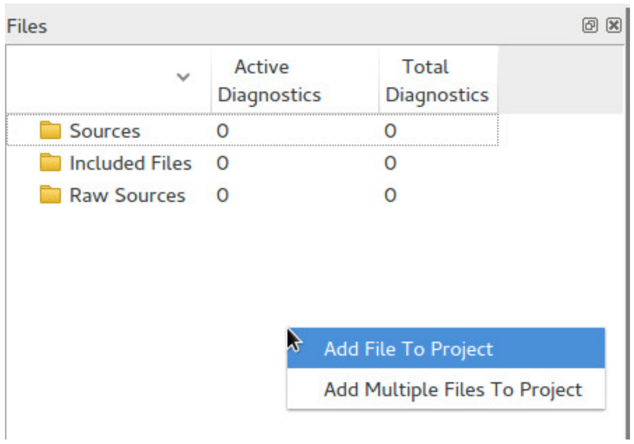


Figure 5.1: User selects New Project

When presented with the project creation dialog, the user must select a number of Java or C# language specific options. In particular, the following fields have specific values that the user should be sure to select:

- Language Family
- Analysis Configuration File
- CCT

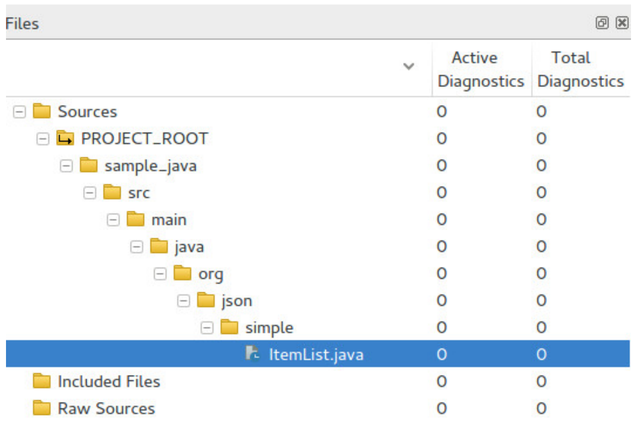
Upon creation, the user is presented with an empty project. To add specific Java source files, the user right-clicks on the project file explorer window and then selects the presented “Add File To Project” menu option:



	Active Diagnostics	Total Diagnostics
Sources	0	0
Included Files	0	0
Raw Sources	0	0

Figure 5.2: The “Add File To Project” menu option

The user will then browse to identify the file to add, and select it. Following this, the selected file is added to the project and displayed in the Project Explorer:



	Active Diagnostics	Total Diagnostics
Sources	0	0
PROJECT_ROOT	0	0
sample_java	0	0
src	0	0
main	0	0
java	0	0
org	0	0
json	0	0
simple	0	0
ItemList.java	0	0
Included Files	0	0
Raw Sources	0	0

Figure 5.3: A java file has been added to the project

At this point, the user can analyse files as normal.

6 Conclusion

The user has been led through a set of simple analysis and test projects; this should enable a newcomer to the software to begin analyzing their own code projects. There are many more advanced options and capabilities available to the user; these are discussed in the full documentation.

For assistance and help, visit the PRQA website and contact support through the website.