Using valgrind with Qt Creator

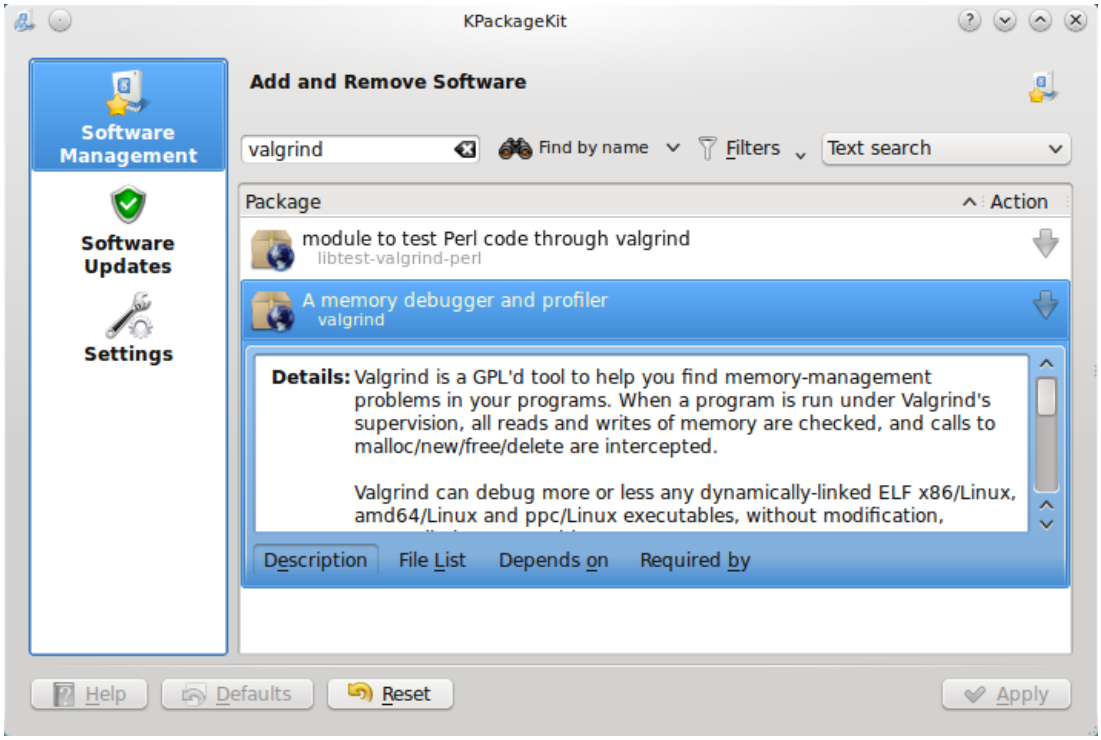
This article will show you how you can use valgrind on Linux to find memory leaks in your Qt based application from within Qt Creator.

07 Nov 2010

valgrind is a low level tool that can do many useful memory-related things, of which we will focus on locating memory leaks - especially useful for long running processes and those that create and destroy a lot of objects. To do this, valgrind tracks the memory allocation/deallocations your application does and generates reports about places that might be losing memory.

Install valgrind

Almost all Linux distributions include valgrind, but usually do not install it by default. Just select valgrind from your package manager (shown in Kubuntu below) to install it.



If you prefer to use the console, all you need to do on a deb based Linux is

```
sudo apt-get install valgrind
```

About 50 MB later, you will have valgrind installed

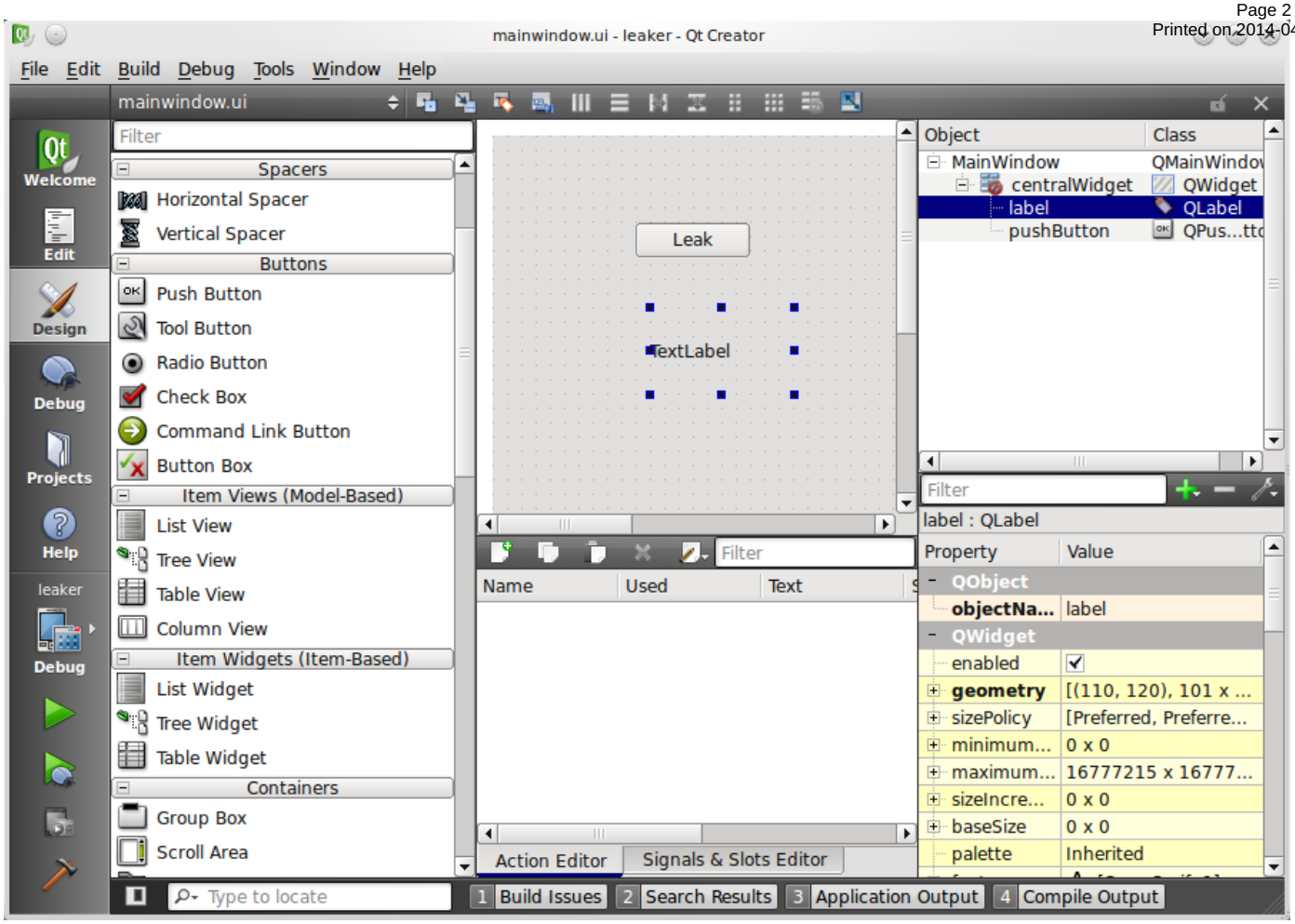
Install the (Nokia) Qt SDK or Qt Creator

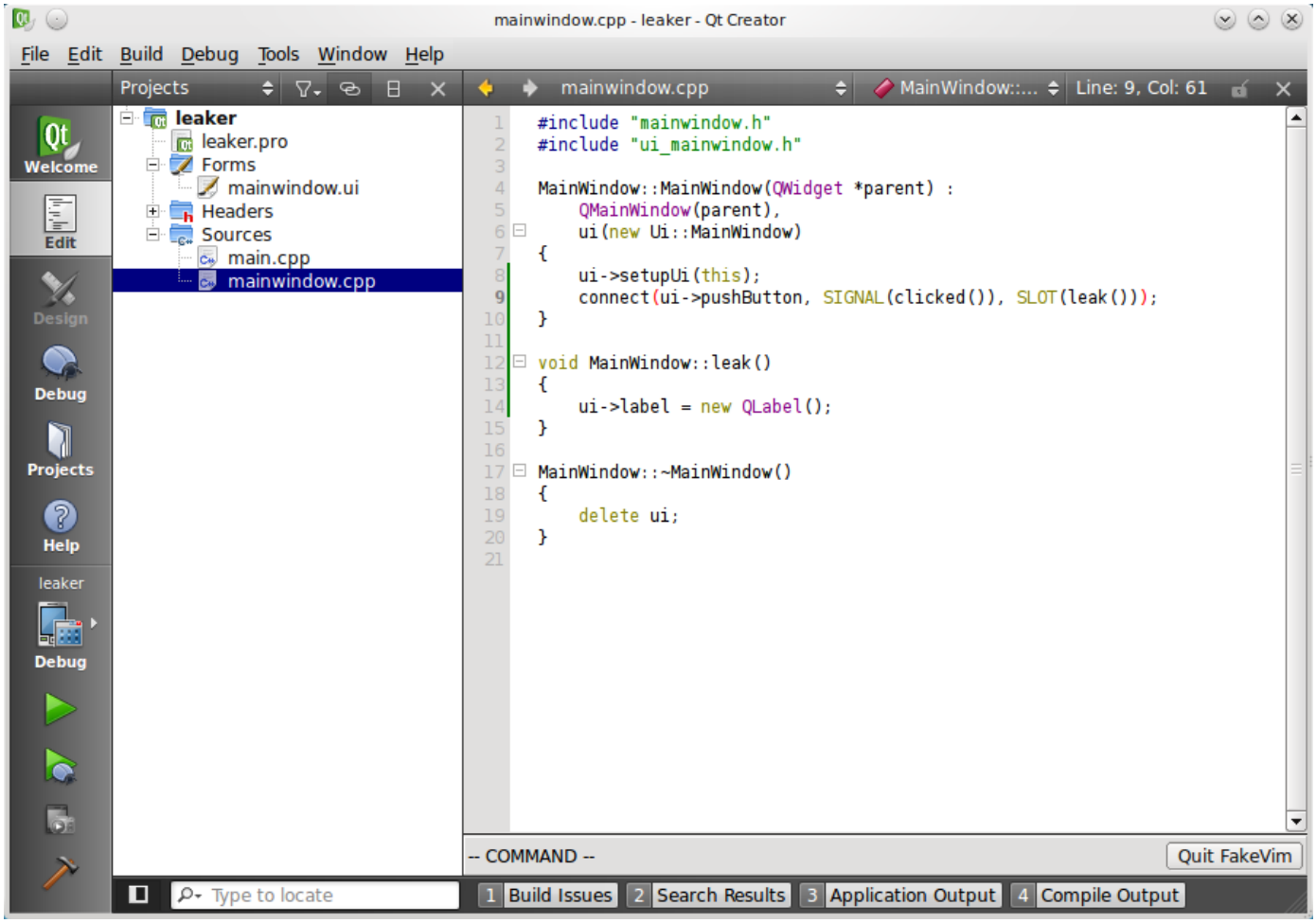
If you do not already have it, you can install the Nokia Qt SDK, but the procedure below will work with any install of Qt Creator. The approach outlined in this article can be used for desktop, Maemo and Simulator targets.

Open project

Create or open an existing project. Here an example of simple application is created. Notice how this example intentionally leaks a QLabel object every time the button is pressed.

(C) Copyright Nokia 2014. All rights reserved.

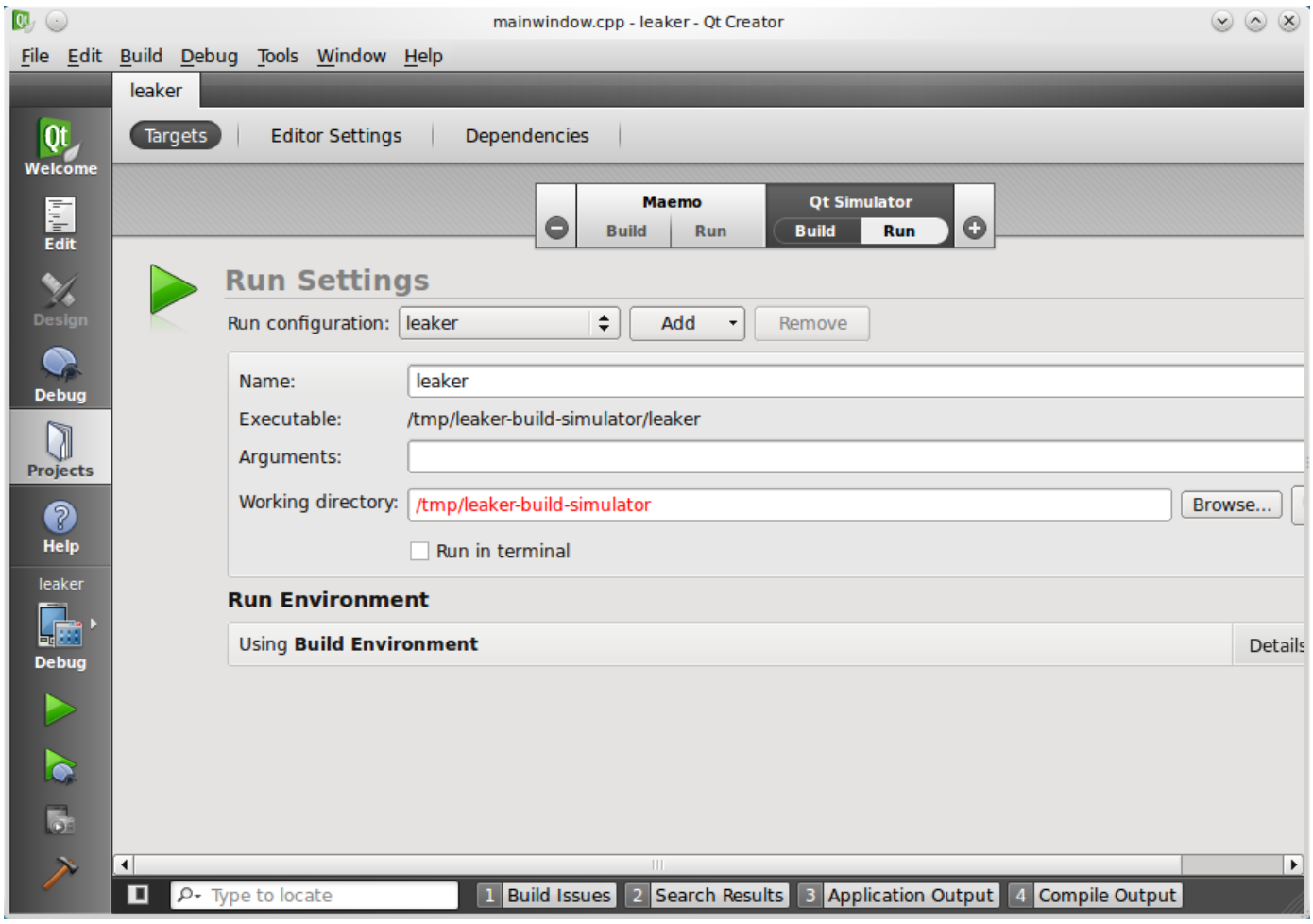




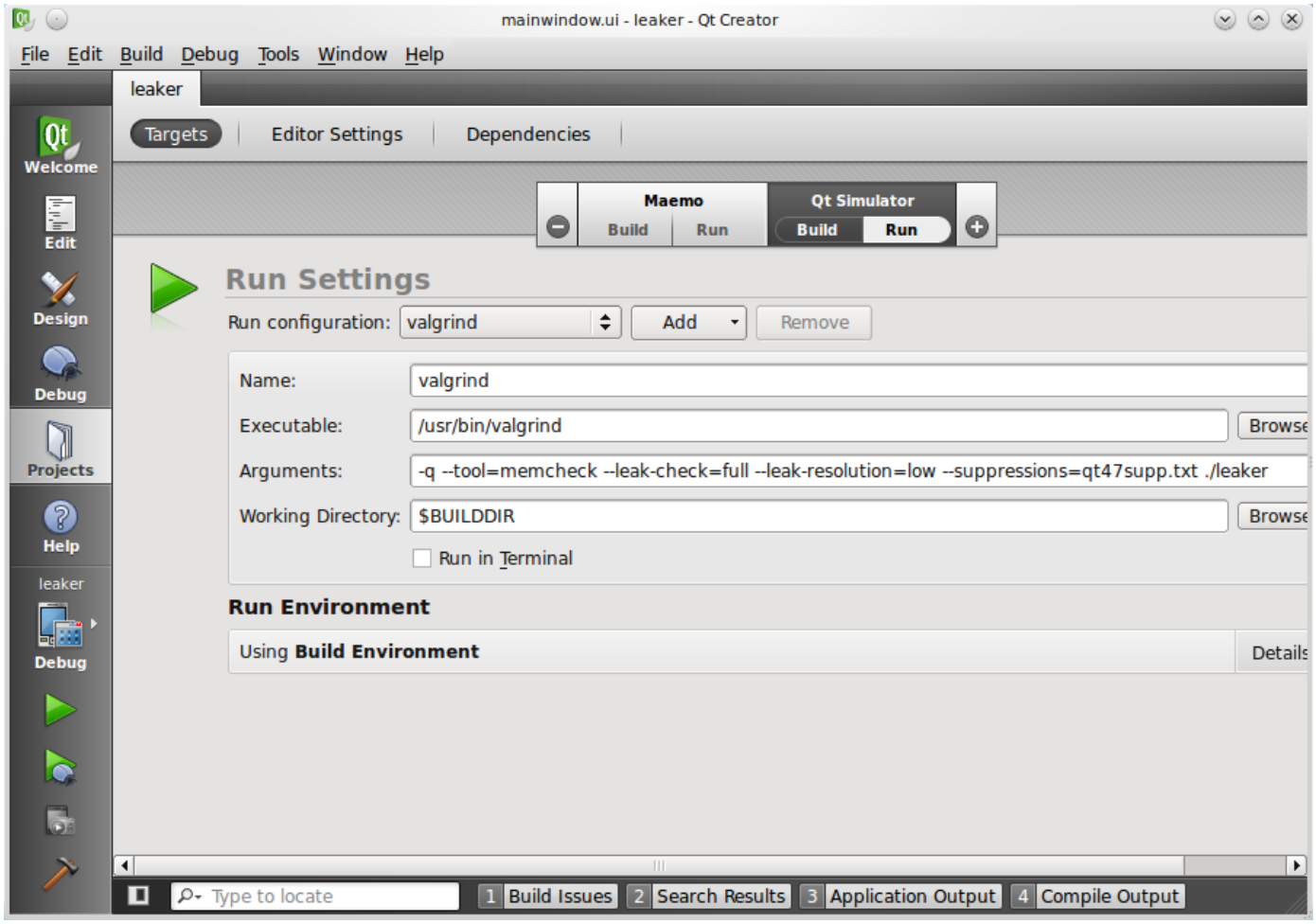
Create run target

(C) Copyright Nokia 2014. All rights reserved.

In order to use valgrind, a special run target needs to be added to the project. Click the projects icon and then the 'run' section of the target you wish to debug. You will see something like



Now click 'add' and specify a new target that looks like the following



(C) Copyright Nokia 2014. All rights reserved.

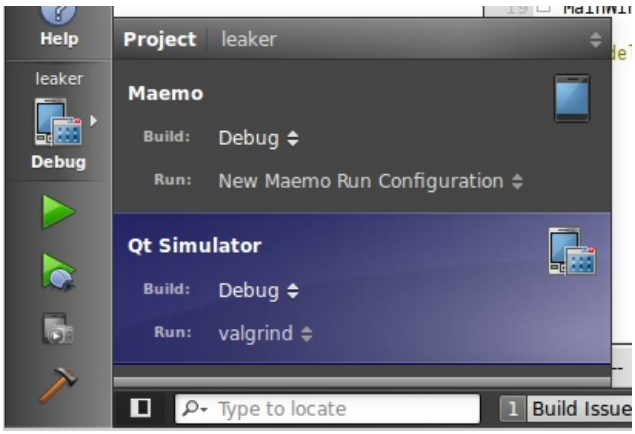
The complex argument line in text form for copy-pasting:

```
-q --tool=memcheck --leak-check=full --leak-resolution=low --suppressions=Qt47supp.txt
./[your-app-target-name]
```

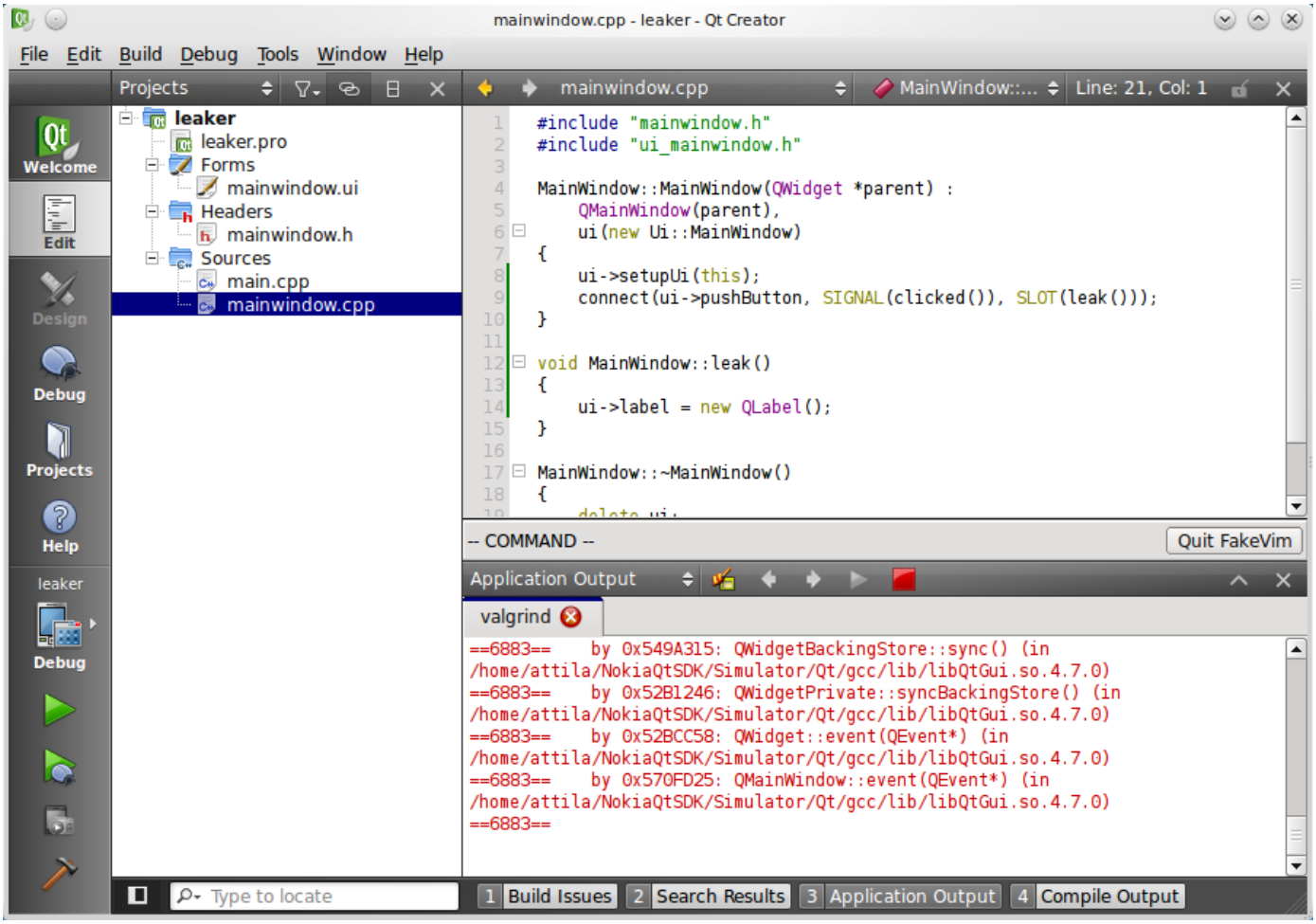
As an initial default Qt4.7 suppression file (see below) for simulator targets you can use File:Qt47supp.txt. Of course, depending on what modules or additional libraries use, you will want to extend it.

Running the application

You will see that now in your target window you have a new run configuration selectable



Make sure you're adding/selecting valgrind in the debug build, as it will not be able to find symbols and code lines in a release build. Upon running the application, there will be a lot of output from valgrind, intertwined with the actual output from your application. Notice that valgrind will print most of its finding when you exit the application. If you followed the article, you'll see a large amount of valgrind output.



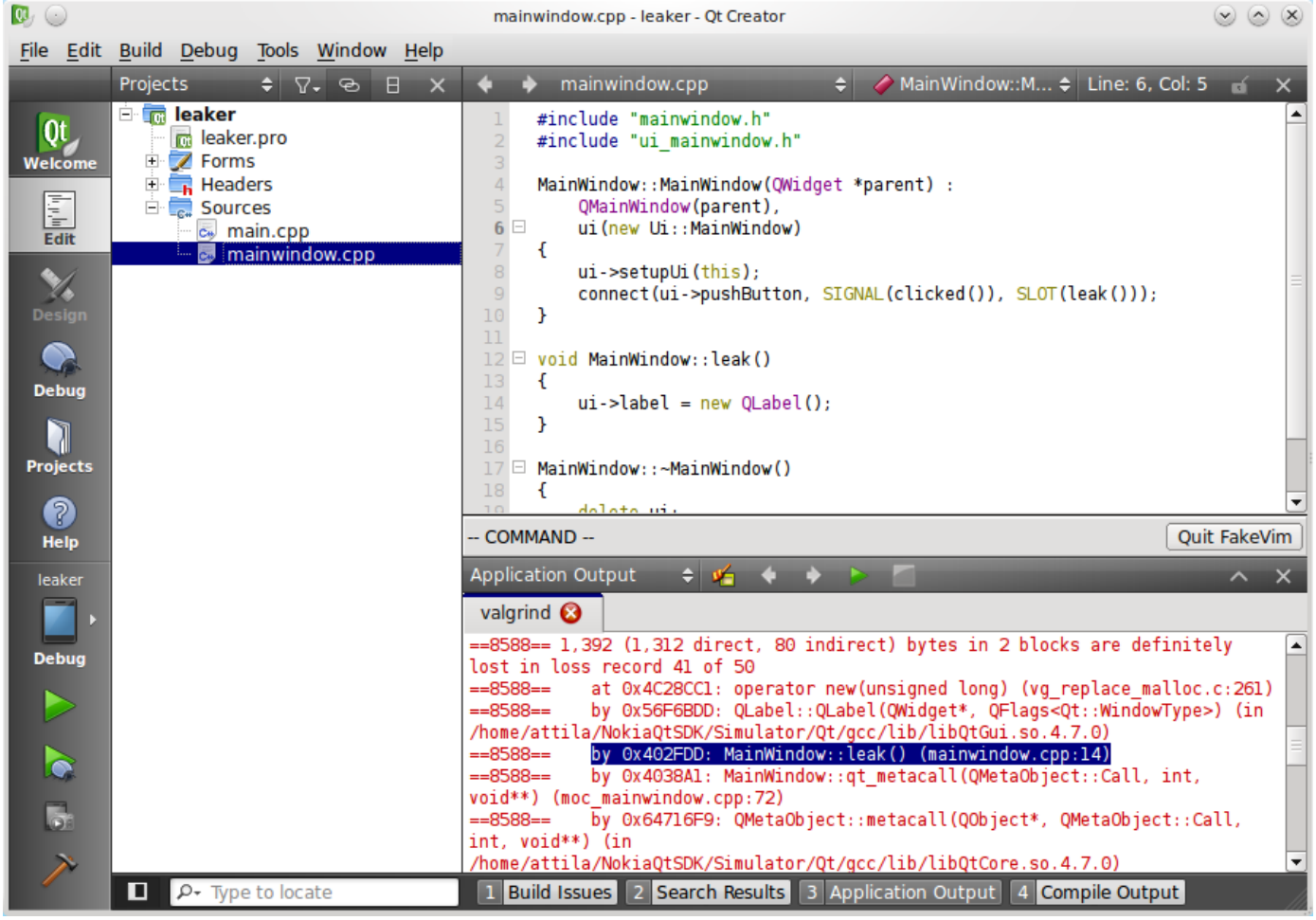
Interpreting the results

(C) Copyright Nokia 2014. All rights reserved.

You will soon realize that valgrind's output can easily become overwhelming. One of the keys of dealing with valgrind efficiently is using good suppression files. Using these files avoids warnings that are false alarms or located in code outside of ours. Generally, you should pay no attention to valgrind complaints about errors in glibc or Qt - it is a common mistake to start valgrind without suppression files and then think that everything is leaking, but that is generally just the side-effect of lower level libraries deallocating memory after valgrind. Thus, to be able to focus on actual leaks in our code, a suppression file needs to be generated that will bring down the valgrind output to a manageable size containing only errors in the code at hand.

As an initial default Qt4.7 suppression file for simulator targets you can use File:Qt47supp.txt. Of course, depending on what modules or additional libraries use, you will want to extend it.

Now, save the suppression file in the build directory and re-run the application. Now you can easily spot the memory leak report with a precise source file and location where the action that caused the leak occurred.



Qt making your life easier

An important note is that in many cases you can prevent memory leaks just by using Qt's smart pointer classes, QScopedPointer and QSharedPointer being of particular interest for most common use-cases.

The QObject class also has mechanisms that help avoid dangling pointers and memory leaks. When a QObject is deleted, it emits a destroyed() signal and also automatically calls the destructor's of its child objects.

Summary

The above should start you down path of finding leaks in your Qt applications. Should you require more detailed reports or want to find out more about what valgrind can do for you, check the Valgrind manual.

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.