ROS Qt Creator Plug-in

Jul 24, 2023

Contents

CHAPTER 1

Installation

1.1 How to Install (Users)

This wiki explains the procedure for installing the ROS Qt Creator Plug-in.

Note: If you primarily want to use this tool for development of other ROS packages (ie: not to work on the plugin itself), please follow the following instructions.

1.1.1 Installation

Important: The install method has changed from using the ppa method to a custom installer. This is to enable the ability to provide richer support leveraging existing ros tools which was not possible using the ppa.

Installation Procedure for Ubuntu 18.04

1. Download Installer:
    1. Bionic Online Installer (Recommended)
    2. Bionic Offline Installer

    Note: The Offline Installer is to be used on machines that do not have internet access.

2. Next proceed to *Qt Installer Procedure*

Installation Procedure for Ubuntu 16.04

1. Download Installer:
    1. Xenial Online Installer (Recommended)
    2. Xenial Offline Installer

    Note: The Offline Installer is to be used on machines that do not have internet access.

2. Next proceed to *Qt Installer Procedure*

Important: If previously installed using the ppa please follow the procedure below to remove old version.

```
sudo apt install ppa-purge
sudo ppa-purge -o beineri
sudo ppa-purge levi-armstrong/qt-libraries-xenial
sudo ppa-purge levi-armstrong/ppa
```

Warning: The ppa-purge removes everything installed from the ppa, so if the ppa is used for other development do not purge.

Installation Procedure for Ubuntu 14.04

1. Download Installer:
    1. Trusty Online Installer (Recommended)
    2. Trusty Offline Installer

    Note: The Offline Installer is to be used on machines that do not have internet access.

2. Next proceed to *Qt Installer Procedure*

Important: If previously installed using the ppa please follow the procedure below to remove old version.

```
sudo apt install ppa-purge
sudo ppa-purge -o beineri
sudo ppa-purge levi-armstrong/qt-libraries-xenial
sudo ppa-purge levi-armstrong/ppa
```

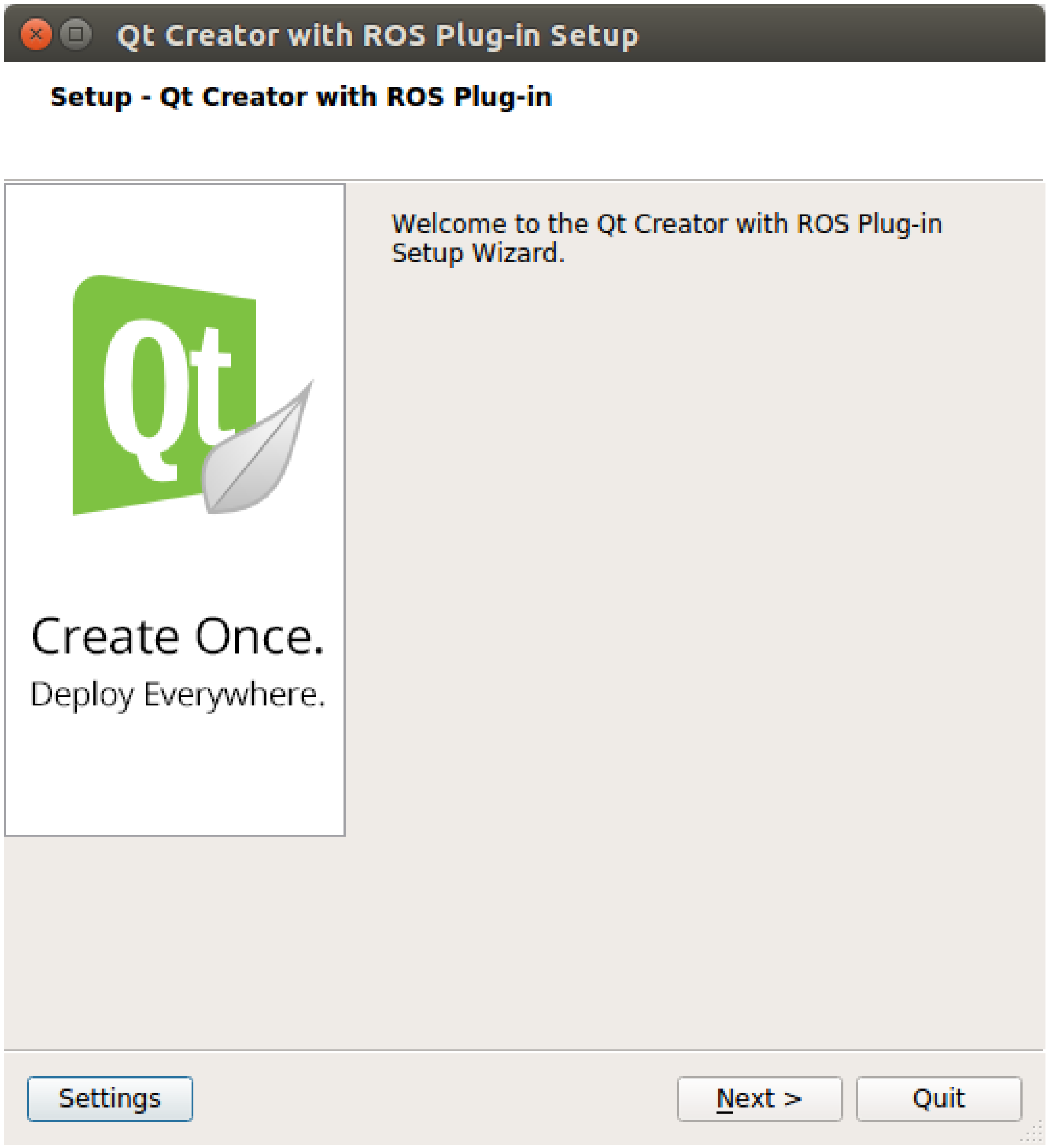
Warning: The ppa-purge removes everything installed from the ppa, so if the ppa is used for other development do not purge.

Archived Versions

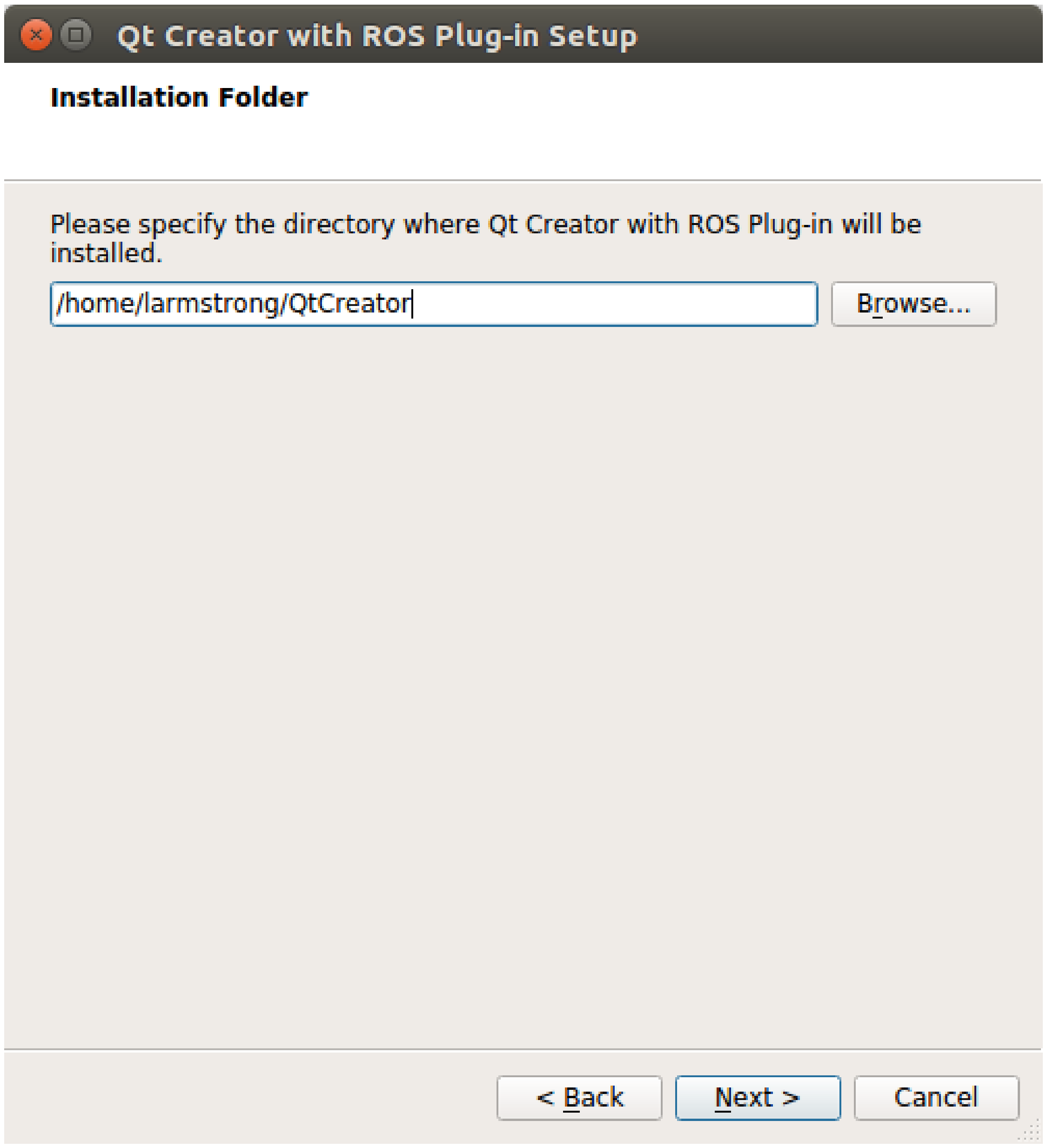
If for some reason you need a version other than the latest, all installers may be found here.

Qt Installer Procedure

1. Then right click on the installer file, select properties and enable execution under permissions.
2. Next double click the installer and it should open and step throught the installer.

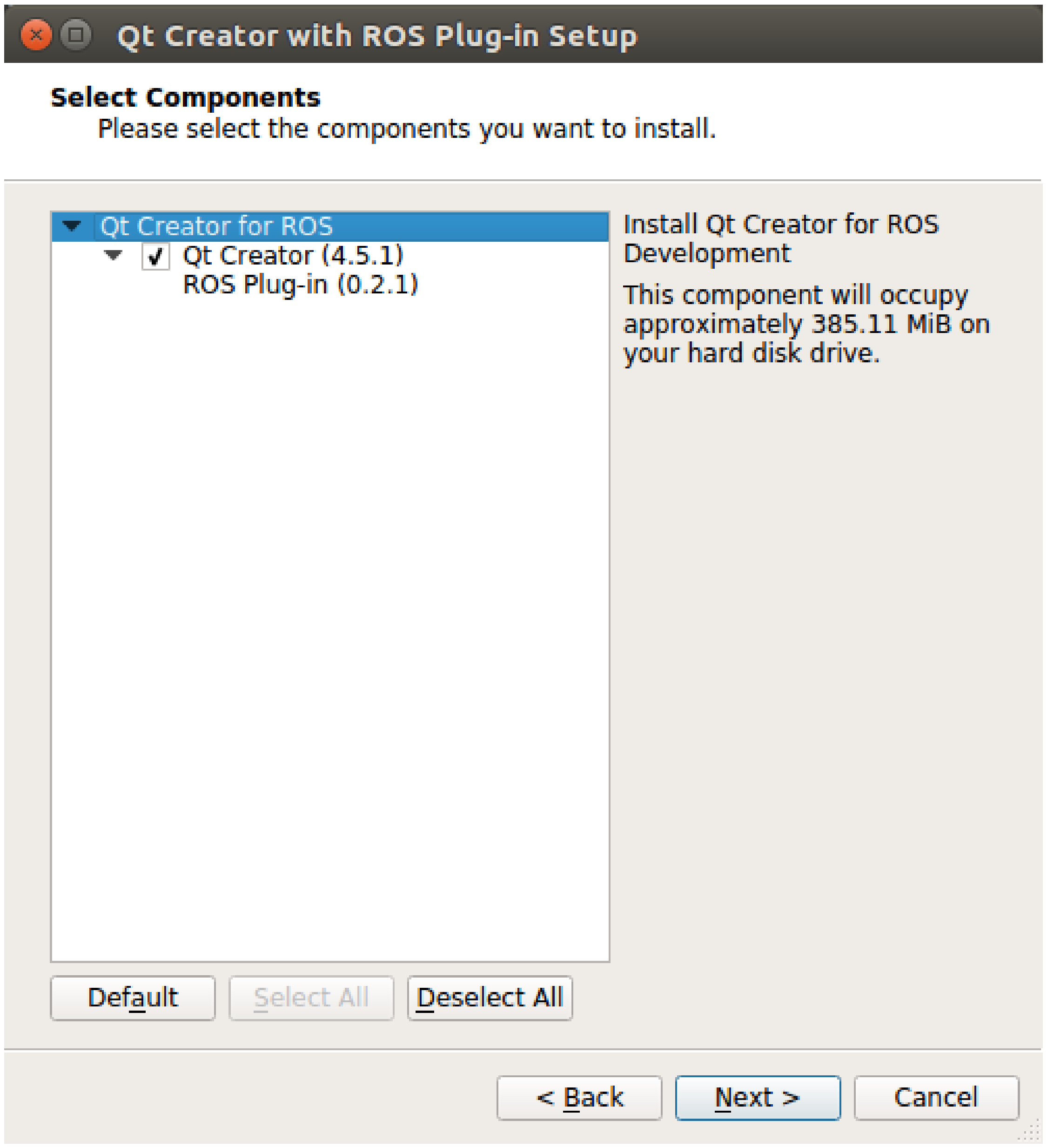

Qt Creator with ROS Plug-in Setup
Setup - Qt Creator with ROS Plug-in
Create Once.
Deploy Everywhere.
Settings
Next >
Quit

Welcome to the Qt Creator with ROS Plug-in Setup Wizard.



Please specify the directory where Qt Creator with ROS Plug-in will be installed.

/home/larmstrong/QtCreator


Qt Creator with ROS Plug-in Setup

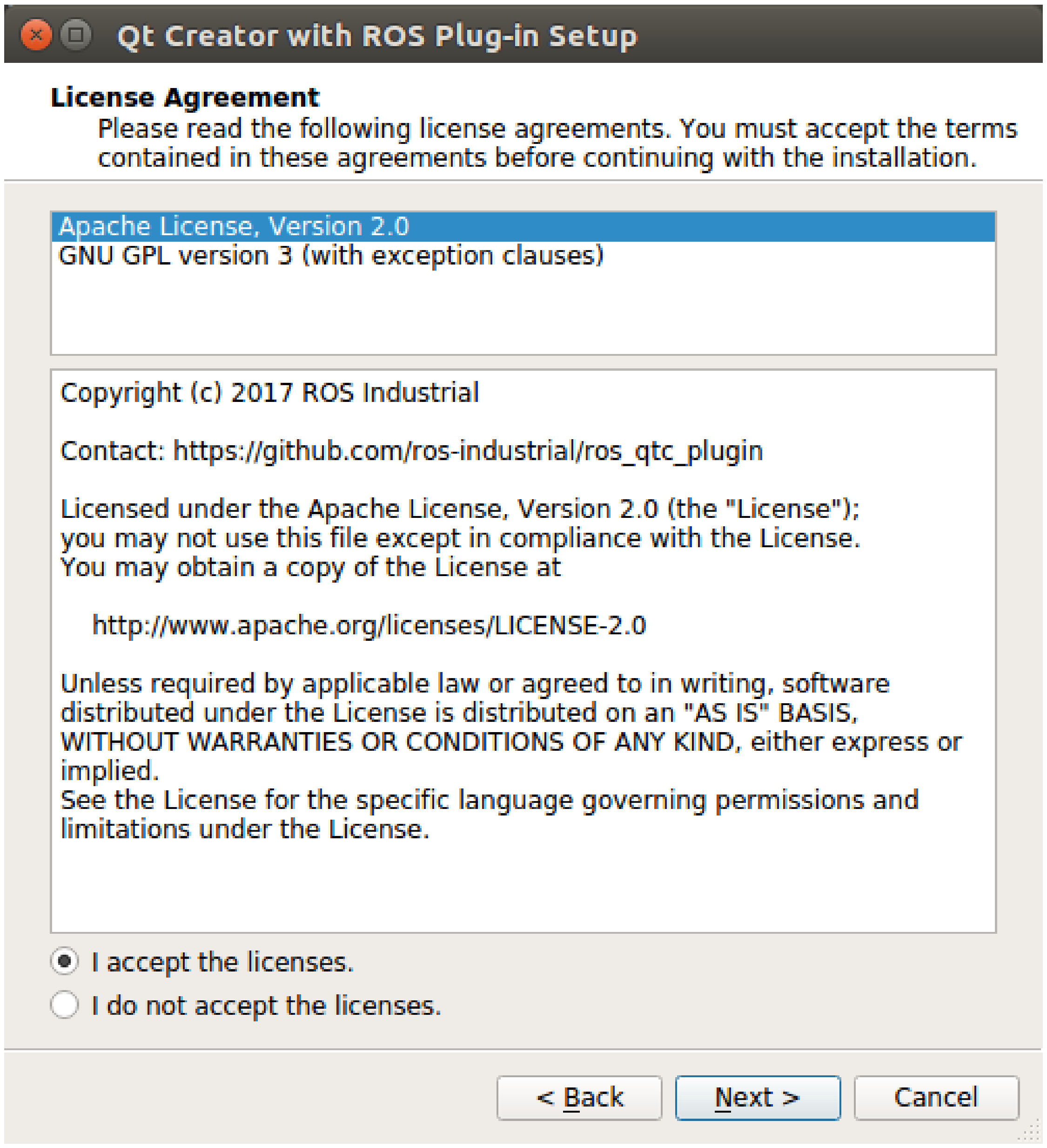
Select Components

Please select the components you want to install.

Qt Creator for ROS
✓ Qt Creator (4.5.1)
ROS Plug-in (0.2.1)

Install Qt Creator for ROS Development

This component will occupy approximately 385.11 MiB on your hard disk drive.



License Agreement

Please read the following license agreements. You must accept the terms contained in these agreements before continuing with the installation.

Apache License, Version 2.0
GNU GPL version 3 (with exception clauses)

Copyright (c) 2017 ROS Industrial

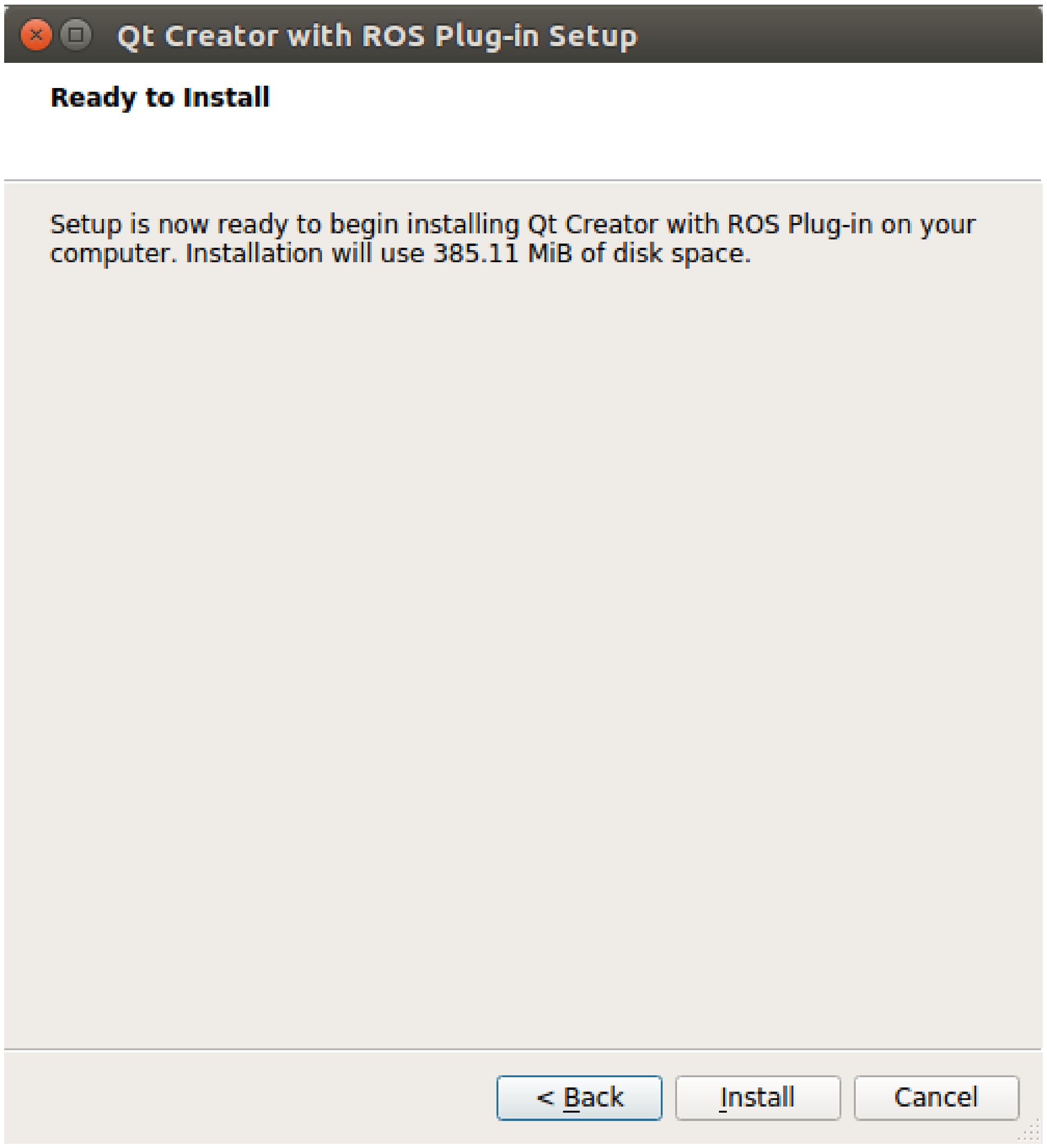
Contact: https://github.com/ros-industrial/ros_qtc_plugin

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

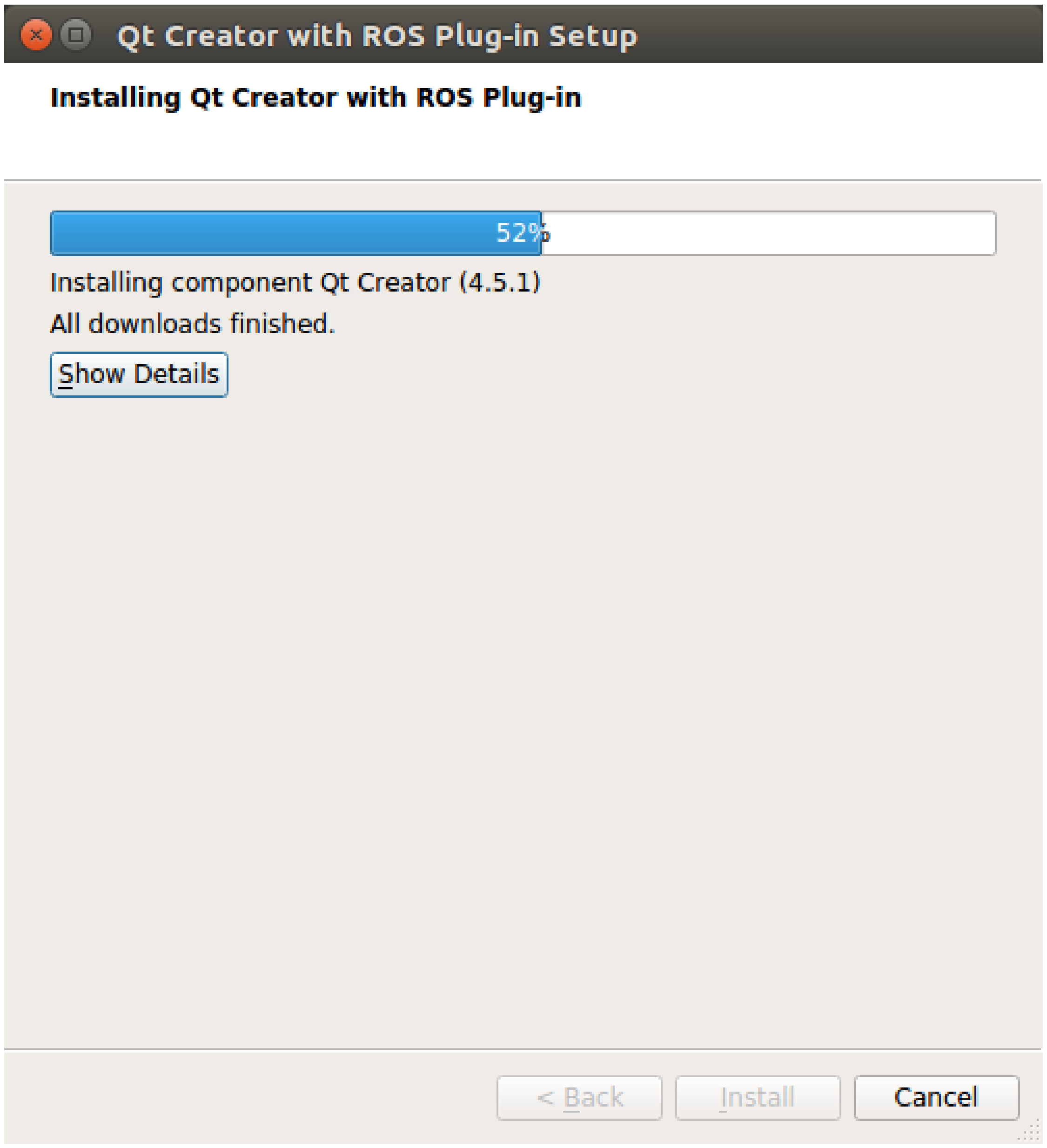
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and limitations under the License.

[✓ I accept the licenses.]
I do not accept the licenses.



Ready to Install

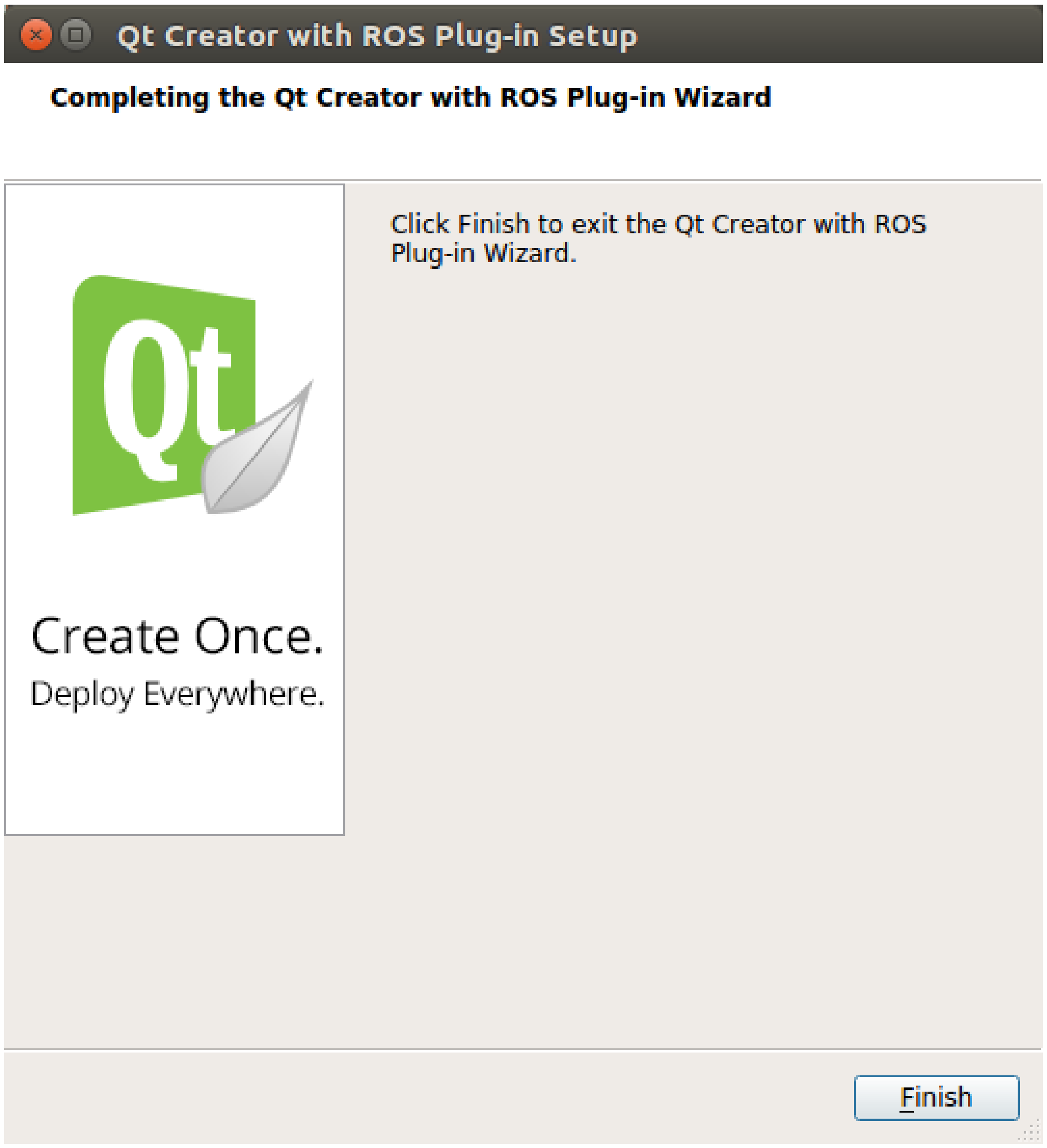
Setup is now ready to begin installing Qt Creator with ROS Plug-in on your computer. Installation will use 385.11 MiB of disk space.



Installing Qt Creator with ROS Plug-in

Installing component Qt Creator (4.5.1)

All downloads finished.

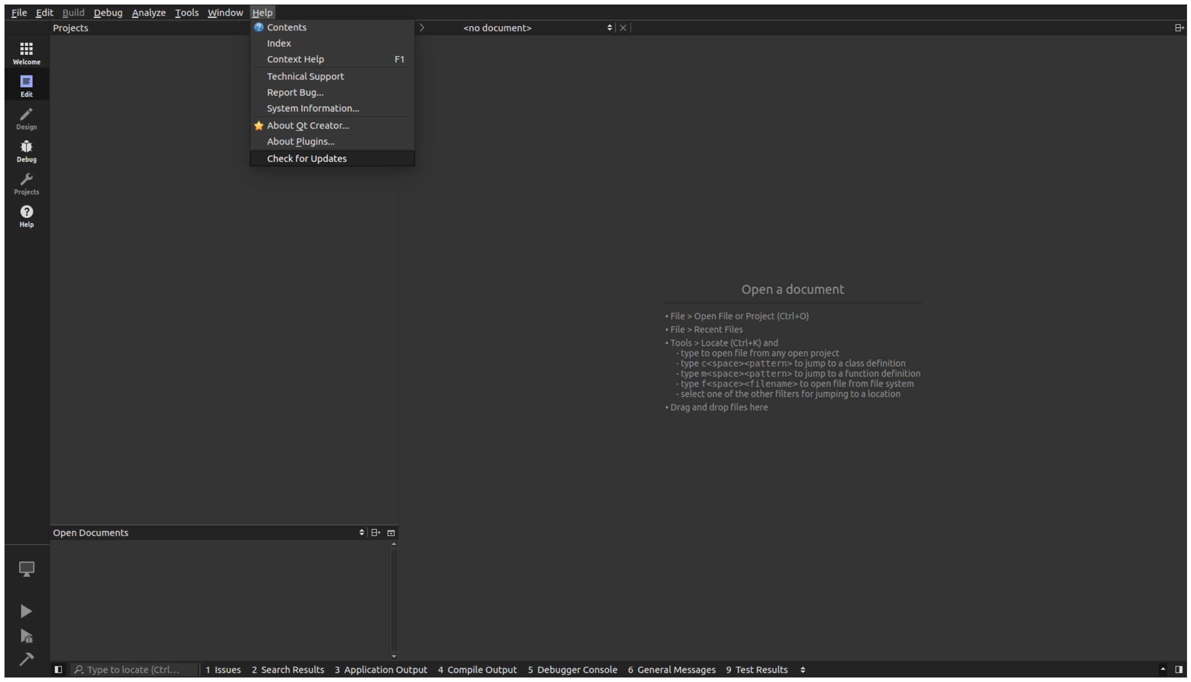


Qt Creator with ROS Plug-in Setup

Completing the Qt Creator with ROS Plug-in Wizard

Click Finish to exit the Qt Creator with ROS Plug-in Wizard.

1. How to get future updates, open Qt Creator and on the menubar under Help select **“Check for Updates”**.



Open a document

- File > Open File or Project (Ctrl+O)
- File > Recent Files
- Tools > Locate (Ctrl+K) and
  - type to open file from any open project
  - type c<space><pattern> to jump to a class definition
  - type m<space><pattern> to jump to a function definition
  - type f<space><filename> to open file from file system
  - select one of the other filters for jumping to a location
- Drag and drop files here

1.1.2 Installation Issues and Conflicts

- No known issues

1.1.3 Testing Plugin.

1. Start Qt Creator
   - Option 1: Launch using the desktop icon.
   - Option 2: Launch from terminal.

   ```
   qtcreator-ros
   ```

2. To verify that the plugin exist, goto File>New File or Project>Projects>Other Project>ROS Workspace. If the ROS Workspace is present then everything built correctly and is ready for development and testing.

1.2 How to Install (Developers)

Note: If you'd like to contribute to the development of the ROS Qt Creator Plug-in, you are considered a *developer*, please follow the following instructions.

1.2.1 Installation

Installation Dependencies for Ubuntu 20.04

```
sudo apt update
sudo apt install libgl1-mesa-dev ninja-build libyaml-cpp-dev libqtermwidget5-0-dev␣
↪libutf8proc-dev
sudo apt install python3-pip
pip install pyyaml requests py7zr
```

1.2.2 Run ROS Qt Creator setup script

1. Clone the latest *devel* version:

   ```
   git clone https://github.com/ros-industrial/ros_qtc_plugin.git -b devel
   ```

2. Navigate to *ros_qtc_plugin* and run the setup script to download additional Qt and Qt Creator dependencies:

   ```
   ./setup.py
   ```

Note: The script will download the official binary distributions of Qt and Qt Creator and thus will only support commonly used architectures. If you want to build the plugin on unsupported architectures, you have to build them from source or use a Linux distribution that provides sufficiently new versions of these packages.

Note: The script will download the latest versions that are compatible with the *devel* branch. The versions can be changed in *versions.yaml* to build against an older or newer version.

Note: Alternatively to the setup script, you can use the official binary installer for Qt and Qt Creator. In this case, you have to select the “Plugin Development” package to install the development headers, and “Debug Symbols” if you want your gdb backtrace to contain Qt Creator symbols.

1.2.3 Build plugin

1. Compile the plugin and create a plugin package

   ```
   cmake -B build -GNinja -DCMAKE_BUILD_TYPE=Debug -DCMAKE_PREFIX_PATH="/tmp/qtc_sdk/
   ↪Tools/QtCreator;/tmp/qtc_sdk/5.15.0/gcc_64"
   cmake --build build --target package
   ```

1.2.4 Testing Plugin

1. Extract the archive to your Qt Creator installation and execute the command below or launch using the desktop launcher.

   ```
   qtcreator
   ```

2. To verify that the plugin exist, goto File>New File or Project>Projects>Other Project>ROS Workspace. If the ROS Workspace is present then everything built correctly and is ready for development and testing.

1.2.5 Debug issues with Plugin

1. The instructions above compile the plugin with debug symbols. You can then debug the plugin by starting Qt Creator with gdb:

   ```
   gdb --ex=r --args qtcreator
   ```

2. After the plugin segfaults, print a backtrace:

   ```
   (gdb) bt
   ```

   and share it on GitHub in a new or active issue.

Note: By default, Qt Creator does not come with debug symbols. The backtrace will only contain symbols of the plugin. Qt Creator debug symbols can be installed either by downloading them from the same source as the setup script, or via the official binary distribution installer (package "Debug Symbols").

CHAPTER 2

FAQ

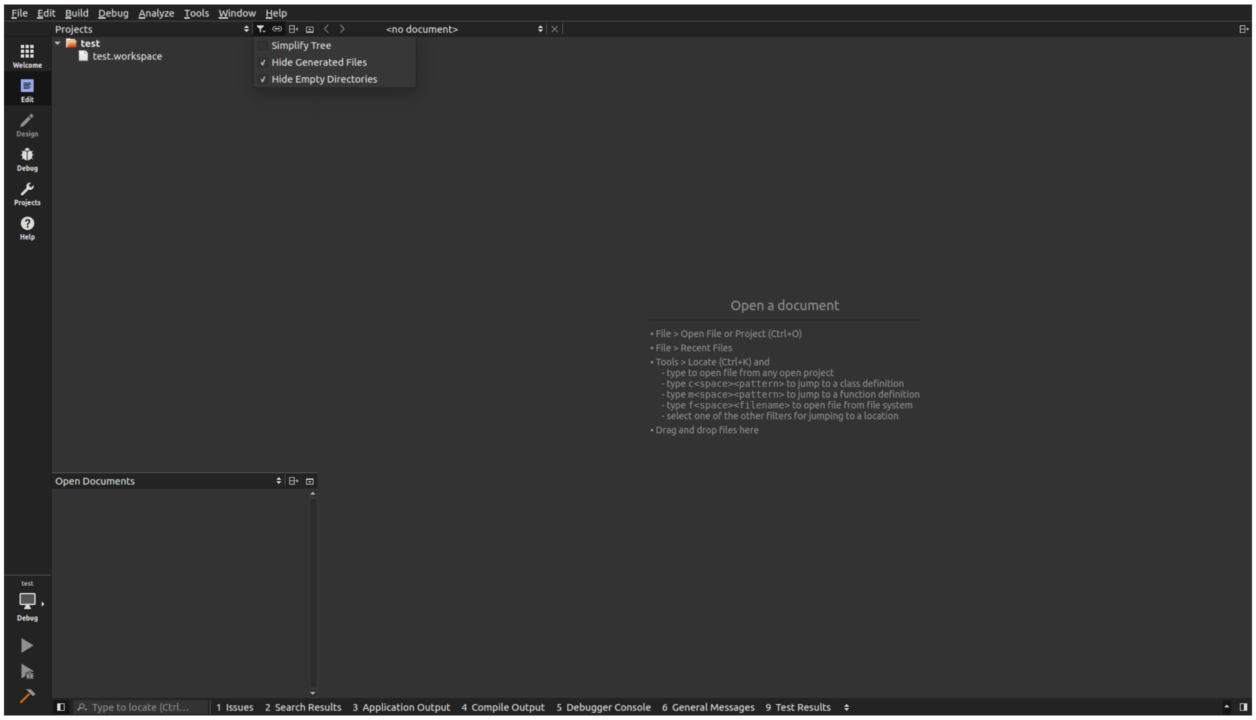
2.1 Frequently Asked Questions

This wiki highlights the frequently asked questions on the issue tracker.

1. *How to show empty folders?*
2. *Are new files automaticly added to the CMakeLists.txt file?*
3. *Warning: This file is not part of any project.?*
4. *Environment variables in .bashrc are not loaded by Qt Creator.*
5. *How do I set the C++ code style to match the ROS Style?*

2.1.1 How to show empty folders?

By default Qt Creator hides empty folders. In the above image you can see that the workspace **src** folder is not shown. Under the project filters uncheck **Hide Empty Directories**.



Open a document

- File > Open File or Project (Ctrl+O)
- File > Recent Files
- Tools > Locate (Ctrl+K) and
  - type to open file from any open project
  - type c<space><pattern> to jump to a class definition
  - type m<space><pattern> to jump to a function definition
  - type f<space><filename> to open file from file system
  - select one of the other filters for jumping to a location
- Drag and drop files here

2.1.2 Are new files automaticly added to the CMakeLists.txt file?

Currently it is the developers responsibility to edit the CMakeLists.txt file.

2.1.3 Warning: This file is not part of any project.?

This can be for several reason.

1. The workspace has not been built from within Qt Creator. Specific flags are passed to cmake which create specific project files which the IDE can parse to build a code model.
2. The file or files have not been added to the CMakeLists.txt file followed by a build from within Qt Creator.
3. Sometime is persist event if you address the previous items, but if code following is working then it should be ignored.

2.1.4 Environment variables in .bashrc are not loaded by Qt Creator.

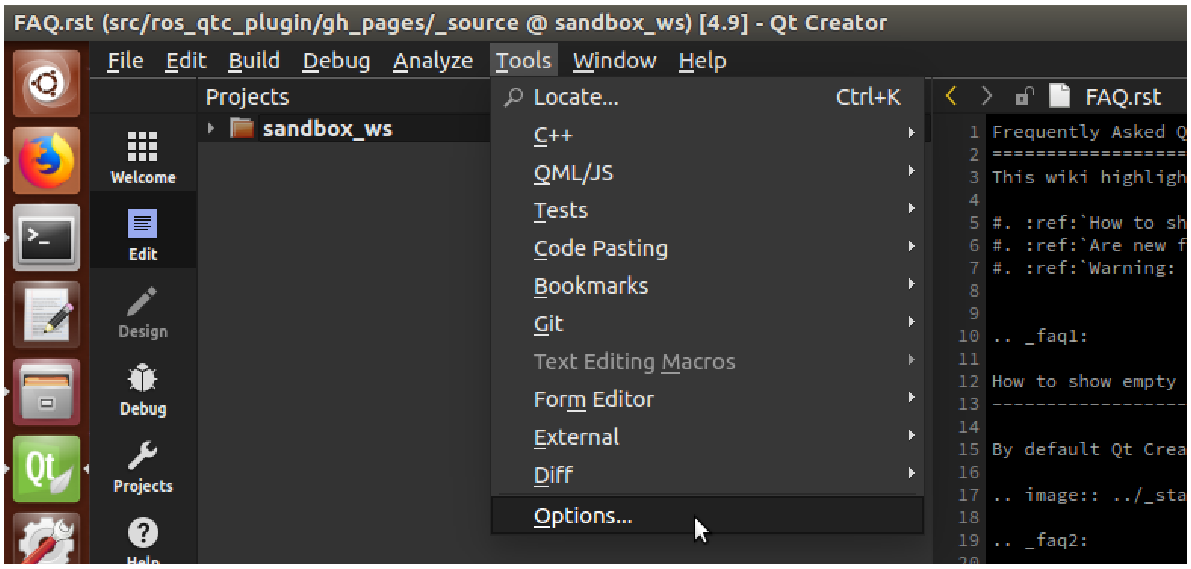
The .bashrc is loaded by bash terminals and is not used by applications. In order for applications to get user defined environment variables you have the two known option below.

1. The environment variables can be added to the build configuration in Qt Creator of the project. This will be required for every new project created in Qt Creator.
2. The environment variables can be added to the .profile file in the users home directory. This is only required once.

2.1.5 How do I set the C++ code style to match the ROS Style?

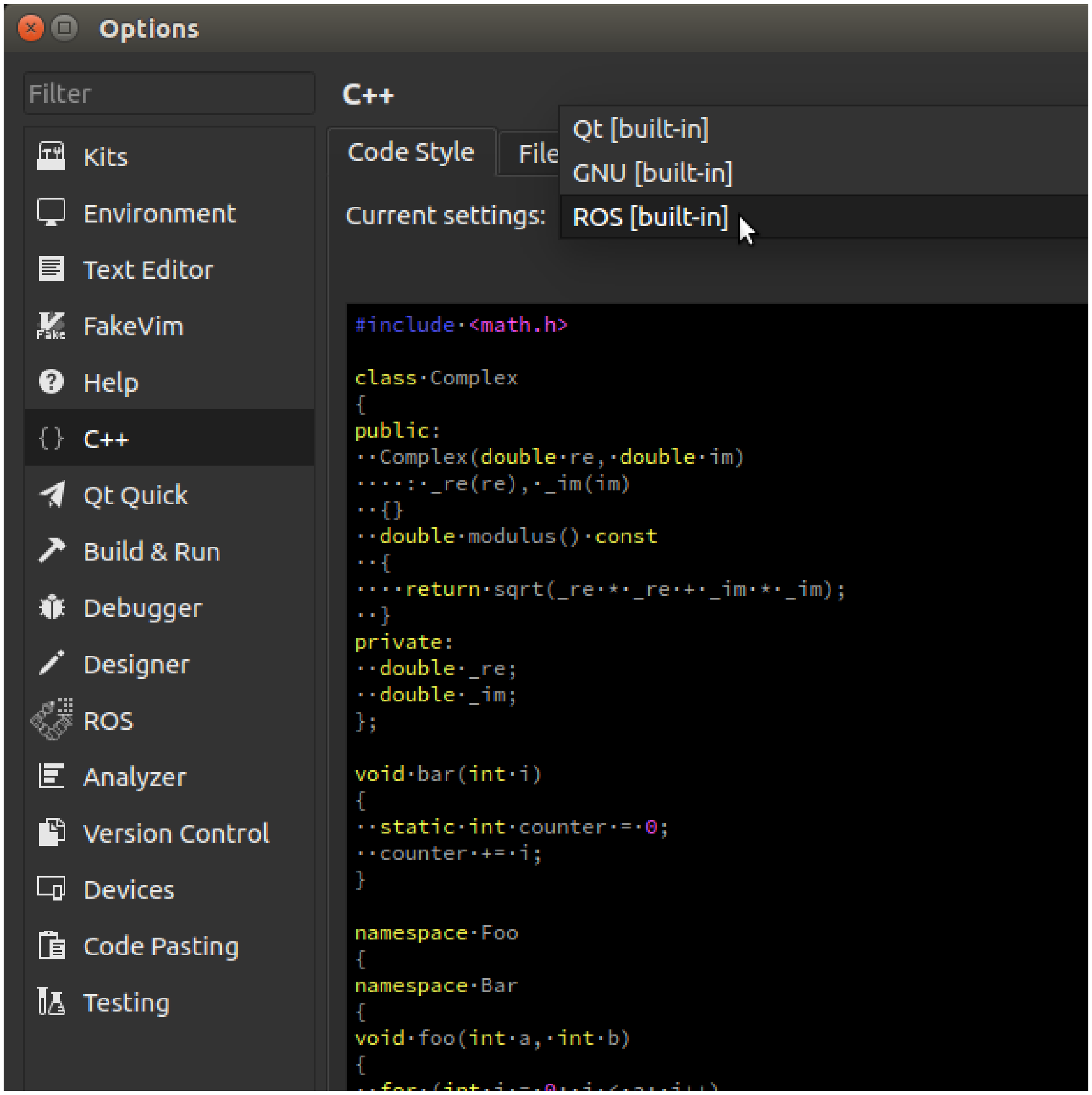
1. First, find the Tools tab on the upper toolbar.

2. From the Tools tab, select Options.



```
Frequently Asked Q
==================
This wiki highligh

#. :ref:`How to sh
#. :ref:`Are new f
#. :ref:`Warning:


.. _faq1:

How to show empty
-----------------

By default Qt Crea

.. image:: ../_sta

.. _faq2:
```

3. In the window that pops up, select C++ in the menu on the left.
4. Select the Code Style tab near the top of the pop-up window.
5. Select ROS from the drop down menu.



Current settings: ROS [built-in]

```
#include·<math.h>

class·Complex
{
public:
··Complex(double·re,·double·im)
····:·_re(re),·_im(im)
··{}
··double·modulus()·const
··{
····return·sqrt(_re·*·_re·+·_im·*·_im);
··}
private:
··double·_re;
··double·_im;
};

void·bar(int·i)
{
··static·int·counter·=·0;
··counter·+=·i;
}

namespace·Foo
{
namespace·Bar
{
void·foo(int·a,·int·b)
{
··for·(int·i·=·0;·i·<·a;·i++)
```

6. Select ‘OK’ at the bottom right to apply settings and return to coding!

CHAPTER 3

Users Help

3.1 Setup Qt Creator for ROS

3.1.1 Setup Ubuntu to allow debugging/ptrace

1. Open file: `sudo gedit /etc/sysctl.d/10-ptrace.conf`
2. Change the value of `kernel.yama.ptrace_scope` to 0
3. Reload the kernel configuration with `sudo systemctl restart procps.service`

3.1.2 Set Theme

If version 3.3.1 or higher was installed you are able to set the theme to dark following the steps bellow.

1. Open Qt Creator
2. Goto: `Tools` > `Options` > `Environment` > `General`
3. There should be a setting for Theme in the "User Interface" group containing a drop down box with two options "default or dark".

3.1.3 Set Syntax Color Schemes

1. Open Qt Creator
2. Goto: `Tools` > `Options` > `Text Editor` > `Font & Colors`
3. There is a drop down box in the "Color Scheme" group where you select different syntax color schemes.
4. Addition schemes can be added as explained in the below links.
   1. https://github.com/welkineins/qtcreator-themes
   2. https://github.com/alexpana/qt-creator-wombat-theme

3.1.4 Set ROS Code Format

1. Open Qt Creator
2. On the sidebar: `Projects` > `Editor`
   - Changing it globally at `Tools` > `Options` > `C++` or within `Projects` > `Code Style` does not work.

3.1.5 Setup Clang Formatting

1. Install Clang *sudo apt-get install clang-format-6.0*
2. Goto: `Tools` > `Options` > `Environment` > `External Tools`
3. Select: `Add` > `Add Tool`
4. Fill in the information below.
   - Description: Clang Cpp Format
   - Executable: /usr/bin/clang-format-6.0
   - Arguments:

```
-style="{Language: Cpp, AccessModifierOffset: -2, AlignAfterOpenBracket: true,
↪ AlignEscapedNewlinesLeft: false, AlignOperands:   true,␣
↪AlignTrailingComments: true, AllowAllParametersOfDeclarationOnNextLine:␣
↪true, AllowShortBlocksOnASingleLine: false,␣
↪AllowShortCaseLabelsOnASingleLine: false,␣
↪AllowShortIfStatementsOnASingleLine: false, AllowShortLoopsOnASingleLine:␣
↪false, AllowShortFunctionsOnASingleLine: All,␣
↪AlwaysBreakAfterDefinitionReturnType: false,␣
↪AlwaysBreakTemplateDeclarations: false, AlwaysBreakBeforeMultilineStrings:␣
↪false, BreakBeforeBinaryOperators: None, BreakBeforeTernaryOperators: true,␣
↪BreakConstructorInitializersBeforeComma: false, BinPackParameters: true,␣
↪BinPackArguments: true, ColumnLimit:     80,␣
↪ConstructorInitializerAllOnOneLineOrOnePerLine: false,␣
↪ConstructorInitializerIndentWidth: 4, DerivePointerAlignment: false,␣
↪ExperimentalAutoDetectBinPacking: false, IndentCaseLabels: false,␣
↪IndentWrappedFunctionNames: false, IndentFunctionDeclarationAfterType:␣
↪false, MaxEmptyLinesToKeep: 1, KeepEmptyLinesAtTheStartOfBlocks: true,␣
↪NamespaceIndentation: None, ObjCBlockIndentWidth: 2,␣
↪ObjCSpaceAfterProperty: false, ObjCSpaceBeforeProtocolList: true,␣
↪PenaltyBreakBeforeFirstCallParameter: 19, PenaltyBreakComment: 300,␣
↪PenaltyBreakString: 1000, PenaltyBreakFirstLessLess: 120,␣
↪PenaltyExcessCharacter: 1000000, PenaltyReturnTypeOnItsOwnLine: 60,␣
↪PointerAlignment: Left , SpacesBeforeTrailingComments: 1,␣
↪Cpp11BracedListStyle: true, Standard: Cpp11, IndentWidth: 2, TabWidth: 8,␣
↪UseTab: Never, BreakBeforeBraces: Allman, SpacesInParentheses: false,␣
↪SpacesInSquareBrackets: false, SpacesInAngles:  false,␣
↪SpaceInEmptyParentheses: false, SpacesInCStyleCastParentheses: false,␣
↪SpaceAfterCStyleCast: false, SpacesInContainerLiterals: true,␣
↪SpaceBeforeAssignmentOperators: true, ContinuationIndentWidth: 4,␣
↪CommentPragmas:  '^ IWYU pragma:', ForEachMacros:   [ foreach, Q_FOREACH,␣
↪BOOST_FOREACH ], SpaceBeforeParens: ControlStatements, DisableFormat:  ␣
↪false}" -i %{CurrentDocument:FilePath}
```

   - Working directory: %{CurrentProject:Path}
   - Output: Show in Pane

    - Error output: Show in Pane
    - Environment: No Changes to apply.
    - Modifies current document: Checked

5. Select `Apply`
6. Now lets add a quick key.
7. Goto: `Tools` > `Options` > `Environment` > `Keyboard`
8. In the filter box type "Clang" and you should pull up the new tool.
9. In the Shortcut section enter the Target text box and press `Ctrl + Shift + k` to set the shortcut.
10. Now to apply the Clang format to a C++ file open the file in Qt Creator and press `Ctrl + Shift + k` and the file should be formatted correctly.

3.1.6 Preventing Qt Creator form stepping into Boost, Eigen, etc.

1. First clone this repository https://github.com/Levi-Armstrong/gdb-7.7.1.git
2. Follow the instruction in the README file
    1. ./configure
    2. make
    3. sudo checkinstall
3. Goto: `Tools` > `Options` > `Debugger` > `GDB`
4. Add the following code below to the "Additional Startup Commands"

```
skip pending on
python
for root, dirs, files in os.walk("/usr/include/boost/"):
  for file in files:
    if file.endswith(".hpp"):
      cmd = "skip file " + os.path.join(root, file)
      gdb.execute(cmd, True)

for root, dirs, files in os.walk("/usr/include/eigen3/Eigen/"):
  for file in files:
    if file.endswith(".hpp"):
      cmd = "skip file " + os.path.join(root, file)
      gdb.execute(cmd, True)
end
skip enable
```

5. Now when you are stepping through your code it should not step into Boost or Eigen. You can also add additional directories following the same process.
6. Also if you would like to skip a particular function refer to the GDB documentation for instruction. It is something along the lines of *skip function function_name*.

3.2 How to Import a ROS Workspace

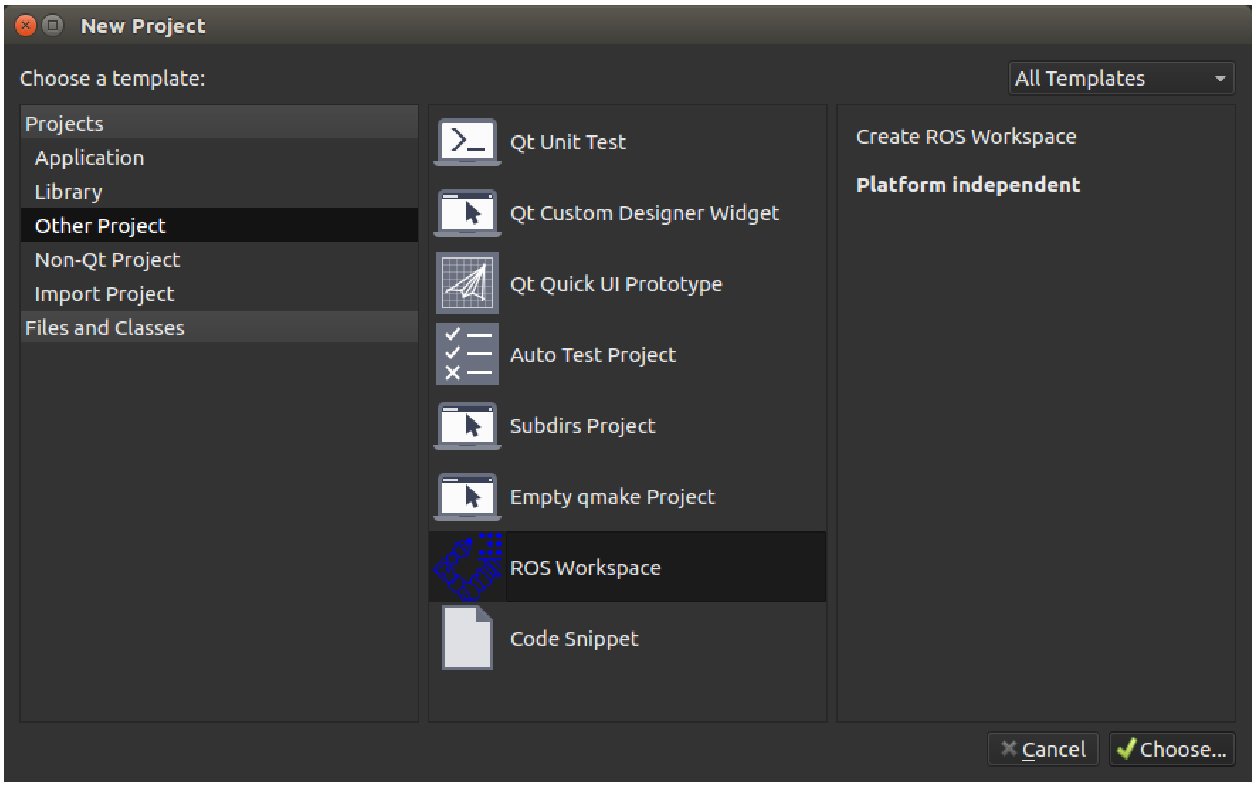
This wiki explains the procedure for importing a ROS Workspace.

Note: The pictures may not look identical based on version but the process flow is the same.

3.2.1 User Guide

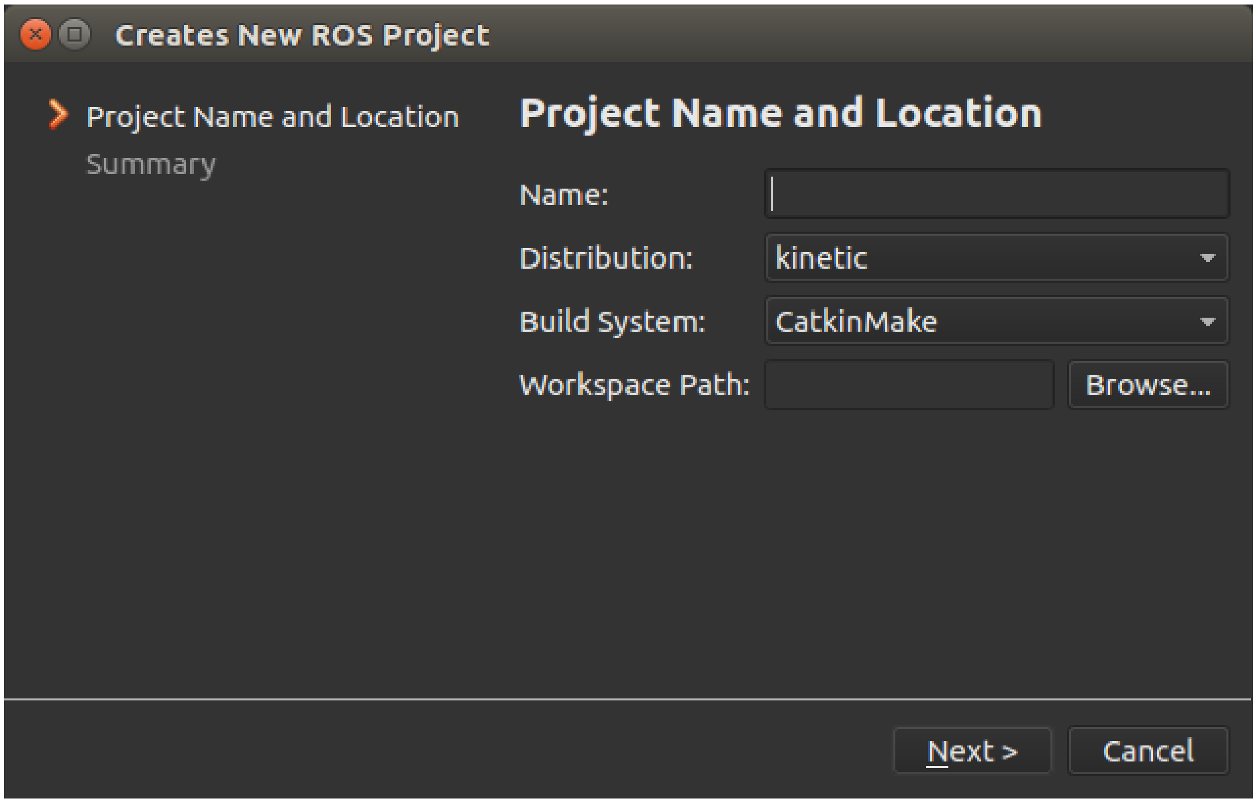
Step 1

On the **Welcome** screen select `New Project` and the following screen should show.



Step 2

Under Projects select `Other Projects` > `ROS Workspace` then select `Choose...` and the following screen should show.



Name:
Distribution: kinetic
Build System: CatkinMake
Workspace Path:

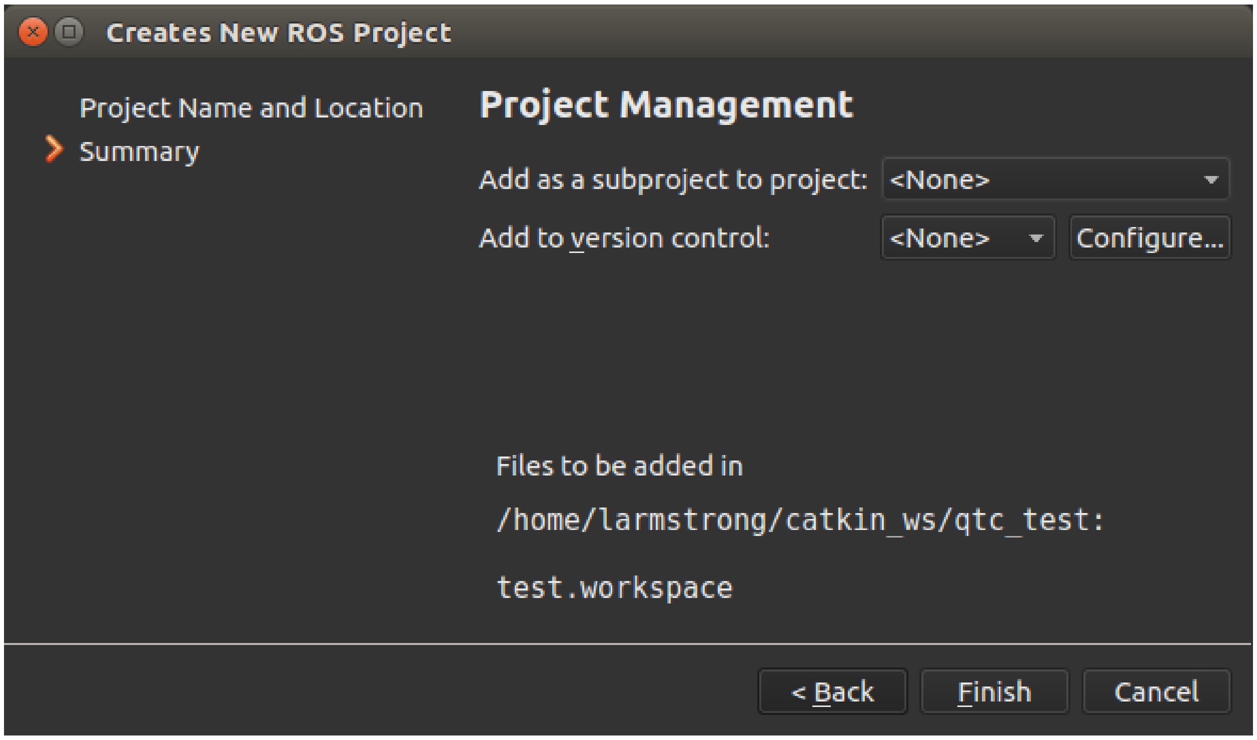
Step 3

Fill out the project information.

Field	Description
Name	Name of the project.
Distribution	The ROS Distribution (indigo, kinetic, etc.)
Build System	The desired build system.
Workspace Path	The path to the workspace folder.

Step 4

Select `Next` and the following screen should show and select `Finish` .



Project Management

Add as a subproject to project: <None>

Add to version control: <None>

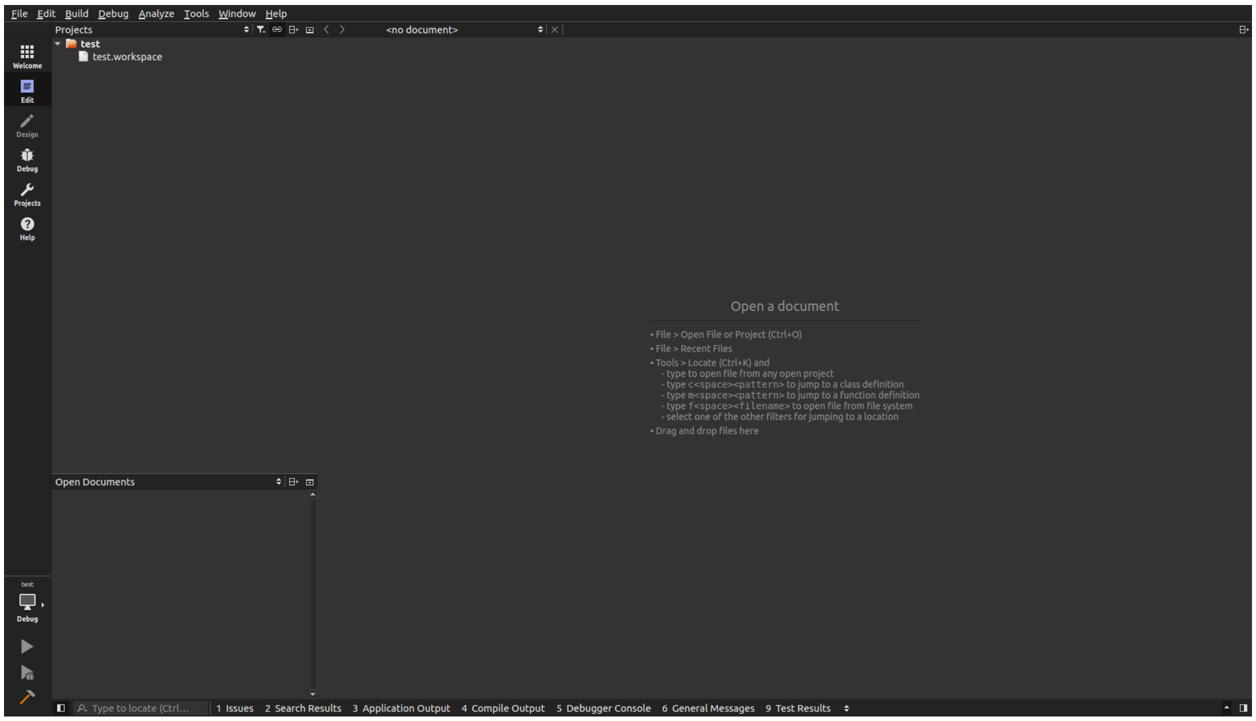
Files to be added in

/home/larmstrong/catkin_ws/qtc_test:

test.workspace

Step 5

The project should be open look similar to the screen below.

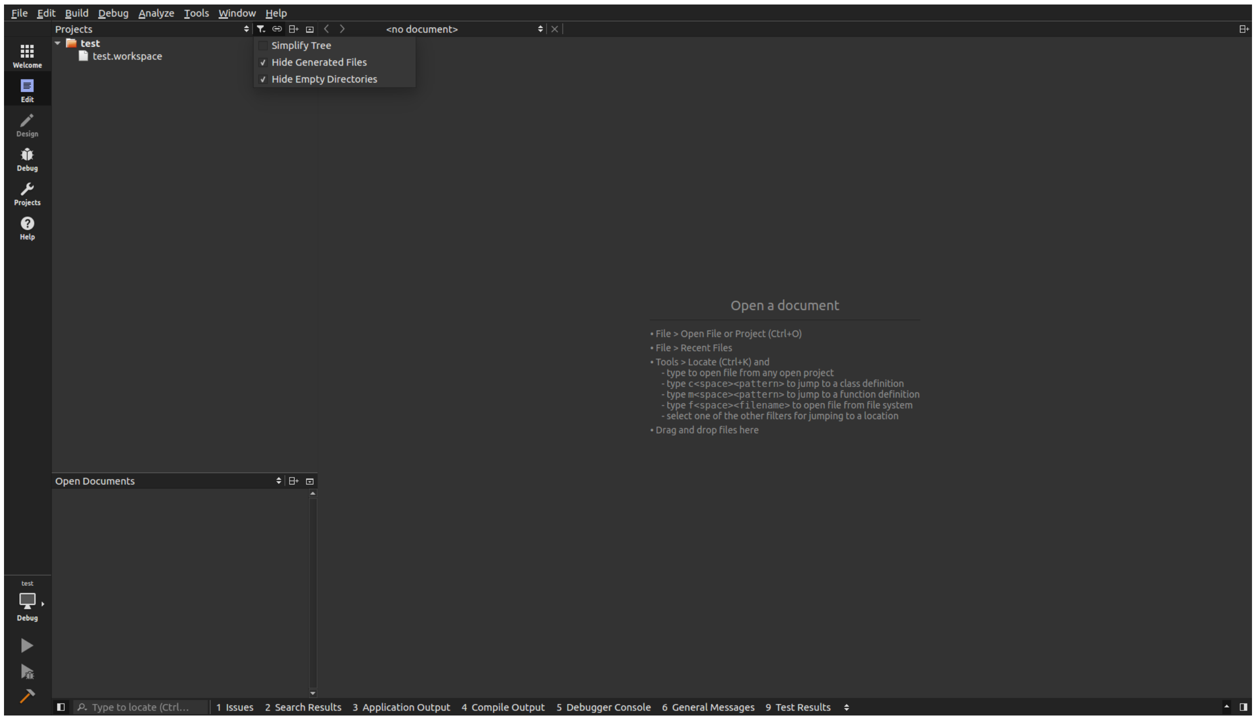

File Edit Build Debug Analyze Tools Window Help
Projects
<no document>
test
test.workspace
Welcome
Edit
Design
Debug
Projects
Help
Open Documents
test
Debug
Type to locate (Ctrl...
1 Issues 2 Search Results 3 Application Output 4 Compile Output 5 Debugger Console 6 General Messages 9 Test Results

Open a document

- File > Open File or Project (Ctrl+O)
- File > Recent Files
- Tools > Locate (Ctrl+K) and
  - type to open file from any open project
  - type c<space><pattern> to jump to a class definition
  - type m<space><pattern> to jump to a function definition
  - type f<space><filename> to open file from file system
  - select one of the other filters for jumping to a location
- Drag and drop files here

Step 6 (Optional)

By default Qt Creator hides empty folders. In the above image you can see that the workspace **src** folder is not shown. Under the project filters uncheck **Hide Empty Directories**.


File Edit Build Debug Analyze Tools Window Help
Projects
<no document>
test
test.workspace
Welcome
Edit
Design
Debug
Projects
Help
Open Documents
test
Debug
Type to locate (Ctrl...
1 Issues 2 Search Results 3 Application Output 4 Compile Output 5 Debugger Console 6 General Messages 9 Test Results

- [ ] Simplify Tree
- [✓] Hide Generated Files
- [✓] Hide Empty Directories

Open a document

- File > Open File or Project (Ctrl+O)
- File > Recent Files
- Tools > Locate (Ctrl+K) and
  - type to open file from any open project
  - type c<space><pattern> to jump to a class definition
  - type m<space><pattern> to jump to a function definition
  - type f<space><filename> to open file from file system
  - select one of the other filters for jumping to a location
- Drag and drop files here

3.3 Debugging Catkin Workspace

3.3.1 Prerequisite

1. Allow ptrace by following these instructions

3.3.2 Attach to a unstarted process

1. Next in Qt Creator browse to the file you wish to debug and insert break points.
2. `Menu Bar` > `Debug` > `Start Debugging` > `Attach to Unstarted Application...`
3. Browse to the executable then select `Start Watching`.
4. Now run your project. `Ctrl` + `R`
5. Now depending on where the breakpoints were placed in qt, it should be stopped at a break point when it reaches one.

3.3.3 Attach to a running process

1. Next in Qt Creator browse to the file you wish to debug and insert break points.
2. Now run your project. `Ctrl` + `R`

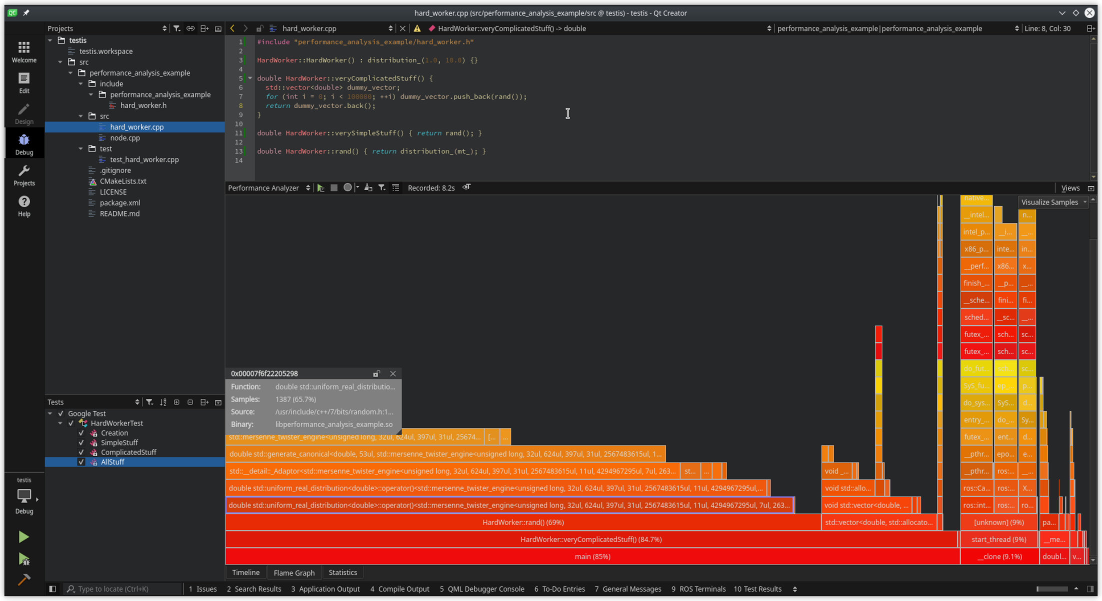
3. `Menu Bar` > `Debug` > `Start Debugging` > `Attach to Running Application...`
4. Now select the Process ID and then click the button `Attach to Process`.
5. Now depending on where the breakpoints were placed in qt, it should be stopped at a break point when it reaches one.

Note:

1. Sometime it will be paused at a ros::spin, so check after it has attached and if it is passed and not at a inserted breakpoint click the continue button in the debugger.
2. Some nodes can execute fast enough to where the debugger is not fully attach to the process before it reaches the first breakpoint, resulting in it never stopping at the breakpoint. To solve this add a *sleep(NumberOfSeconds)* command at the start of the node to allow enough time for the debugger to attach. Usually 3 seconds is enough but varies depending on the size of the node.

3.4 Perf and friends

There is a tool called perf that can collect data of the system. This can be targeted to collect data of your program and then parse that dump of data to be visualized. The parser is called perfparser from the Qt people and, along with it, they have also developed a plugin to visualize data directly in the IDE as shown in the image below. However this data can be visualized with other programs as the KDAB people show with the project Hotspot.


hard_worker.cpp (src/performance_analysis_example/src @ testis) - testis - Qt Creator
hard_worker.cpp
HardWorker::veryComplicatedStuff() -> double
Performance Analyzer
Recorded: 8.2s
HardWorker::rand() (69%)
HardWorker::veryComplicatedStuff() (84.7%)
main (85%)
Timeline
Flame Graph
Statistics

```
#include "performance_analysis_example/hard_worker.h"

HardWorker::HardWorker() : distribution_(1.0, 10.0) {}

double HardWorker::veryComplicatedStuff() {
  std::vector<double> dummy_vector;
  for (int i = 0; i < 100000; ++i) dummy_vector.push_back(rand());
  return dummy_vector.back();
}

double HardWorker::verySimpleStuff() { return rand(); }

double HardWorker::rand() { return distribution_(mt_); }
```

3.4.1 System setup

1. Install perf. In Ubuntu like systems:

```
sudo apt install linux-tools-common linux-tools-generic
```

2. Allow to run perf system wide by running. If is the first time, reboot:

```
echo -e "kernel.perf_event_paranoid=-1\nkernel.kptr_restrict=0" | sudo tee /etc/
↪sysctl.d/10-perf.conf
```

If by running *cat /proc/sys/kernel/kptr_restrict* the result is 0, the permissions are set right.

3.4.2 QtCreator Setup

1. Install Qt Creator with the ROS plugin by dowloading the latest version.
2. Setup the workspace in QtCreator
3. Compile (Shortcut with Ctrl+B)
4. Add the custom executable. With catkin, all executables are outputed under *<workspace>/devel/lib/<my_package>*. We will create a configuration in QtCreator so it can use it. We will rename the custom executable to make it easier to find and use afterwards.

3.4.3 Performance analysis

1. Go to the debug view.
2. Select the custom executable created before from the bottom left menu.
3. Change the tool to **Performance Analyzer**
4. Start the debug session with the play button. The program should run.
5. Start recording data by pressing the grey circle.
6. Stop recording data by pressing the red circle again. Red means that is recording.
7. Stop the node (if you want to, this step assumes free will).
8. Et voila! You can visualize your data, a very useful view is the *Flame graph*.

In the gif below it can be seen how the *veryComplicatedStuff* function is taking 85% of the execution of the process.

3.4.4 Bonus: Running tests from QtCreator

Qt Creator also supports Google Tests natively and you can, not only run an specific test from the IDE but also debug it!. To do so:

1. Compile the tests with *catkin run_tests <my_package>*. This will generate a binary with the tests.
2. Add the custom executable. As before, create a custom executable but, this time, the target should be the previously generated binary.
3. Go the Tests view and make use of it.
4. When running the tests, it will ask about which *Custom executable* to use. Select the one created right now.

CHAPTER 4

Tutorial Videos

CHAPTER 5

Developers Help

5.1 Where to find Qt Creator Plug-in Support

There are three reliable ways to get into contact:

1. There is the bug tracker on bugreports.qt.io. People here are very aware of everything entered there, but may not react immediately. But this is the best place for feature requests, bug reports and similar things that should not be forgotten.
2. Our mailing list is actively read by all developers. That is available here which should get most questions answered.
3. The fastest option is IRC: #qt-creator on the freenode network. All developers hang out there, at least while they are in the office. So European business hours tend to work best.

Note: This link provides a good overview of names to ping for specific questions.

CHAPTER 6

Old Tutorials

1. How to create a catkin workspace.
2. How to configure the run and build settings.

Note: Videos may not always be up-to-date so if you are not sure about something please refer to the *User Help* section.