# RHX

# Application Note: Building RHX Source Code

111001010001011100

28 February 2022

The RHX code is built with a C++ framework called Qt, and while the initial download and installation of Qt can take some time, it is necessary for developers who wish to modify the Intan RHX software (or any of the Intan legacy programs that similarly are built with Qt). Qt is a free framework that allows for cross-platform (Windows, Mac, and Linux) programs with useful features like UI, multi-threading, and much more. This document aims to assist developers in setting up (1) the Qt Creator IDE, (2) the install of the Qt framework itself (and an accompanying C++ compiler), and (3) the steps to properly build the Intan RHX project with Qt.

While the examples in this document use Windows, Qt is cross-platform and analogous steps can be taken on Mac and Linux. The end of this document includes a section on platform-specific concerns for non-Windows builds.

Qt and Qt Creator can be downloaded from the Qt website: <u>https://www.qt.io/product/development-tools</u>. By selecting 'Download Qt', 'Go open source', and 'Download the Qt Online Installer' you can download Qt's installer program that will then allow you to install the specific Qt features you need. This will you require you to create a free Qt account.

- 1. <u>Qt Creator</u>. While it is possible to compile through other IDEs, we recommend you use the (free) IDE, Qt Creator, available from the Qt Online Installer. The exact version of Qt Creator is not that important; you should probably use the most up-to-date release available. At the time of writing this document, we are using version 4.14.0. Note that the version of Qt Creator is not related to the version of Qt that a project is built with, which can be much more relevant. With a single version of Qt Creator, developers can select several different versions of Qt to build a project with. Once Qt Creator has been downloaded and installed, you can double-check the version you have with 'Help' -> 'About Qt Creator...'.
- 2. <u>Qt/C++</u>. You'll need the Qt framework itself, which can also be downloaded through the Qt Online Installer. Qt has a lot of different versions, some with significant changes, so it's important to get a version close to what we have used to develop the Intan RHX software. We currently use Qt 5.12.8 for RHX compilation, and we compile RHX as a 64-bit binary. While you could likely use different Qt versions successfully, we recommend you install the 64-bit 5.12.8 version. Note that in the Qt installer, you can click the 'Archive' checkbox and re-run the filter to get a more complete list of available Qt versions.

You'll also need a C++ compiler to link your Qt version to make up a 'kit'. For Windows, we use MSVC 2017 64-bit. The full name is Microsoft Visual C++ Compiler 16.5.30011.22 (x86\_amd64). You can probably use a different compiler with some minor changes, but for consistency we recommend using the same compiler if possible.

3. <u>Build Intan RHX Software</u>. Finally, using Qt Creator, we link the Qt version with our C++ compiler to create a kit, and then use that kit to build the IntanRHX project. The first step is to open the IntanRHX.pro file, with 'File' -> 'Open File or Project...'. Click the 'Projects' tab on the left, then click on 'Manage Kits' to bring up a dialog to set up your configuration. If the auto-detect feature works with no problems, you should see 'Desktop Qt 5.12.8 MSVC2017 64bit' listed under the auto-detect section of Kits. (Don't worry if the other listed Qt kits vary from the picture, as this will change depending on what versions of Qt and C++ compilers you have installed.)



QC Options - Qt Creator						×			
Filter	Kits								
🖽 Kits	Kits Qt Versions Com	pilers Debuggers	CMake						
Environment	Name					Add			
- Text Editor	<ul> <li>Auto-detected</li> <li>A Desktop Qt 5.12</li> </ul>	.8 MSVC2015 64bit				Clone			
	Desktop Qt 5.12	Desktop Qt 5.12.8 MSVC2017 32bit     Remove							
Facevim	Desktop Qt 5.12.8 MinGW 32-bit     Desktop Qt 5.12.8 MSVC2017 64bit     Desktop Qt 5.12.8 MSVC2017 64bit								
🕐 Help	A Desktop Qt 5.14	A Desktop Qt 5.12.45 Million Or 15 debit			Desktop Local PC				
{} C++	Desktop Qt 5.14	2 MSVC2017 52bit		Sys Root:	Locarre				
🖪 Qt Quick	Desktop Qt 5.14	.2 MinGW 32-bit		Compiler:	Microsoft Visual C++ (x86_amd64)	Compiler 16.5.30011.22			
➤ Build & Run	Qt 5.12.8 for UN	VP 32bit (MSVC 2017) VP 64bit (MSVC 2015)		Environment:	CDB Engine using "C	Program Files (v86)\Windows			
fff Obs	Qt 5.12.8 for UN	VP 64bit (MSVC 2017) VP ARMv7 (MSVC 2015	)	bebuggen.	Kits\10\Debuggers\x6	54\cdb.exe"			
	A Qt 5.12.8 for UV	VP ARMv7 (MSVC 2017 ssembly (default)		Qt version: mkspec:	Qt 5.12.8 MSVC2017 t	94bit			
the Debugger	Qt 5.14.2 for U	VP 32bit (MSVC 2017)		Additional Qbs Profile Setti	ngs: Surtem (Make at Cill	Program			
Designer	Qt 5.14.2 for UV	VP 64bit (MSVC 2017)		Cmake.	Files\CMake\bin\cma	ake.exe			
Python	A Qt 5.14.2 for UN	VP ARMV7 (MSVC 2015 VP ARMV7 (MSVC 2017	)	CMake Generator:	Generator: Ninja Extra generator:				
E Analyzer	✓ Manual Namo.	Dealstee Ot % (Ota)	MCUC2017 6 dhit	CMake Configuration:	CMAKE_CXX_COMPII {Compiler:Executable	LER:STRING=% :Cxx}			
P Version Control		Desktop Qt %(Qt.vi	ensionly misve 2017 ondit		CMAKE_C_COMPILER	R:STRING=%{Compiler:Executable:C}			
D Devices	File system name: OT_OMAKE_XEC				QT_QMAKE_EXECUTA	BLE:STRING=%			
-	Device type:				(Quqmaketxecutable	1			
	Device:	Local PC (detault to	r Desktop)			Pranage			
Language Client	Sysroot:					Browse			
🛃 Testing	Compiler:	C: Microsoft Visi	Jal C++ Compiler 16.5.30011.22 (x86_amd64)			Manage			
		C++: Microsoft Visi	C++: Microsoft Visual C++ Compiler 16.5.30011.22 (x86_amd64)						
	Environment:	Force UTF-8 MSV	C compiler output			Change			
	Debugger:	Auto-detected CDB	at C:\Program Files (x86)\Windows Kits\10\Debuggers\x64\cdb.ex	xe		Manage			
	Qt version:		7 64bit			Manage			
	Qt mkspec:								
	Additional Qbs Profile Settings					Change			
	CMake Tool:	System CMake at C	\Program Files\CMake\bin\cmake.exe			Manage			
	CMake generator:	<none> - Ninja, Plat</none>	form: <none>, Toolset: <none></none></none>			Change			
	CMake Configuration:	CMAKE_CXX_COMPI	.ER:STRING=%{Compiler:Executable:Cxx}; CMAKE_C_COMPILER	l:STRING=%{Compiler:Executable:C};	CMAKE_PREFIX_PATH:S	Change			
					OK Cancel	Apply			



If you don't see this yet, make sure your Qt Versions looks something like this:





#### And your Compilers looks something like this:

💶 Options - Qt Creator		×
Filter	Kits	
🖽 Kits	Kits Qt Versions Compilers Debuggers CMake	
D Environment	Name Type	Add 👻
Text Editor	Default LLVM 64 bit based on MSVC2019 Clang	Clone
🌠 FakeVim	Clang (c, xoo odoit in c; toch loois/Qcreator/loin/clang/bin) Clang Microsoft Visual C++ Compiler 15.9:28307.1585 (x86) MSVC	Remove
? Help	Microsoft Visual c++ Compiler 15.9:28307.1580 (amde4_x80) MSVC Microsoft Visual C++ Compiler 15.9:28307.1580 (amde4) MSVC	Remove All
{} C++	Microsoft Visual C++ Compiler 15.9/2830/11363 (x80_amd04) MSVC	Auto-detection Settings
	MinGW 7.3.0 52-010 T0F C++ MinGW MinGW 7.3.0 64-bit for C++ MinGW	Auto-detection settings
	Microsoft Visual C++ Compiler 16.530011.22 (x86) Microsoft Visual C++ Compiler 16.530011.22 (amd64_x86) MSVC	
	Microsoft Visual C++ Compiler 10:53001122 (x86_amd64) MSVC	
Qbs	Microsoft Visual C++ Compiler 10.0 (x86) Microsoft Visual C++ Compiler 12.0 (x86) MSVC MSVC MSVC ABI: x86-windows- msv2019-ne-64bit	
t Debugger	Microsoft Visual C++ Compiler 12.0 (ambd) MSVC Microsoft Visual C++ Compiler 12.0 (ambd4) MSVC	
Designer	Microsoft Visual C++ Compiler 12.0 (x86 arm) MSVC Microsoft Visual C++ Compiler 12.0 (x86 arm) MSVC	
n Python	Default LIVM 64 bit based on MSVC2019 Clang	
Analyzer	Microsoft Visual C++ Compiler 15.328307.158 (am64,x86) MSVC	
P Version Control	With Soft Visual C++ Complicit 15/3/2000/1305 (and C+)	
Devices		
💼 Code Pasting	Name: Microsoft Visual C++ Compiler 16.5.30011.22 (x86_amd64)	
⊖ <sup>∰</sup> Language Client	Microsoft Visual C++ Compiler 16.5.30011.22 (x86_amd64)	
🛃 Testing	ABI: v86-windows-msur2019-ne-6dhit v v86 v - windows v - msur2019 v - ne v - 6dhit v	
	ок	Cancel Apply



While there are options to manually add your own Qt versions and compilers, we've only had to do that rarely. If your installed compilers don't show up from the auto-detect, then something is probably wrong. You might need to point Qt Creator to where the correct 'qmake.exe' file for your given Qt version is located. Once you have a valid kit set up, just make sure that it is active in the 'Projects' tab:

Build	& Run				Separate debug in	nfo:
	Desktop Ot 5.12.8 MSVC2015 64bit					
Ğ	Desktop Ot 5.12.8 MSVC2017 32bit	QML debugging ar	nd profiling:			
Ē	Desktop QL 5/12.8 MSVC2017 64bit					
_	> Build					
	▶ Run	Decktory Of 5 12 8 MSVG2	017 6464		amake svstem∩ b	ehavior when parsing:
- -	Desktop Qt 5.12.8 MinGW 32-bit	Desktop Qt 5.12.0 MSVC2	Deskton			
- -	Desktop Qt 5.12.8 MinGW 64-bit	Device:	Local PC			
	Desktop Qt 5.14.2 MSVC2015 64bit	Sys Root:				
	Desktop Qt 5.14.2 MSVC2017 32bit	sktop Qt 5.14.2 MSVC2017 32bit Compiler: Microsoft Visual C+		mpiler 16.5.30011.22		xe IntanRHX.pro
	Desktop Qt 5.14.2 MSVC2017 64bit	Environment:	(xoo_amao+)			C: \Users \afoy \Desktop
	Desktop Qt 5.14.2 MinGW 32-bit	Debugger:	CDB Engine using "C:\Prog	ram Files (x86	)\Windows	
	Desktop Qt 5.14.2 MinGW 64-bit	0	Kits\10\Debuggers\x64\cdb.exe"			
	Qt 4.8.6 MSVC 32	Qt version:	Qt 5.12.8 MSVC2017 64bit			
÷	Qt 5.12.8 for UWP 32bit (MSVC 2017)	Additional Qbs Profile Settings				
Ð	Qt 5.12.8 for UWP 64bit (MSVC 2015)	CMake:	System CMake at C:\Progra	m		ean in C:\Users\afov\De
÷	Qt 5.12.8 for UWP 64bit (MSVC 2017)	CM-la Caracter	Files\CMake\bin\cmake.ex	e		
- 15	Qt 5.12.8 for UWP ARMv7 (MSVC 2015)	CMake Generator:	Extra generator:			
- 15	Qt 5.12.8 for UWP ARMv7 (MSVC 2017)	CMake Configuration:	CMAKE_CXX_COMPILER:ST	RING=%		
10	Qt 5.14.2 WebAssembly		{Compiler:Executable:Cxx}		ent	
÷	Qt 5.14.2 for UWP 32bit (MSVC 2017)		CMAKE_C_COMPILER:STRI	PATH:STRING=%{Ot:QT_INSTALL_PREFIX} vironment		vironment
Ð	Qt 5.14.2 for UWP 64bit (MSVC 2015)		QT_QMAKE_EXECUTABLE:S	TRING=%		
Ð	Qt 5.14.2 for UWP 64bit (MSVC 2017)		{Qt:qmakeExecutable}			ironment
15	Qt 5.14.2 for UWP ARMv7 (MSVC 2015)					Darcorc
- 15	Qt 5.14.2 for UWP ARMv7 (MSVC 2017)				custom outpu	L Paisers
	Qt4_32_MSVC				There are no cust	om parsers active
	Qt4_32_minGW					
•	Replacement for "Desktop"					
Proje	ct Settings					

As long as that is active, you should be able to click 'Edit' and begin to navigate the IntanRHX project and edit C++ files at will, and then select either 'Debug' or 'Release' above the green Run button at the bottom left. You should then be able to click the green Run button to start the code building and running. Note that the first build is likely to take a long time.

Note that at runtime, the IntanRHX software looks in the executable directory for various files, including 'kernel.cl', 'ConfigRHDController.bit', 'ConfigRHDInterfaceBoard.bit', 'ConfigRHSController.bit', 'ConfigXEM6010Tester.bit', 'USBEvaluationBoard.bit', and 'okFrontPanel.dll'.



#### **Platform-Specific Concerns**

<u>Mac</u>: Instead of using a Qt 5.12.8 MSVC2017 64bit kit, we've used Qt 5.12.10 with the 'Clang (C++, x86 64bit in /usr/bin)' compiler. Also, instead of looking for 'okFrontPanel.dll', the file 'libokFrontPanel.dylib' is needed. Depending on which stage of the build/deploy process you're on, you may need this file to be alongside the IntanRHX executable (inside the IntanRHX.app -> Contents -> MacOS directory), or in Frameworks (inside the IntanRHX.app -> Contents -> Frameworks directory).

Linux: We've had success only when building Qt statically. The details of how to build Qt statically on Linux is beyond the scope of this document, but ultimately the command that worked for us was:

```
./configure -static -prefix ~/qt-build-dir -platform linux-g++ -xcb-xlib
-fontconfig -system-freetype
```

Then,

make

#### followed by

make install

Once Qt is built, build IntanRHX statically from the directory that contains the .pro file:

```
PATH=~/qt-build-dir/bin:$PATH
export PATH
qmake -config release
make
```

After this, an IntanRHX binary should be built. Make sure to run this with libokFrontPanel.so and rules.d in the same directory, and you should be able to run it.

