QDC-09-04-001 V1.0.2 (2017-11-01)

User Guide

Isomet Modular Synthesiser (iMS)

Getting Started Developing iMS Applications for the QNX Neutrino RTOS

Contents

1 Introduction

1.1 Background

The Isomet Modular Synthesiser (iMS) System is a suite of hardware and software components that allow any developer who is integrating Acousto-Optic (AO) components into their system to rapidly design an RF driver to control the AO with precision and sophistication.

The system comprises a number of different hardware devices with differing capabilities that are intended to generate one or more RF signals which, when suitably amplified, may generate the controlled deflection or modulation of a laser when directed through an AO device. The driver operation may consist of a single tone at its simplest through to a sequence of alternating “Images” at its most complex.

At its heart, the hardware consists of a Direct Digital Synthesis (DDS) engine coupled to some sophisticated electronics with large memory storage capability and substantial options for conditioning of the generated RF signal in various ways. The hardware also contains supporting circuitry for creating and reading auxiliary signals and monitoring the health and status of the amplifier and AO devices.

At Isomet, we have spent many years developing and refining both the hardware and the communications interfaces that control it. While some aspects of operation may be controlled through simple discrete I/O signals, to get the most out of the system requires physically connecting the system to a host computer system and running application software to perform some kind of application specific task. There are a number of scenarios in which this may be accomplished:

1. By connecting the iMS device to a small embedded system such as a microcontroller, typically using RS422 / RS485 serial communications. The microcontroller will usually be running bare-metal, i.e. without an operating system, and will issue and receive the discrete binary communications messages that pass back and forth between the iMS device and the host.

   This mode of operation might be appropriate for very simple applications in which the iMS is configured to generate basic tones in response to some external system event. Alternatively, the microcontroller might perform some specific function such as monitoring system health while a different host is performing more complex operations on one of the other interfaces.

   This mode is not recommended for more complex scenarios involving Image or Sequence transfer as the binary protocol between the devices would become unwieldy, requiring the specific ordering of some messages to avoid unpredictable results. In such cases, one should consider one of the following options.

2. By connecting the iMS device to a host PC, Server or Workstation running a compatible operating system such as Microsoft Windows®, typically using USB communications. Isomet supply free of charge a Software Development Kit (SDK) containing libraries in a number of formats and application software to speed the development and integration of the AO driver into the encompassing system.

   Users may use the included application and example software either as a starting point to explore the capabilities of the system or in its own right as dedicated control software. If they should choose to then write their own custom drive software, there is extensive software support in the included libraries to allow them to do so in one of a number of languages using a well defined Application Programmers' Interface (API) and easy to understand function calls such as, for example, `ImagePlayer.Play()`.

   The library code is thoroughly tested and abstracts away from the user's software all the details of implementation and messaging requirements. It also includes a Hardware Abstraction Layer (HAL) which means the same application software will work irrespective of whether connection is made to the iMS through USB, Ethernet, RS422 or any other interface. Additionally, the library performs all error detection and handling, including such concepts as CRC insertion and checking, messaging timeouts and interface health status / device watchdog capability.

3. By connecting the iMS device to an embedded host system running a real-time operating system such as Linux or QNX Neutrino. This is a hybrid option of the above two and might typically consist of an embedded computing platform or some custom hardware containing a System-on-Chip (SoC) running an ARM processing core. This might be appropriate to an industrial application requiring more capability than could be provided through a simple microcontroller connection.

   Isomet can and do supply versions of our SDK tailored for specific embedded configurations where such an implementation does not yet exist. In this case, the provided SDK implementation will consist of the core C++ library code that forms the heart of the SDK as provided for host PC systems, compiled into a shared object or similar dynamic library. We will provide example code and documentation to support the user development and test the library functionality, but typically will not include the extra "bells and whistles" that can be expected from the host computing versions such as GUI software or additional language support, unless specifically requested to do so.

   In providing these embedded SDK options, we use the same C++ code base that forms the core library code of the host PC SDK, and strive to ensure 100% or near-100% portability between them such that application software developed on, for example, a Microsoft Windows® platform can be expected to work without modification on, for example, a QNX Neutrino platform. Where differences exist or alternate behaviour is expected, we will endeavour to ensure these are documented and highlighted.

The rest of this document focuses on option 3 above, as applied to a QNX Neutrino RTOS platform and aims to act as a guide to getting started with iMS development when targetting a QNX system.

2 System Requirements

2.1 Isomet iMS SDK Version

At the time this document was written, the latest version of the Isomet iMS SDK available was v1.4.2.

This is the version that is referred to in this document and has been compiled for QNX Neutrino according to the conventions described herein. Future releases of the SDK may also be compiled for QNX Neutrino but should not be assumed to conform to the same descriptions unless explicity stated as so.

If in doubt, please speak to your usual contact at Isomet for clarification.

2.2 QNX Neutrino Development Environment

2.2.1 Version

The QNX Neutrino version of the Isomet iMS SDK has been developed using the QNX Software Development Platform (SDP) v6.6.0.

2.2.2 Target Platforms

The SDK deliverables include shared library object and example applications compiled for both X86 (Intel compatible) and ARMLE-V7 (Little Endian ARMv7 instruction set). Both are 32-bit only.

2.2.3 Test Setup

We developed and compiled the SDK and application software using the QNX Momentics IDE. This is an Eclipse based environment familiar to most embedded software developers and offers a simple to use interface. We encourage you to begin your iMS development using Momentics as you can import our example project to get a feel for how our software environment should be structured. Should you wish to switch to a Makefile based command line flow subsequently, we see no reason that it shouldn't work with our environment.

Our development host machine runs Ubuntu 16.04LTS.

We used two target systems running QNX Neutrino RTOS to test the compiled library code and application software. One uses the QNX Neutrino X86 VM image that is downloadable from the QNX user account centre. This was installed in an instance of a Oracle Virtualbox Virtual Machine. The second is a pre-compiled image for the TI AM335x ARM processor running on a Beaglebone Black Single Board Computer, available from here:

http://community.qnx.com/sf/wiki/do/viewPage/projects.bsp/wiki/TiAm335Beaglebone

Both were installed with little issue and minimal modification. On the Beaglebone, we established a terminal prompt using LVTTL-RS232 level shifters on the 6-pin header J1. We also needed to start `qconn` at boot by including it in `/etc/rc.d/startup_aps.sh` to enable a connection to be made to the Momentics System Information Perspective.

For the VM install, we initially used Bridged networking mode and allowed the VM to receive its own IP address from the network DHCP server. However, when probing the Ethernet communications between the VM and the SDK application software using Wireshark, we discovered that this led to duplication of messages which could ultimately lead to failed communications. We switched to NAT (Network Address Translation) mode which resolved this issue although be aware that Port Forwarding must be set up to enable port 8000/TCP and allow

Momentics to communicate with the qconn server on the VM. Note that Ethernet communications using the iMS SDK uses ports in the range of 28241 to 28245 although port forwarding should not be necessary.

2.2.4 Licensing

All of our QNX development work has been performed with the generous support and cooperation of Blackberry QNX under a Partner Software License Agreement (PSLA). It is your responsibility to ensure that you have an appropriate license agreement in place to develop systems that include or make use of our SDK and associated software in your QNX based end systems.

3 Getting Started

3.1 Deliverables

Once you have obtained the iMS SDK for QNX Neutrino from the Isomet website, unzip and untar it to a location of your choosing.

The SDK includes:

- Shared and Static library objects for ARMv7LE and X86 target platforms
- C++ header files for including in your application software
- A console test application that uses a simple menu / submenu structure to perform a variety of example operations on a connected iMS system
- Full API documentation in HTML and PDF form
- Source code for the console test application
- An archived Momentics IDE project for compiling the console test applications
- This Getting started guide

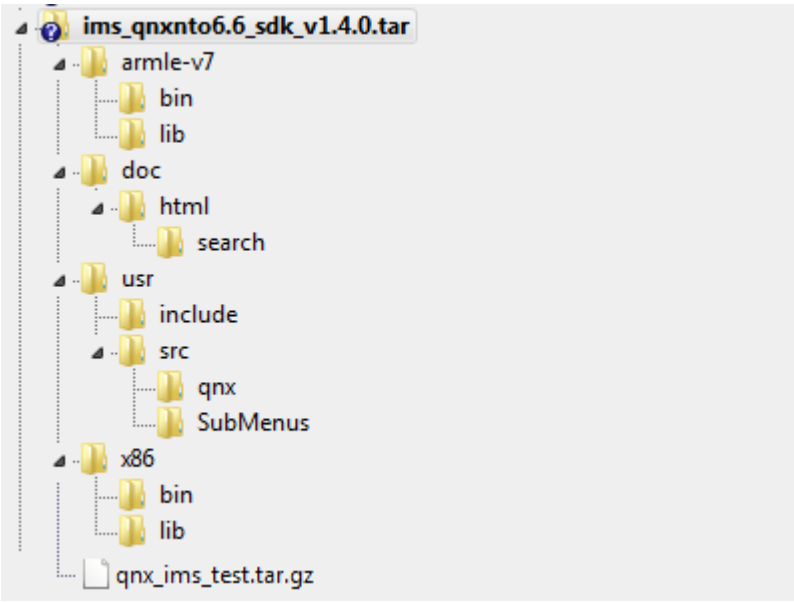

Figure 1: QNX SDK Directory Structure

armle-v7/		Compiled objects for ARMv7 LE platform
|	bin/	Console Test Application executable
|	lib/	Shared (.so) and Static (.a) library files
doc/		PDF reference manual for API and this Getting Started guide
|	html/	HTML reference manual. Open index.html to view.
usr/		Application and library source code

|	include/	C++ header files for API
|	src/	Example test application source code
|	src/qnx/	Test application top level source file
|	src/SubMenus	Source files for lower level menus in test application, each performing a different function associated with typical operations performed on an iMS System
x86		Compiled objects for X86 platform
|	bin/	Console test application executable
|	lib/	Shared (.so) and Static (.a) library files
qnx_ims_test.tar.gz		Archived Momentics IDE project for Console Test Application

3.2 Installing on the QNX Target Platform

Once you have a QNX target platform set up and running, try installing the example test application.

Using your chosen method of file transfer (you could use the QNX System Information perspective Filesystem Explorer in Momentics, or mount a CIFS partition from the target, for example), copy the `qnx_ims_test` executable from the `bin/` subfolder of the relevant platform architecture (x86 or armv7le) to a location of your choice on the target.

Also copy the `libims.so` object from the `lib/` subfolder to the same location on the target. Log in to the target system and rename the object to `libims.so.1`. Then create a symbolic link to the renamed object as follows:

```
ln -s libims.so.1 libims.so
```

and either move both files into `/usr/lib` so that they are visible on the linker path or update the environment variable `LD_LIBRARY_PATH` to point to their location.

The test application requires dynamic linkage to the math library libm.so. If this is not already on your target system, you may have to copy it in from your SDP install from `~/qnx660/target/qnx6/armle-v7/lib/libm.so.1`.

Make sure all the other dependencies of libims.so are present on the system or copy them in from the SDP target folder (see section 4.1 for a list of dependencies).

Run the test application and you should see a Welcome screen, followed a few seconds later by all the iMS Systems it has discovered on the same network subnet.

Figure 2: qnx_ims_test application: welcome screen

Select a system by its number (then Enter) to move to the main menu, which will display details about the configuration and firmware versions of the system to which it is attached.

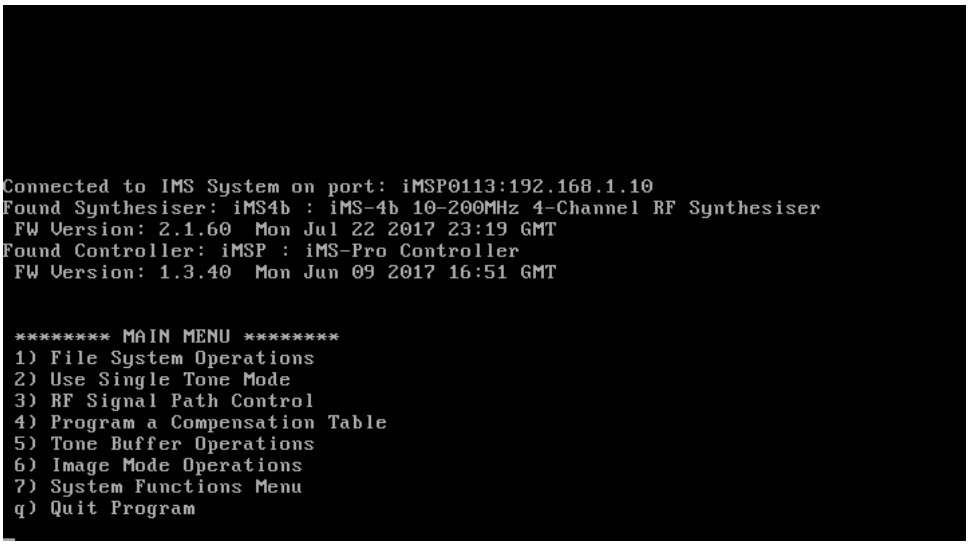

Figure 3: qnx_ims_test Main Menu

From here, you should browse around the menus and submenus to try out different functions made available by the test application. In Image mode, two example images may be created – a frequency ramp and a sawtooth – which may be downloaded to the iMS and played out. These could be observed on a spectrum analyser.

3.3 Building the Test Application

Full source code and an example project is supplied to demonstrate how to generate the test application as a guide to starting your own application development. To build the test application, open your QNX Momentics IDE installation and start a new workspace. From the File menu, choose Import. In the dialog box shown, select General → Existing Projects into Workspace:

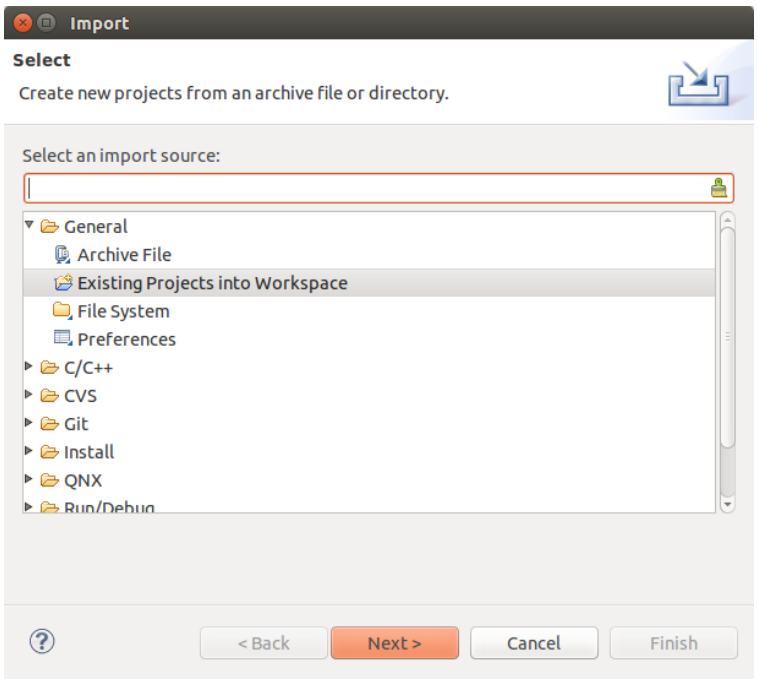

Figure 4: Import Archived Test Application Project

From the next screen, choose “Select archive file” and browse to the SDK location, clicking on the qnx_ims_test.tar.gz archive in the top directory:

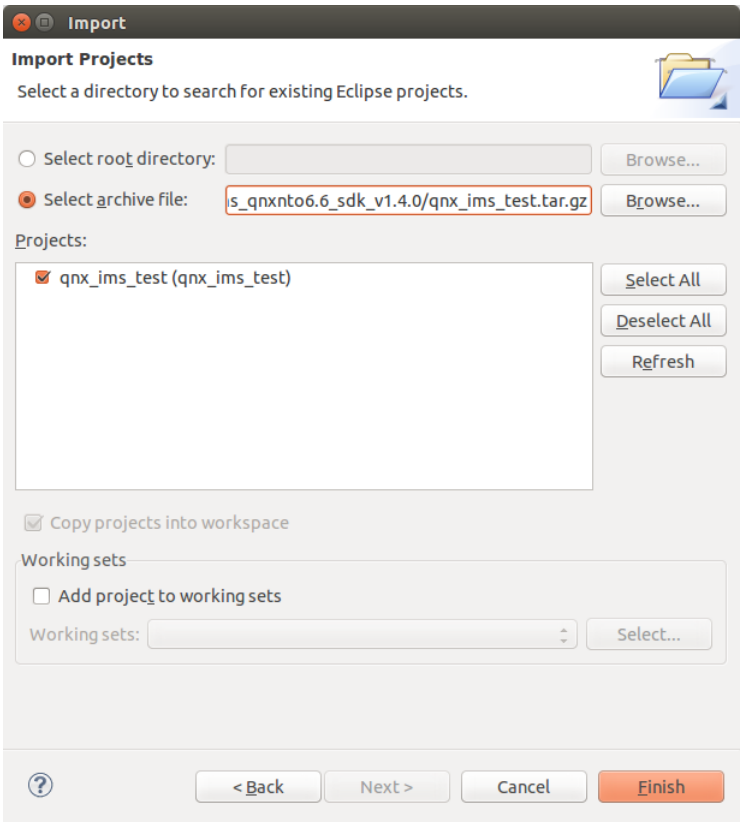

Figure 5: Select Archive File

Make sure the project “qnx_ims_test” is checked then select Finish.

You should see the project file open in the Project Explorer with entries for “Includes”, “arm”, “EXTRA_INCVPATH_console_app”, “EXTRA_INCVPATH_include”, “x86”, “common.mk” and “Makefile”.

There are some absolute paths that will need modifying to build successfully.

Right click “qnx_ims_test” and select Properties. Select “QNX C/C++ Project” and then the “Compiler” tab.

Under Category, select “Extra sources paths” then change the two import directories to point to the location of the `usr/src/qnx/` and `usr/src/SubMenus/` subdirectories of the SDK installation.

Switch category to the “Extra include paths” then change the two import directories to point to the location of the `usr/include/` and `usr/src/` subdirectories of the SDK installation.

Switch category back to “General Options” and check that under Other options is listed “`-std=c++11`”.

Now switch to the Linker tab. Under “General Options” check that the project is set to Link against the Default CPP Library.

If building for both ARM and X86, Click the “Advanced >>” button to bring out a pane that permits you to select different options for different builds. Select first armle-v7 then x86 for “Platform”. Then under the Linker tab, select “Extra Library paths” for the category and change the directory expression to point to the `armle-v7/lib/` subdirectory or `x86/lib/` subdirectory respectively.

Click the “Regular <<” button to remove the pane and return to setting global build options. Under the “Extra libraries category, add two entries, one pointing to “ims” of type “Dynamic” and the other pointing to “m” also of type Dynamic. This will link the build to the iMS library object and the c standard math library (required for the Example Compensation Table generator SubMenu).

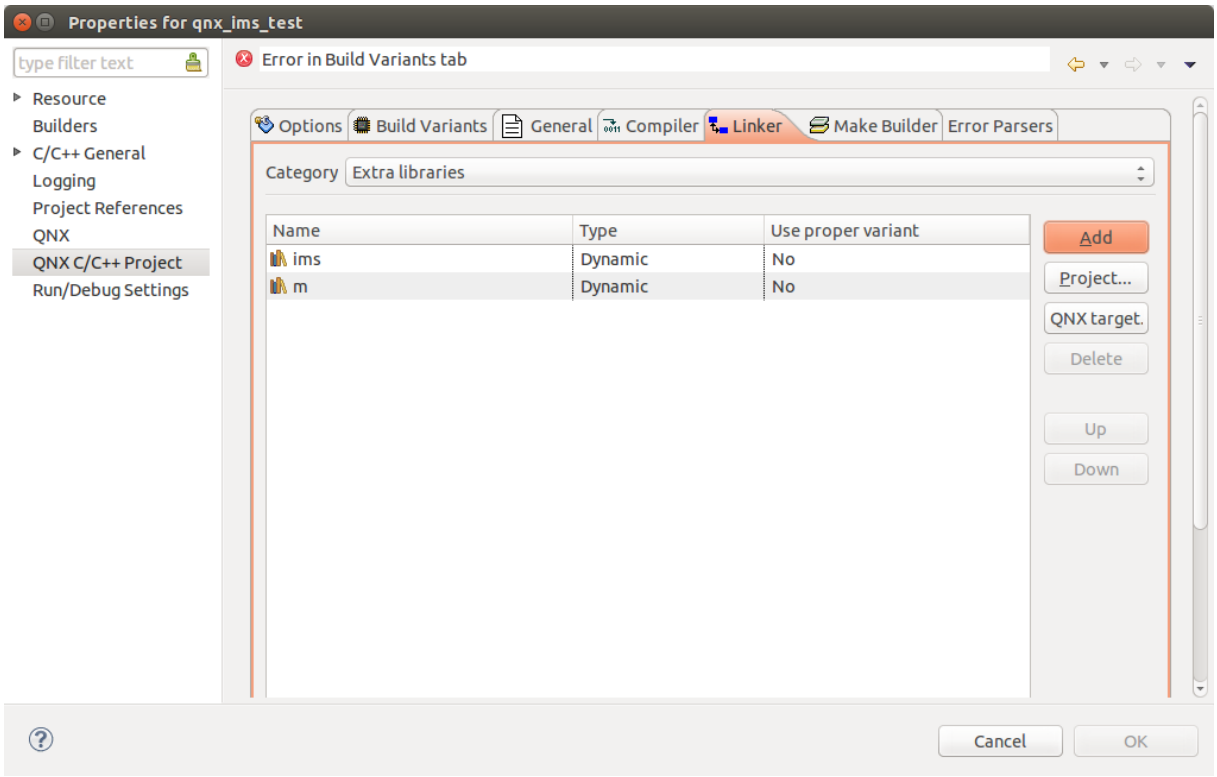

Figure 6: Test Application Linker Settings

Click OK to confirm the project properties, then from the menu select Project → Build All.

You should see the workspace build successfully. Select the Console tab to view build progress. It may take a minute or two.

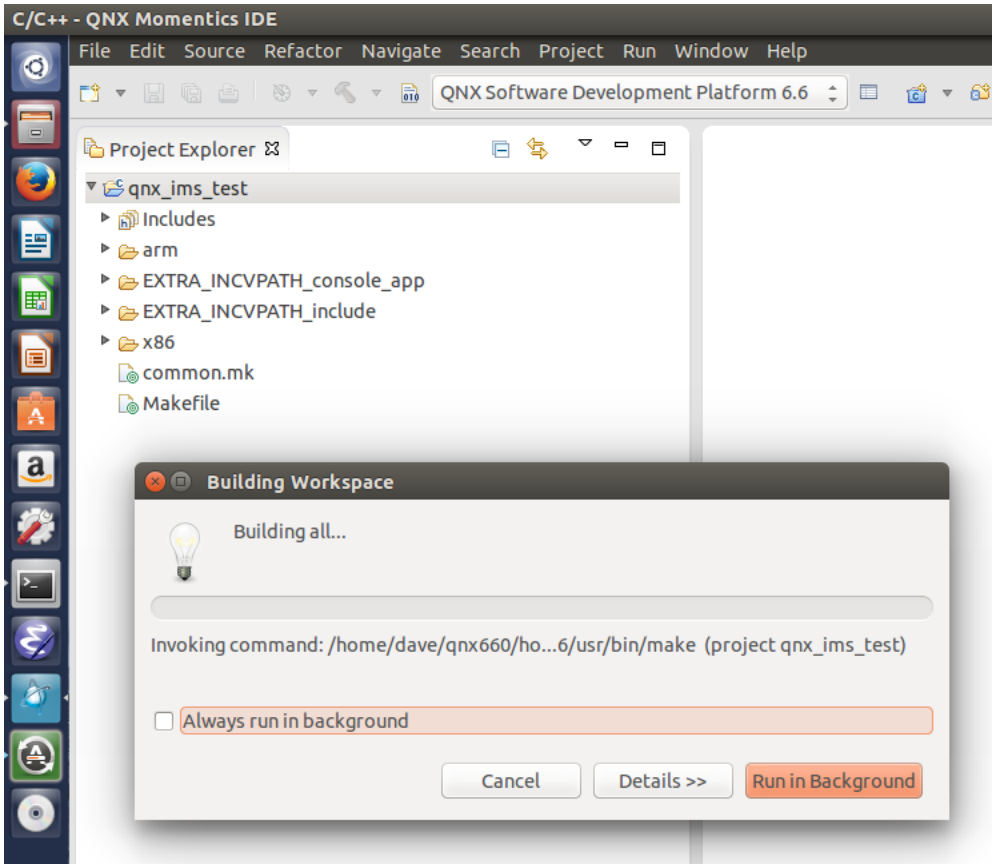

Figure 7: Test Application Build Progress

Once complete, you should be able to copy the executable from the output directory to the target system in the same manner as outlined in section 3.2.

3.4 Build your own iMS Application

Having built and run the test application, you should now be ready to start designing your own iMS based application on QNX Neutrino. Browse through the ims_test_app.cc main source file and the various SubMenu classes to get a feel for how iMS software is typically designed and feel free to copy example source code where useful.

To start your own project, set up your workspace then choose File → New QNX C++ Project. Enter a project name, choose type “Application” and select the relevant Build Variants and Finish.

You will need to set the Project Properties in much the same way as described for the test application in section 3.3. Make sure to point the compiler include paths to the SDK C++ header directory and the linker to the SDK library directory. Also ensure that it is setup to support the C++11 standard with the “`-std=c++11`” compiler option and the Linker to link against the default (Dinkum) C++ library.

If you've followed all of these steps correctly, you should be ready to start building your own applications!

4 Other Information

4.1 Dependencies

The iMS library (both static and shared) require the following dependencies to be present on the target system:

libcpp.so	Default C++ standard library
libm.so	C Math Library
libsocket.so	Berkeley Sockets Networking library
libxml2.so	XML Parsing and Generating library (required for ImageProject class)
libiconv.so	Sub-dependency of libxml2

In most cases, these will be already present on your target system, usually in some standard location such as `/lib/` or `/usr/lib/`. If not, they can be added from your SDP at `~/qnx660/target/qnx6/armle-v7/lib` or `/usr/lib` or the `x86` equivalent.

Note that for the Beaglebone Black BSP that we installed, we had to copy libm, libcpp and libiconv.

4.2 Portability

The SDK for QNX Neutrino is designed to be completely portable such that software written for an iMS application for one platform can be built on a different platform with a different SDK of the same version and is guaranteed to compile. In other words, there is 100% compatibility in the API between different SDKs of the same version number. This means you can do your initial development in Microsoft Visual Studio® for example and move to QNX Neutrino at such time that suitable hardware becomes available.

4.3 Limitations

While the SDK is guaranteed to be 100% API compatible, there are two minor limitations as to the operation of the libraries on a QNX Neutrino RTOS system as detailed below. There are no known other limitations or incompatibilities at this time.

4.3.1 Connection Interfaces

On non-embedded iMS SDK implementations, all of the connection interface types supported by iMS Controllers are supported: USB, Ethernet and RS422/485.

On QNX Neutrino, only Ethernet is supported. There are no plans at present to extend this support to other interface types.

4.3.2 ImageProject .iip Support

The ImageProject class supports storing Image Project data in two formats: the default Compressed format (typically with a '.iip' extension – Isomet Image Project) or the uncompressed XML format. .IIP is usually

favoured because XML files can quickly become very large when many Images or Images with a significant number of Image Points are stored.

However, unfortunately the QNX implementation will not read from or write to .iip files. This is because the standard distribution of libxml2 in the QNX target libraries is not built with zlib support. The workaround for this is to use .xml Image Project files only (by supplying a filename to the ImageProject class with a ".xml" extension, the class will automatically choose not to compress / decompress the file data). Compression may still be achieved by pre- / post- processing the .xml files in a separate shell process.

If it is desired to use a .iip file that has, for example been created on another system, one can do so quite straightforwardly by unzipping the .iip file directly using any decompression tool that supports zlib, for example 7-zip (http://www.7-zip.org).

At present, we do not feel that is of sufficient inconvenience to warrant a fix for this issue, but if it proves to be more than a minor bother to users following feedback, we will look to roll a custom build of libxml2 that does support zlib for future QNX releases.

Annex A Change History

Version	Authors	Date	Status	Comment
1.0.1	Dave Cowan	12-Sept-2017	In work	
1.0.2	Dave Cowan	01-Nov-2017	In work	Updated for SDK v1.4.2