Quickstart Guide

QNX Software Systems Limited
1001 Farrar Road
Ottawa, Ontario
K2K 0B3
Canada

Voice: +1 613 591-0931
Fax: +1 613 591-3579
Email: info@qnx.com
Web: http://www.qnx.com/

Electronic edition published: Thursday, February 20, 2014

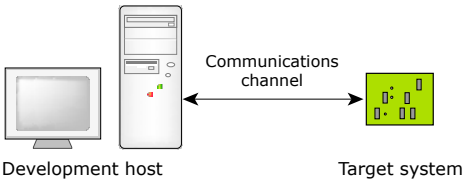
Ten Steps to Developing a QNX Neutrino Program

This guide will help you install and configure the QNX Software Development Platform, which includes the QNX Neutrino RTOS and the QNX Momentics Tool Suite, so you can start developing right away!

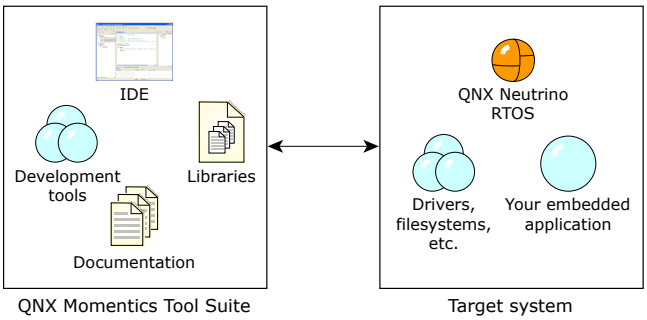
1. Requirements

To write programs that run under the QNX Neutrino RTOS, the first thing you need is the QNX Software Development Platform. This includes the QNX Momentics Tool Suite, which contains everything you need to develop programs that run under the QNX Neutrino RTOS: compiler, linker, libraries, and other QNX Neutrino components, precompiled for all CPU architectures that the QNX Neutrino RTOS supports. This tool suite features an extensive Integrated Development Environment, the QNX Momentics IDE.

You can install QNX SDP on a Windows or Linux *development host* and deploy the QNX Neutrino RTOS on a *target system*:



The development host runs the QNX Momentics Tool Suite; the target system runs the QNX Neutrino RTOS itself plus all the programs you're going to develop:



If you don't have the QNX Software Development Platform, you can download an evaluation version from `www.qnx.com/products/evaluation/`.

You have several choices for the target system that will run the QNX Neutrino RTOS:

- **Embedded hardware:** You can run the QNX Neutrino RTOS on a *reference platform*, a reference design made by a CPU vendor. You'll need a QNX Board Support Package (BSP) for your platform. The documentation that comes with each BSP explains how to build a QNX Neutrino image and install it on that target system.

  For more information about BSPs, see the *BSPs and Drivers* project on our Foundry27 website, `http://community.qnx.com`, as well as the Working with a BSP chapter of the *Building Embedded Systems* guide.

- **Virtual machine:** You can install and run the QNX Neutrino RTOS as a virtual machine in a VMware session.

Although VMware is a handy way to try QNX Neutrino, note that virtual machines don't necessarily support hard realtime.

- **PC target:** You can run the QNX Neutrino on a normal PC, but this is a more advanced task because you have to start the drivers that are appropriate for the hardware. The `procnto` microkernel itself requires only about 700 KB.

Since the QNX Neutrino RTOS is designed the same way for all platforms and is used in the same way, for this Quickstart guide we'll use Linux or Windows as a development host, and a virtual machine as the target.

2. Installing QNX SDP on the development host

Boot your Linux or Windows system and download QNX SDP 6.6 from the Download area on our website, *www.qnx.com/*. Follow the instructions given in the installation note and on the screen.

The installation program will ask you for a license key. If you downloaded an evaluation version of QNX SDP from our website, you should have received an email containing the key. Otherwise, you'll find your key on your license certificate.

3. Installing the QNX Neutrino RTOS on the target system

Next, set up your QNX Neutrino RTOS target as a virtual machine.

We provide a VMware image that's compatible with VMware Workstation 9.0 and VMware Player 5.0. This image is a minimal QNX Neutrino system. You can download a VMware image from:

http://www.qnx.com/products/evaluation/eval-target.html

After you start the virtual machine, you're automatically logged in a `root`. To see a list of the processes that currently exist in your system, type the following on the QNX Neutrino target's console:

```
pidin | less
```

Each process is optional, which means that later in your design, you can remove processes to save resources—or you can add other processes to increase the system's functionality. This also applies for graphics, networking, or audio; each QNX Neutrino is a single process that you can load dynamically.

One of the programs you should see in the list is `qconn`, which you'll need later. Type `q` to exit the `less` command.

4. Networking with the QNX Neutrino RTOS

We've set up the virtual machine to use Network Address Translation (NAT), so that it uses the same IP address as your development host. To determine the target system's IP address, you can use the `ifconfig` command on the target's console:

```
# ifconfig
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 33192
        inet 127.0.0.1 netmask 0xff000000
en0: flags=80008843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST,SHIM> mtu 1500
        address: 00:0c:29:03:51:5a
        media: Ethernet 10baseT full-duplex
        status: active
        inet 192.168.153.132 netmask 0xffffff00 broadcast 192.168.153.255
```

On your development host, use `ping` *IP_address* to check that it can reach your QNX Neutrino target on the network:

```
H:\>ping 192.168.153.132

Pinging 192.168.153.132 with 32 bytes of data:
Reply from 192.168.153.132: bytes=32 time<1ms TTL=255
Reply from 192.168.153.132: bytes=32 time<1ms TTL=255
Reply from 192.168.153.132: bytes=32 time<1ms TTL=255
Reply from 192.168.153.132: bytes=32 time<1ms TTL=255

Ping statistics for 192.168.153.132:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

If your host machine uses a firewall, you might not be able to `ping` it from the target. On Windows, you might have to enable **Allow incoming echo requests** in the ICMP settings.

If you're using a Virtual Private Network (VPN), you might have to disconnect it.

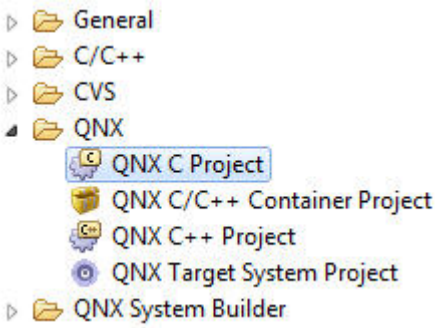
5. Creating a program project

Start the QNX Momentics IDE on your development host:

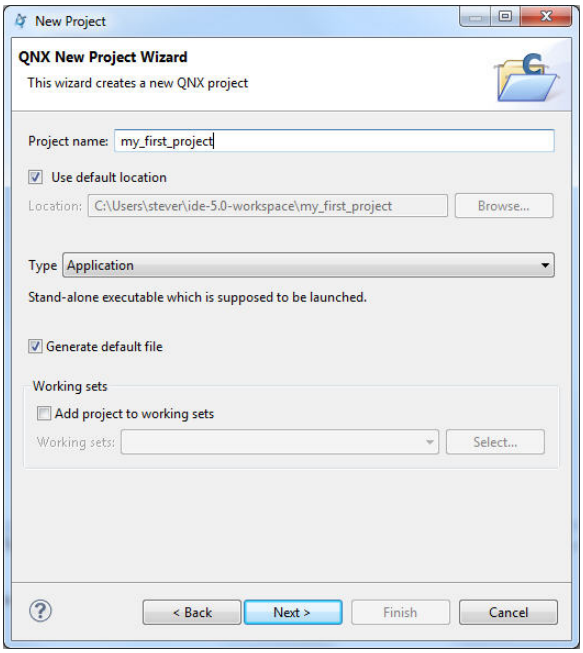
- on a Windows development host, choose **QNX Software Development Platform 6.6 → run-qde.vbs** from the Start menu, or run *base_directory*`\run-qde.vbs` from the command line
- on a Linux development host, run *base_directory*`/run-qde.sh`

where *base_directory* is where you installed QNX SDP. The first time you start the IDE, it asks you to choose a *workspace*, a folder where it can store your projects and other files. The IDE then displays its Welcome page. When you're ready to start, click the Workbench icon:

Now create a QNX C Project: from the File menu, select **New → Project...**. In the New Project dialog, expand **QNX**, and then select **QNX C Project**:

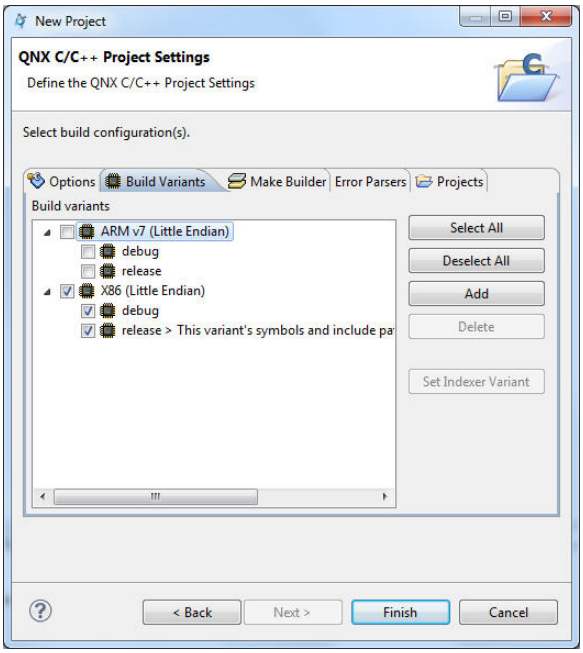


Click **Next**. In the resulting dialog, give your project a name:



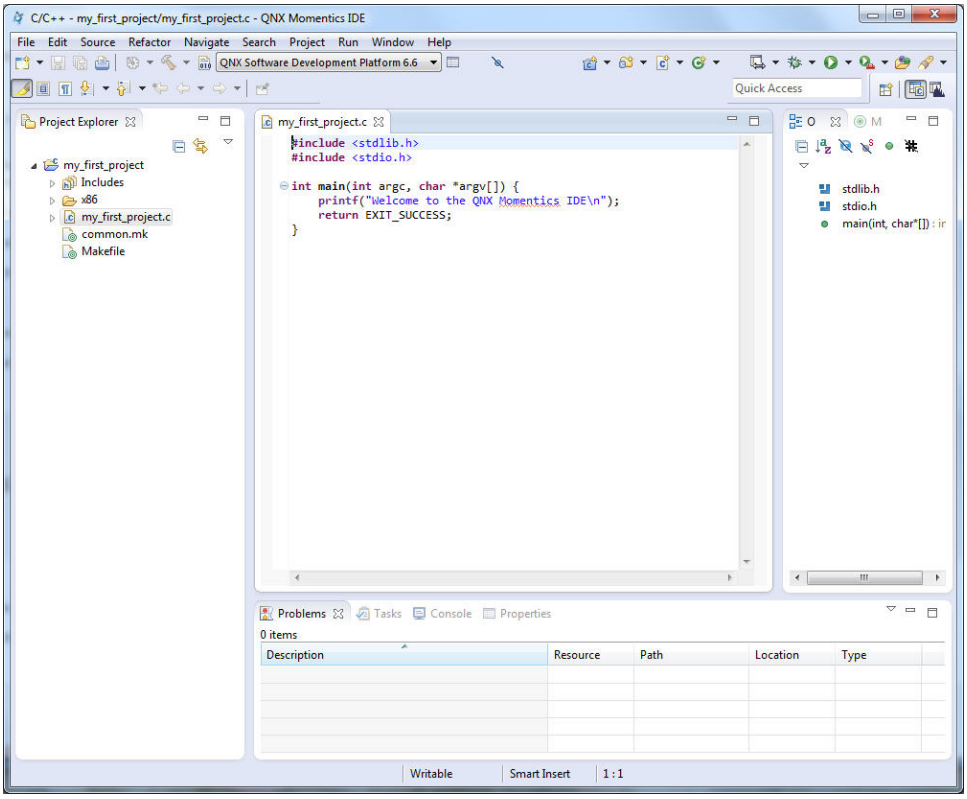
Make sure that **Generate default file** is checked, leave **Add project to working sets** unchecked, and then click **Next**.

You now need to select a CPU architecture for the binary you're creating. To do this, go to the Build Variants tab. Select the appropriate CPU type: ARM or x86. You can also select compilation with or without debug information; we'll be using both later, so make sure the debug and release variants are both checked.



Click **Finish**. A ready-to-use project structure with a `Makefile` is created for you, including a small program ("Welcome to the QNX Momentics IDE"), which you'll find in an automatically generated source code file.

The IDE now switches to the C/C++ perspective, which features the navigator, the editor, and other useful *views*, areas that display information that's relevant to the task at hand:

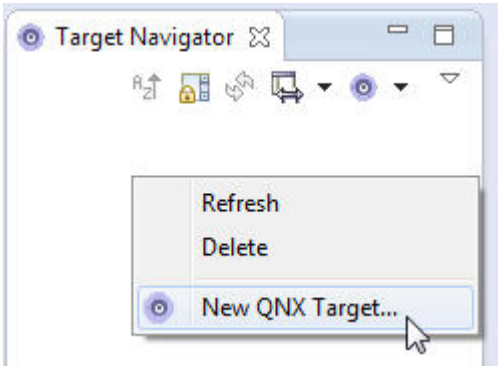


6. Communicating with the QNX Neutrino RTOS

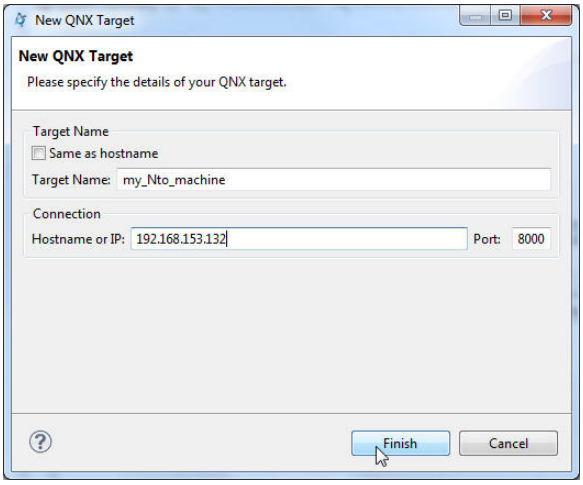
Your target system must be able to respond to requests from the development environment. To make this possible, the target must be running the `qconn` program. If you didn't see it earlier in the output of `pidin`, you can start `qconn` from the console:

```
qconn &
```

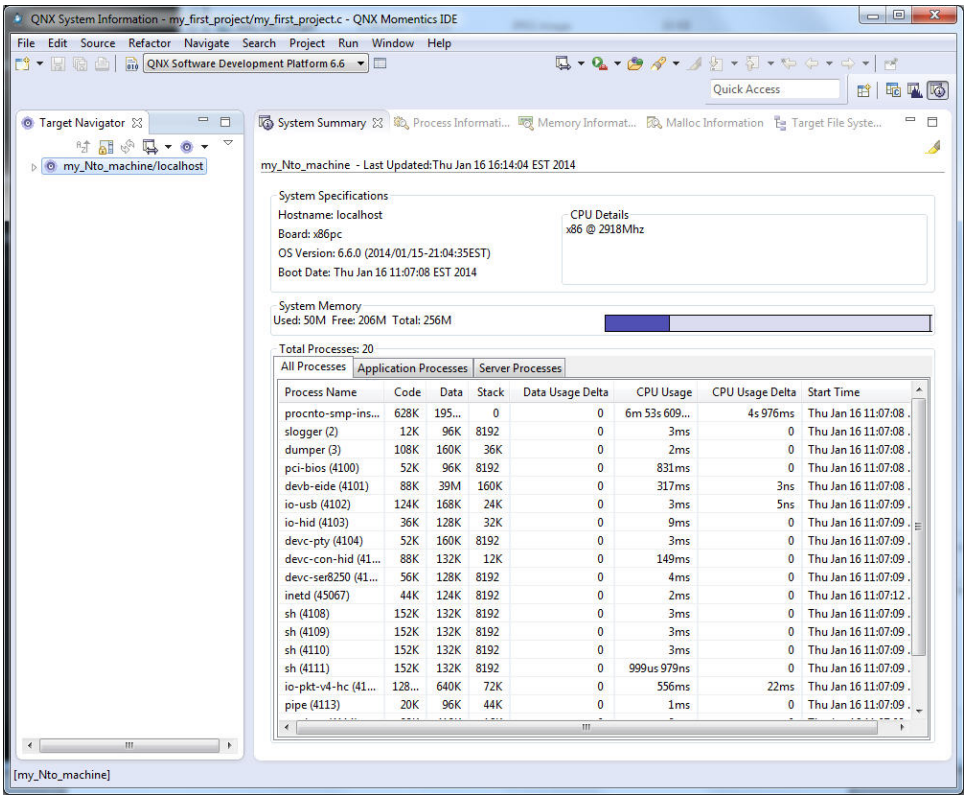
To access your target system from the IDE, you have to create a *target project*. Open the System Information perspective: in the Window menu, select **Open Perspective → QNX System Information**. In the empty Target Navigator view, press the right mouse button and select **New QNX Target...** from the context menu:



If you wish, you can uncheck **Same as hostname** and provide a name for your target system. Enter its IP address in the corresponding field:



Click **Finish**, and then select your new target in the Target Navigator. You will now see a list of all the processes in your QNX Neutrino RTOS system. The views (the tabs at the top) provide other information to you. You can find even more useful views in the Window menu under **Show View**.



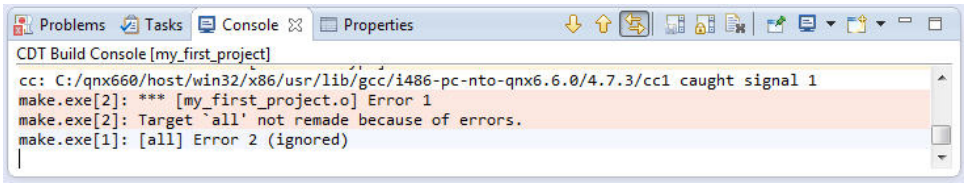
7. Compiling and linking

Now switch back to the C/C++ perspective by choosing its icon in the right side of the toolbar:

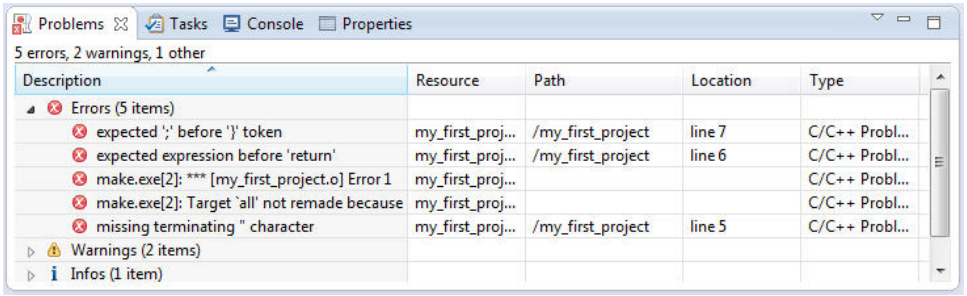
If you didn't do so when you created the project, you need to select compilation with or without debug information. To do this, right-click the project name in the Project Explorer view, and then choose **Properties**. Click **QNX C/C++ Project**, click **Build Variants**, and then expand the x86 item. Make sure that both the debug and release variants are checked. Click **OK**; the IDE offers to rebuild the project.

During the creation of the QNX C Project, a QNX-made directory structure with `Makefiles` was generated. Now to create a binary, please right-click the project name, and then select **Build Project**. The compiler and linker will now do their work.

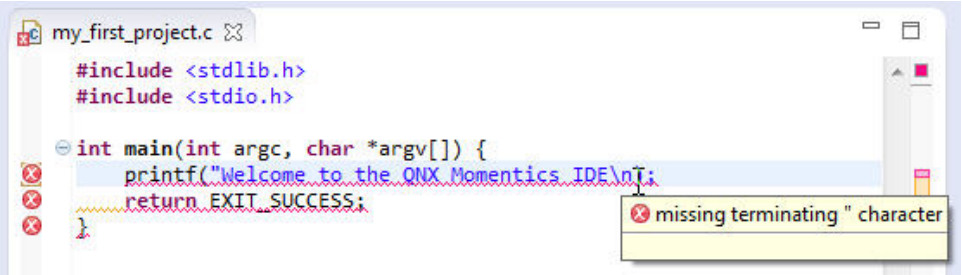
You will find the compiler output in the C-Build output in the Console view, including any errors (you shouldn't see any errors, but we've added one in the examples below):



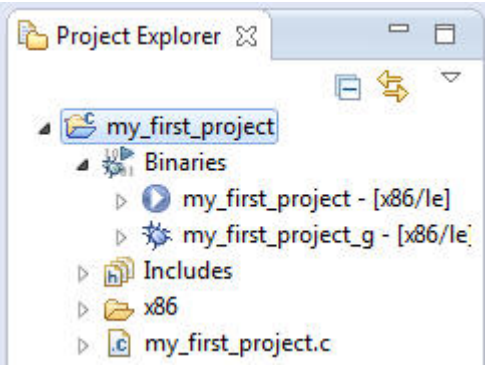
However, if errors occur during compiling, you will find the Problems view more useful, because it displays the output of the compiler in an interpreted and more readable fashion than the Console view:



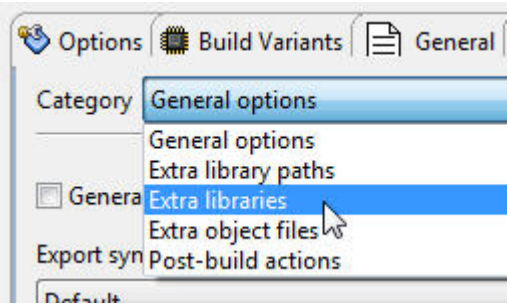
The Editor view also gives you information about an error if you leave the pointer over it:



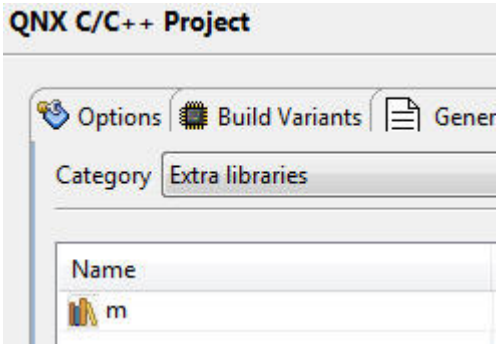
After the build operation, your binaries will be displayed in the `Binaries` folder. Physically, they're located in the CPU directory under `o` (for object) and `o-g` (`-g` for the debug option passed to the compiler). The IDE automatically created the corresponding `Makefiles`.



The QNX library `libc.so`, which contains many basic functions, is linked dynamically to your binary by default. If you want to add other libraries later, you can do so under the **Project → Properties** section. From there, click on **QNX C/C++ Project**, then **Linker**, and then choose **Extra Libraries** in the **Category** field:



Click **Add**, and type the name of the library, without the `lib` prefix or the extension. For example, to add the math library, `libm.so`, you just have to type `m` in the Name field:

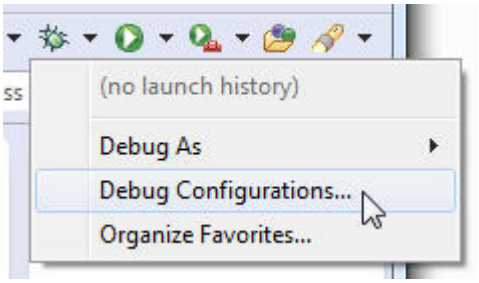


Click **OK**. The linker will now link the library when you build the project.

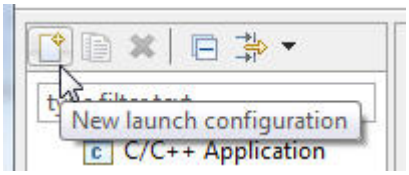
8. Preparing to launch the program

To run and debug the newly built program on your target system, you need to create a *launch configuration*. It consists of various settings that affect how the program starts (e.g. command-line parameters, environment variables). You enter these once, and then you can use this collection of settings again and again.

Now create your own launch configuration: from the dropdown menu beside the "bug" icon on the toolbar, select **Debug Configurations. . .** :



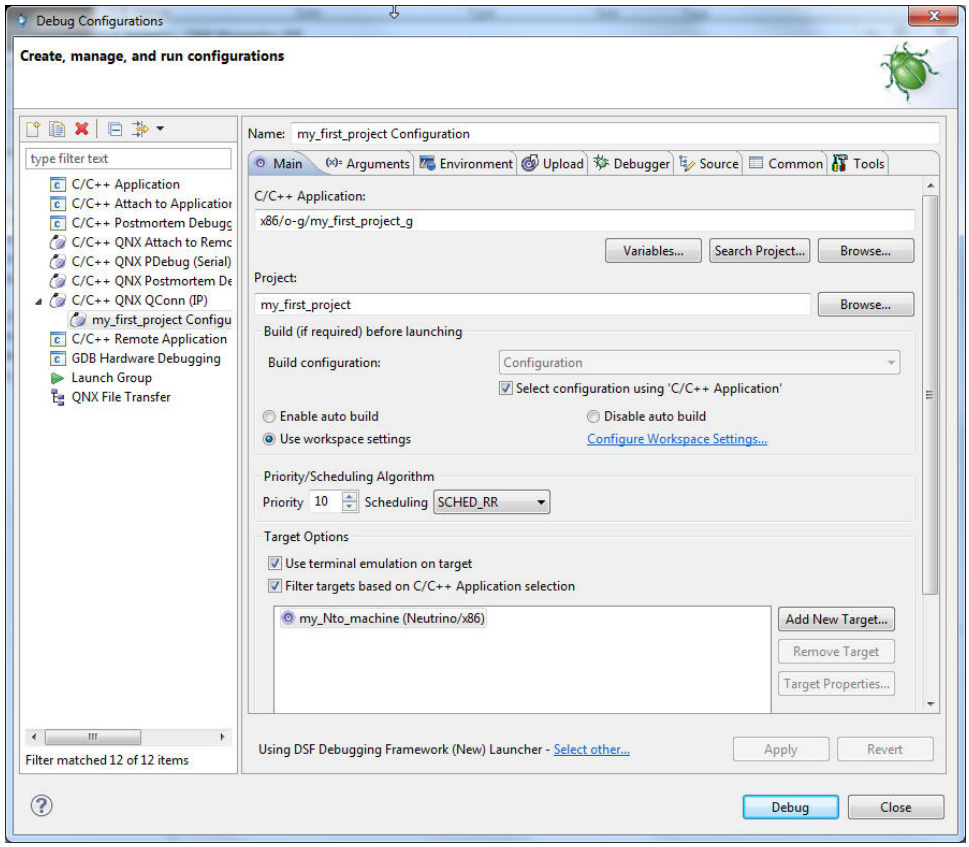
A dialog window opens, where you can start existing launch configurations, change them, or create new ones. On the left, select **C/C++ QNX QConn (IP)**. This type of launch configuration is meant for network-based (cross) development with the QNX Neutrino RTOS running on the target system, using the `qconn` program. Now click on the **New launch configuration** icon:



You will now be presented with many configuration possibilities that all deal with starting your executable program. Right now, only the **Main** tab needs your input. Later, however, you should also take a look at what the other tabs have to offer.

If you wish, you can change the name of your configuration at the top of the dialog. If your project isn't already selected, click the **Browse. . .** button beside the Project field, and select your project. Next to the **C/C++ Application** field, press the **Search Project** button and choose your binary. The names of binaries compiled with debug information include a suffix of _g. Since we would like to start the Debugger in the next step, please choose the binary with the debug information. Click **OK**.

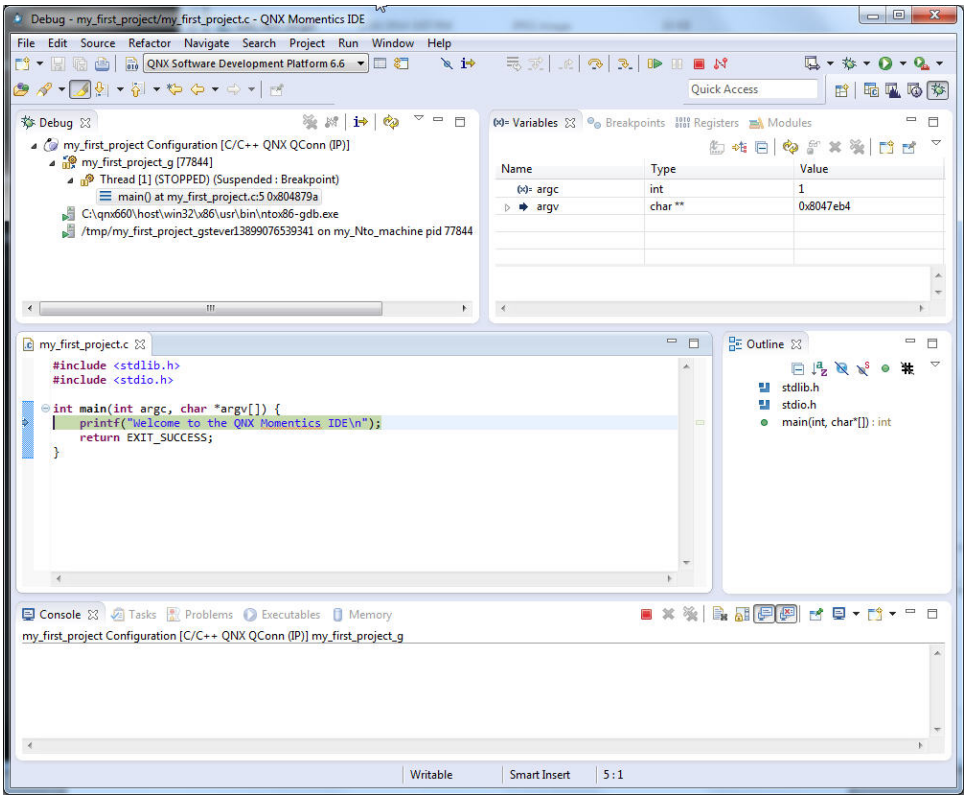
Make sure your target system is listed under **Target Options**, and then click **Apply**—the launch configuration is now ready:



9. Starting and debugging

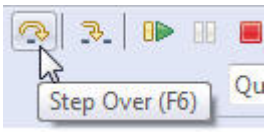
You should still be in the Debug launch configuration dialog. You just created a configuration for launching your program, which you now can start in the debugger. To do this, please click **Debug**.

The IDE now switches to the Debug perspective and transfers your program from your development machine across the network to your target QNX Neutrino system, and then starts it under the control of the debugger. You will see that the debugger stops in the first line of your program. In the Debug view, you'll see an overview of your process, including the call stack. Using the buttons in the main bar of the Debug view, you can control the debugger.



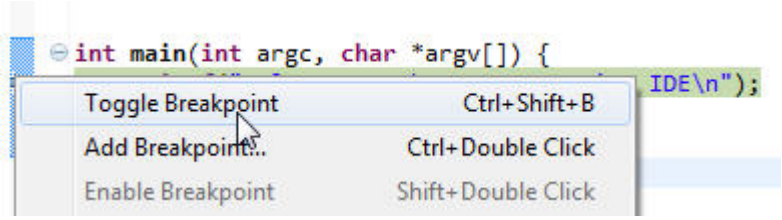
When you run or debug your application from the IDE, any input is read from the IDE's console, and any output goes to it. Once execution has passed the line that calls *printf()*, you should see the “Welcome to the QNX Momentics IDE” message in the Console window.

Using the **Step Over** button, you can jump to the next line of code:



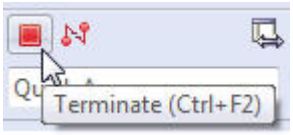
During debugging, you can watch the Variables view on the right, which displays how your variables change. You can use the **Step Into** button to let the debugger go into the code of a function (which, of course, is useful only if you have the source code for this function).

To set a breakpoint, place the mouse pointer over the left border of the source display, press the right mouse button and choose **Toggle Breakpoint** from the context menu. The breakpoint is shown as a little circle, which you can also set or remove while you write your code.



When the running program hits a breakpoint, it stops in the debugger, and you can, for example, examine your variables. If you click the **Resume** button, your program continues until it hits the next breakpoint:

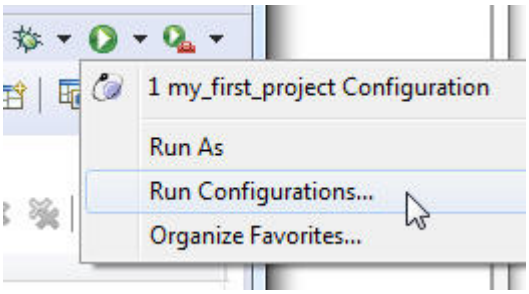
To abort program execution, use the **Terminate** button:



After the program has finished running, you can use the **Remove All Terminated Launches** button to clear all terminated launches from the Debug view:

The debugger keeps the project's files open while the program is running. Be sure to terminate the debug session before you try to rebuild your project, or else the build will fail.

To run your program as a standalone binary (without the debugger), open the dropdown menu beside the Run icon and choose **Run Configurations...** :



Then you can use the launch configuration (described in the previous step) to start your program. Or create a new launch configuration and select the binary without debug information. You can also use the System Information perspective's Target File System Navigator (**Window → Show View**) to manually transfer your binary, and then run it by double-clicking on it (or by right-clicking on it and selecting **Run**).

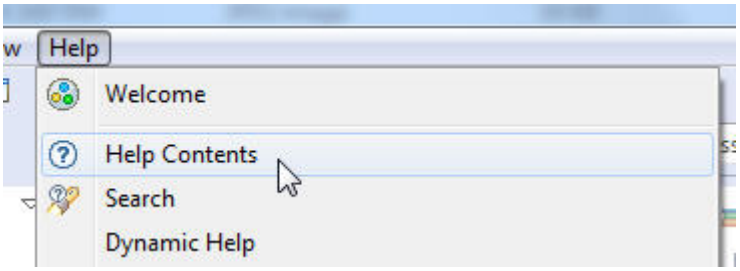
It's also possible to leave the binary on a shared network drive on your development host, mount the drive on your QNX Neutrino target (see the entry for `fs-cifs` in the QNX Neutrino *Utilities Reference*), and run the binary from there.

10. Making the program your own

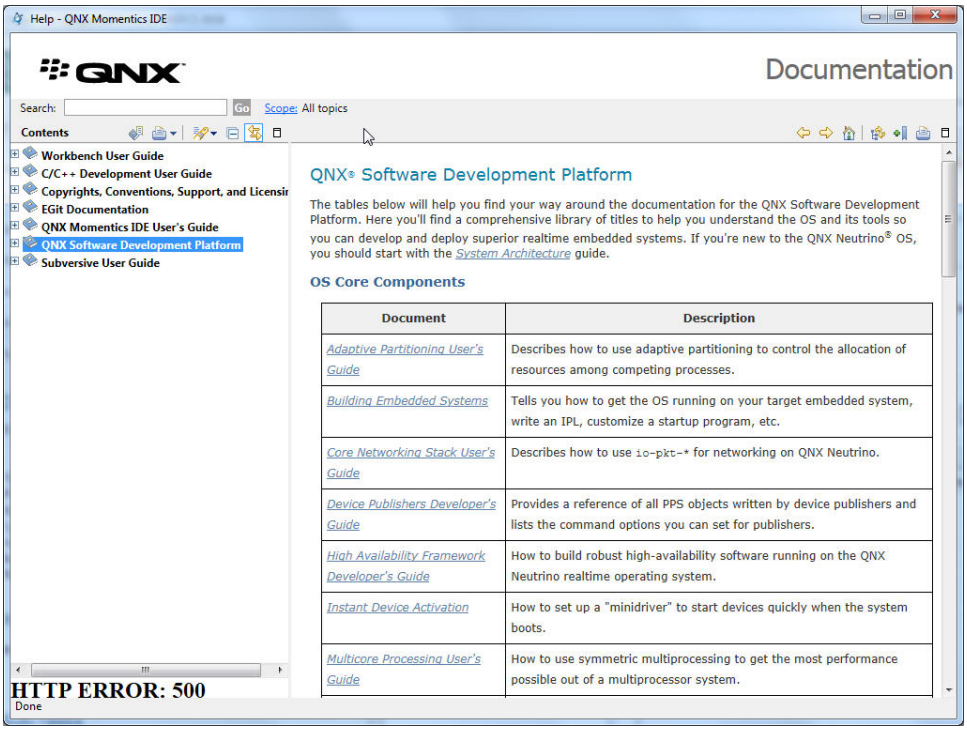
To turn this default program into your own first QNX Neutrino program, you can modify and extend the source code we just created. Try some of our sample programs and copy code from them into your project. And now that you've started, you'll probably want a lot more information, such as how to create your own threads, how the QNX Neutrino message-passing works, which process-synchronization methods are available, how to get access to I/O areas, or how to build a QNX Neutrino resource manager. But don't worry: all this is (almost) as simple as the quick start you just experienced!

The IDE includes a number of tutorials to help you get started. Choose **Help → Welcome** from the IDE's toolbar, and then click the Tutorials icon:

The IDE's Help system includes the QNX documentation, along with information about the Eclipse platform. In the Help menu, click **Help Contents**:



The roadmap for the QNX Software Development Platform will help you find out where to look in the documentation for the information you need. We recommend browsing the QNX Neutrino *System Architecture* guide, the IDE *User's Guide*, and the QNX Neutrino *Programmer's Guide*.



The IDE even includes source code examples covering thread creation, usage of mutexes, message-passing and other methods of interprocess communication, as well as a QNX Neutrino resource-manager template. Choose **Help** → **Welcome**, and then click the Samples icon:

The source features extensive comments, explaining what is done there. For every function you are interested in, you also should consult the QNX Neutrino *C Library Reference*; for utilities, see the *Utilities Reference*.

While you explore the QNX Neutrino RTOS and its SDK, you will probably have further questions. Please contact your QNX Account Manager, Field Application Engineer, or our support department, and visit our Foundry27 community website (`http://community.qnx.com`). We want to be with you from the start, because we are successful only if you are!