QNX® Software Development Platform 7.0

Quickstart Guide

BlackBerry | QNX

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: March 06, 2020

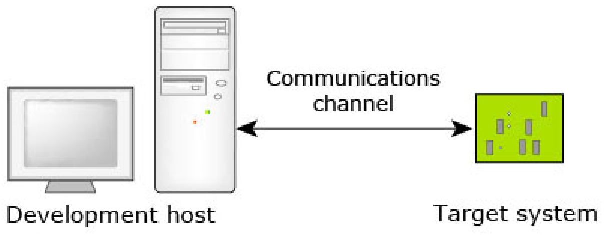
Seven Steps to Developing a QNX Neutrino Program

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!

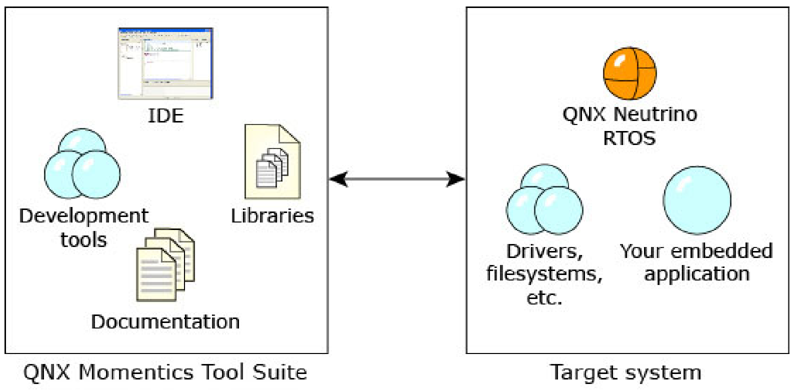
Requirements

To write programs that run under the QNX Neutrino RTOS, the first thing you need is the QNX Software Development Platform (SDP). 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 components, precompiled for all CPU architectures that QNX Neutrino supports. This tool suite features an extensive Integrated Development Environment, the QNX Momentics IDE.

You can install QNX SDP on a Linux, macOS, or Windows *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 SDP, 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. Each BSP comes with documentation that explains how to build a QNX Neutrino image and install it on that target system.

  For more information about BSPs, see our website, *www.qnx.com*, as well as the Working with QNX BSPs chapter of the *Building Embedded Systems* guide.

- **Virtual machine:** You can 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 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.

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 Windows as a development host, and a virtual machine as the target.

1. Installing QNX SDP on the development host

Boot your Linux, macOS, or Windows system and download the QNX Software Center archive from the Download area on our website, *www.qnx.com*. Follow the instructions given in the installation note and on the screen. Once you've installed the QNX Software Center, use it to install QNX SDP. Instructions on doing so are found in the *QNX Software Center User's Guide*.

When you install QNX SDP, the QNX Software Center will prompt you for your myQNX credentials. When you enter them, it retrieves the license keys assigned to your account and you just have to accept one. If it doesn't find any keys, contact the license manager in your organization so they can assign you a key, or access your License Certificate and use the information within it to manually register a license to your account.

For information about confirming which licenses have been assigned to you and viewing your SDP 7 license certificates, see the *QNX Software Center User's Guide*.

2. 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 Pro 12.0 or later, VMware Workstation Player 12.0 or later, and VMware Fusion Pro 8.0 or later. This image is a minimal QNX Neutrino system. You can download a VMware image from the QNX Software Center by choosing the Available tab, expanding the Reference Images list, expanding the Tools list, and then choosing the appropriate virtual machine.

When the installation is complete, start VMware, choose **File → Open...** and navigate to *vm_base_directory***/vmimages/qnx700.***architecture.build_ID/architecture***/QNX_SDP.vmx**, where *vm_base_directory* is where you installed the addon.

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

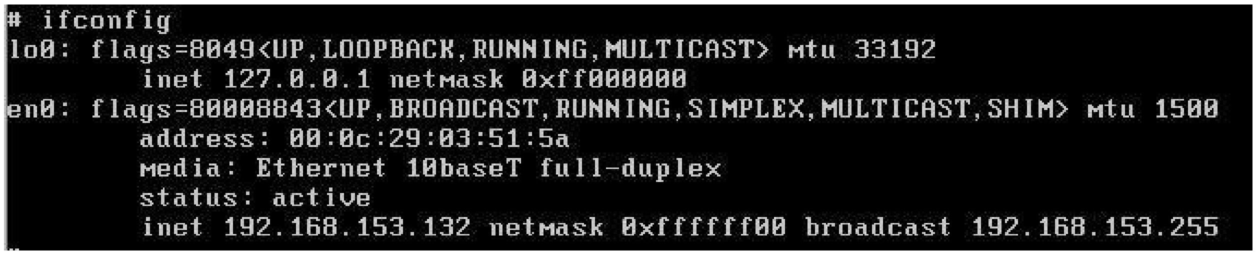
```
pidin | less
```

Each process is optional (except **procnto-smp-instr**, which is the microkernel), 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 component 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.

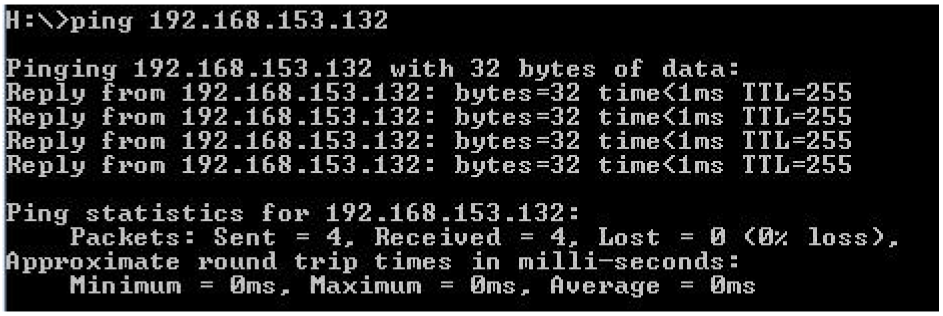
3. Networking with the QNX Neutrino RTOS

We've set up the virtual machine to use Network Address Translation (NAT), so that it's on the same IP network 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.

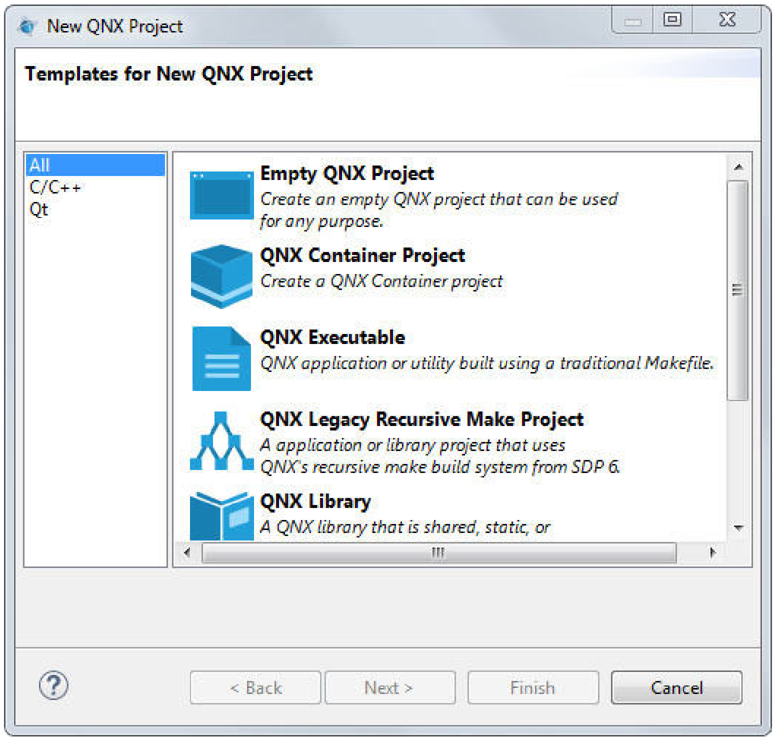
4. Creating a program project

Start the QNX Momentics IDE on your development host:

- on Windows, choose **QNX** → **QNX Momentics IDE** from the Start menu; you can also click the desktop icon with the same label, which is added when you install QNX SDP
- on Linux, run ***base_directory*/qnxmomentics/qde**, where *base_directory* is where you installed QNX SDP
- on macOS, click the icon labelled **QNX Momentics IDE** from the launchpad; this icon is added when you install 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 in the upper right corner:

Now create a project: from the File menu, select **New** → **QNX Project**, and then select **QNX Executable**:



Templates for New QNX Project

Empty QNX Project
Create an empty QNX project that can be used for any purpose.

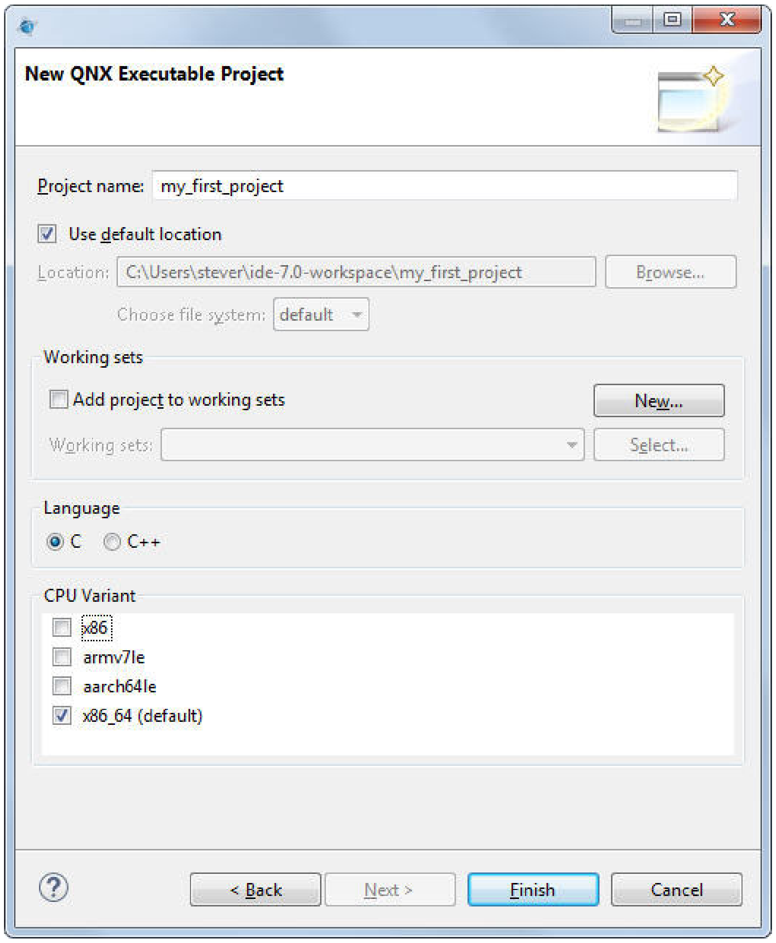
QNX Container Project
Create a QNX Container project

QNX Executable
QNX application or utility built using a traditional Makefile.

QNX Legacy Recursive Make Project
A application or library project that uses QNX's recursive make build system from SDP 6.

QNX Library
A QNX library that is shared, static, or

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



New QNX Executable Project

Project name: my_first_project

[✓] Use default location

Location: C:\Users\stever\ide-7.0-workspace\my_first_project

Choose file system: default

Working sets

[ ] Add project to working sets

Working sets:

Language

● C ○ C++

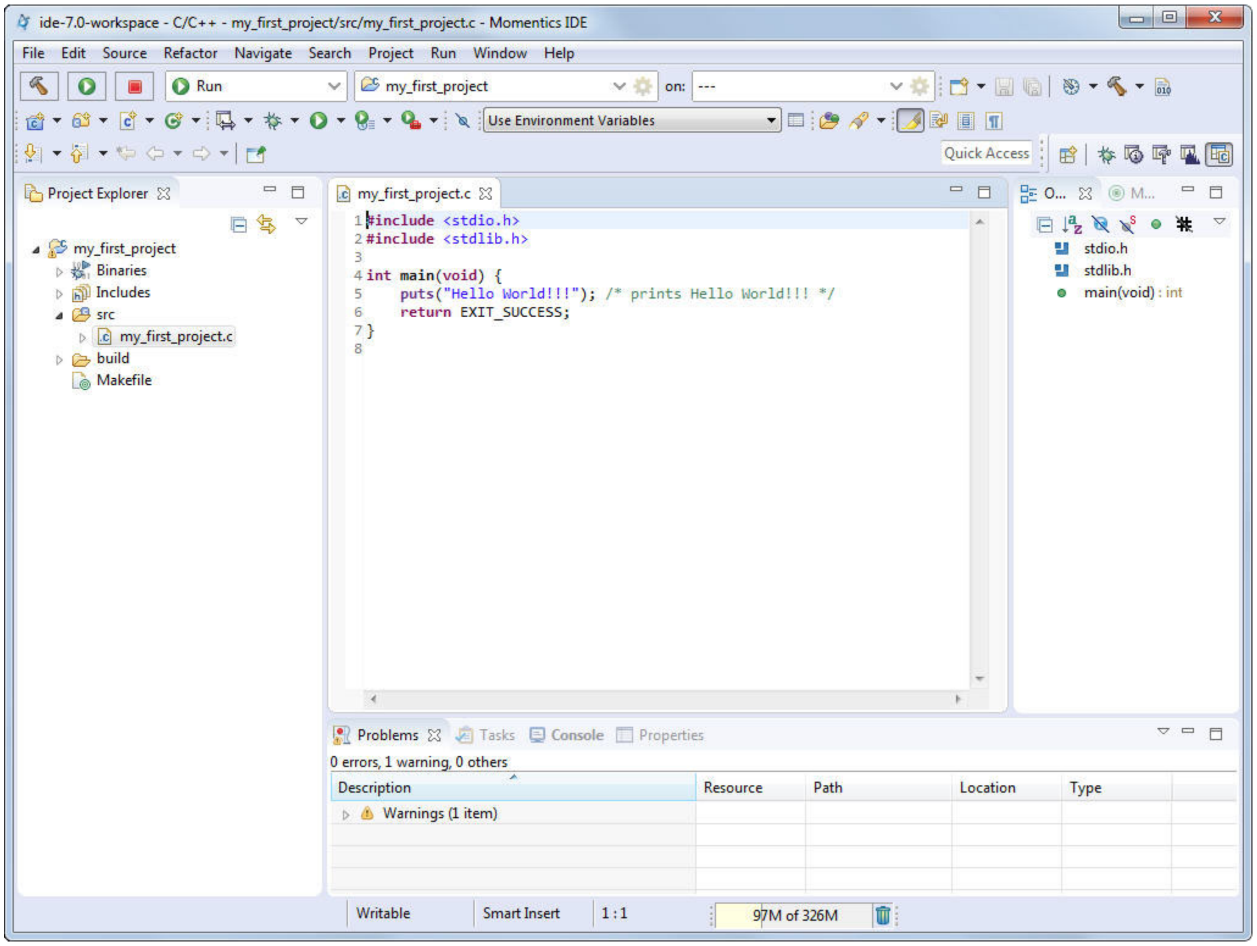
CPU Variant

[ ] x86
[ ] armv7le
[ ] aarch64le
[✓] x86_64 (default)

Leave **Use default location** checked and **Add project to working sets** unchecked, and then select a CPU architecture for the binary you're creating.

Click **Finish**. A ready-to-use project structure with a makefile is created for you, including a small program (“Hello World!!!”), 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:



```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    puts("Hello World!!!"); /* prints Hello World!!! */
    return EXIT_SUCCESS;
}
```

0 errors, 1 warning, 0 others

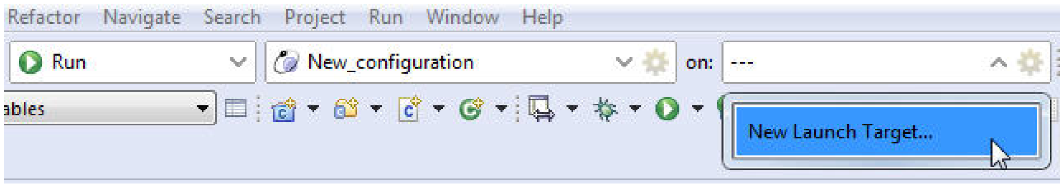
Description	Resource	Path	Location	Type
Warnings (1 item)				

5. 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:

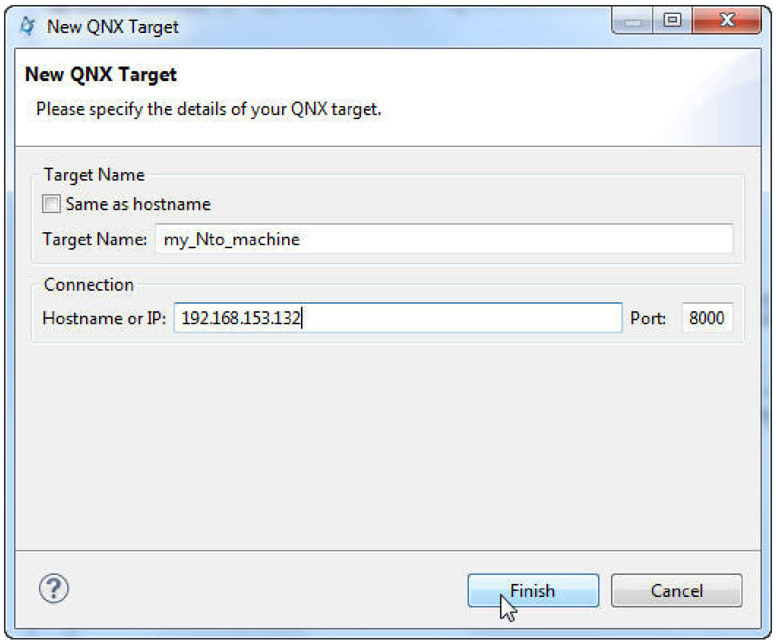
```
qconn &
```

To access your target system from the IDE and run applications on it, you have to create a *target connection*. Click on the **on:** box in the launch bar at the top of the IDE window, and select **New Launch Target**.



Choose **QNX Target** and click **Next**.

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:



New QNX Target

Please specify the details of your QNX target.

Target Name

Same as hostname

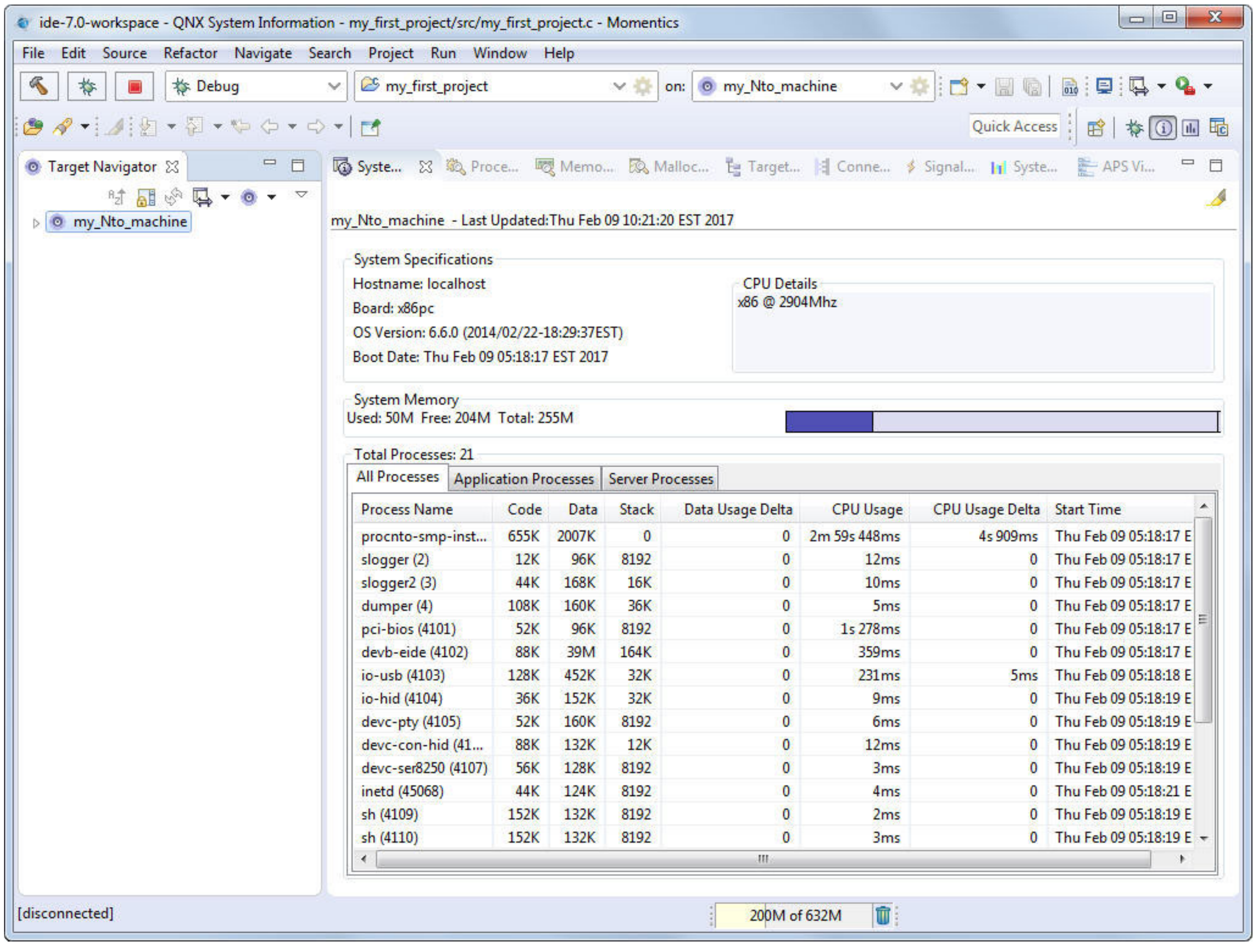
Target Name: my_Nto_machine

Connection

Hostname or IP: 192.168.153.132 Port: 8000

Click **Finish**. If you want to see what's running on the target, open the System Information perspective by clicking on its icon (ⓘ) in the right side of the toolbar and then select your new target in the Target Navigator.

In the **System Summary** view, you will see a list of all processes in your QNX Neutrino RTOS system:


ide-7.0-workspace - QNX System Information - my_first_project/src/my_first_project.c - Momentics
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
my_first_project
on: my_Nto_machine
Quick Access
Target Navigator
my_Nto_machine
Syste... Proce... Memo... Malloc... Target... Conne... Signal... Syste... APS Vi...
[disconnected]
200M of 632M

my_Nto_machine - Last Updated:Thu Feb 09 10:21:20 EST 2017

System Specifications
Hostname: localhost
Board: x86pc
OS Version: 6.6.0 (2014/02/22-18:29:37EST)
Boot Date: Thu Feb 09 05:18:17 EST 2017

CPU Details
x86 @ 2904Mhz

System Memory
Used: 50M Free: 204M Total: 255M

Total Processes: 21

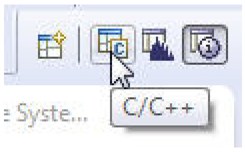
All Processes | Application Processes | Server Processes

Process Name	Code	Data	Stack	Data Usage Delta	CPU Usage	CPU Usage Delta	Start Time
procnto-smp-inst...	655K	2007K	0	0	2m 59s 448ms	4s 909ms	Thu Feb 09 05:18:17 E
slogger (2)	12K	96K	8192	0	12ms	0	Thu Feb 09 05:18:17 E
slogger2 (3)	44K	168K	16K	0	10ms	0	Thu Feb 09 05:18:17 E
dumper (4)	108K	160K	36K	0	5ms	0	Thu Feb 09 05:18:17 E
pci-bios (4101)	52K	96K	8192	0	1s 278ms	0	Thu Feb 09 05:18:17 E
devb-eide (4102)	88K	39M	164K	0	359ms	0	Thu Feb 09 05:18:17 E
io-usb (4103)	128K	452K	32K	0	231ms	5ms	Thu Feb 09 05:18:18 E
io-hid (4104)	36K	152K	32K	0	9ms	0	Thu Feb 09 05:18:19 E
devc-pty (4105)	52K	160K	8192	0	6ms	0	Thu Feb 09 05:18:19 E
devc-con-hid (41...	88K	132K	12K	0	12ms	0	Thu Feb 09 05:18:19 E
devc-ser8250 (4107)	56K	128K	8192	0	3ms	0	Thu Feb 09 05:18:19 E
inetd (45068)	44K	124K	8192	0	4ms	0	Thu Feb 09 05:18:21 E
sh (4109)	152K	132K	8192	0	2ms	0	Thu Feb 09 05:18:19 E
sh (4110)	152K	132K	8192	0	3ms	0	Thu Feb 09 05:18:19 E

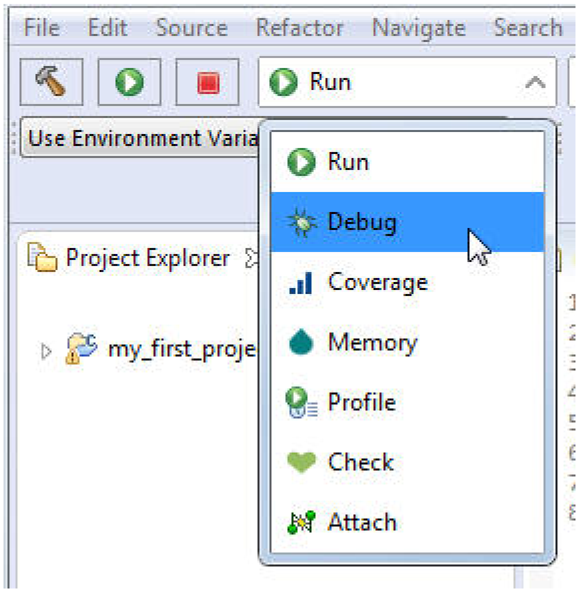
The views (the tabs at the top) provide other information and you can find even more views in the Window menu under **Show View**.

6. Compiling and linking

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

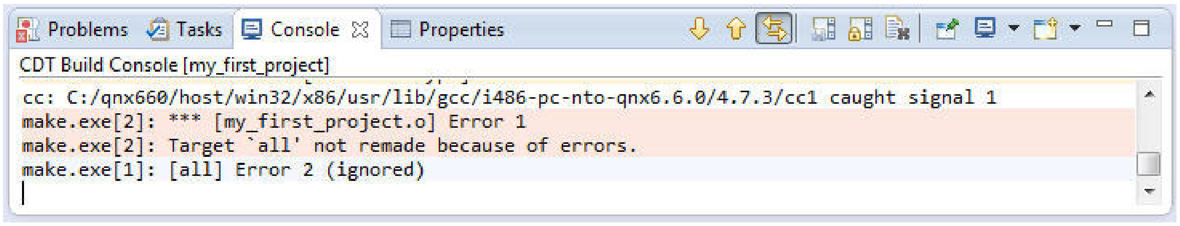


You need to select compilation with debug information. To do this, choose **Run** or **Debug** from the launch bar:



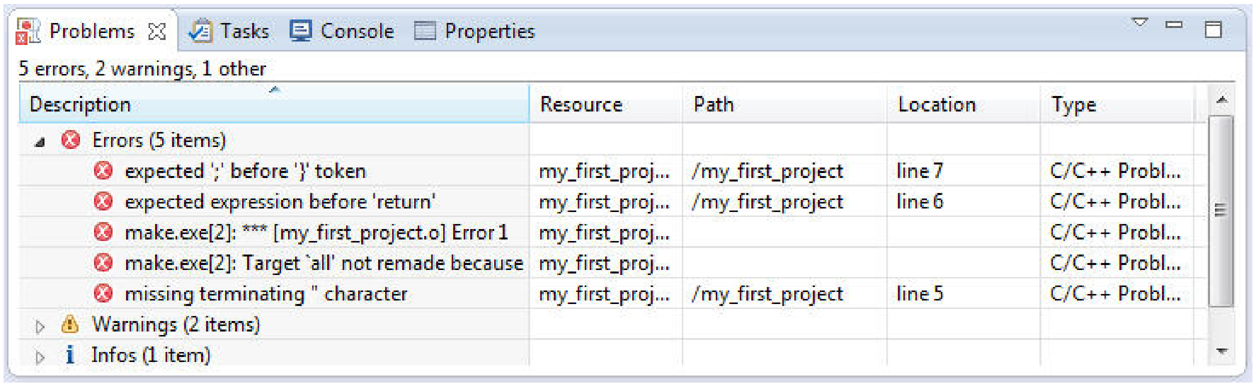
When you created the project, the IDE also created a makefile. Now to create a binary, click the Build button ( ) or right-click the project name and select **Build Project**. The compiler and linker will now do their work.

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 example below):



```
cc: C:/qnx660/host/win32/x86/usr/lib/gcc/i486-pc-nto-qnx6.6.0/4.7.3/cc1 caught signal 1
make.exe[2]: *** [my_first_project.o] Error 1
make.exe[2]: Target `all' not remade because of errors.
make.exe[1]: [all] Error 2 (ignored)
```

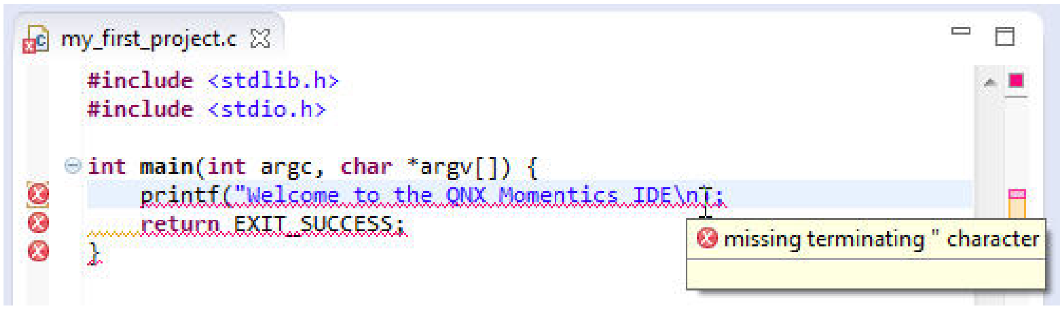
If errors occur during compiling, you will find the Problems view more useful, because it displays the output of the compiler in an interpreted and more readable fashion than the Console view:



5 errors, 2 warnings, 1 other

Description	Resource	Path	Location	Type
Errors (5 items)				
expected ';' before '}' token	my_first_proj...	/my_first_project	line 7	C/C++ Probl...
expected expression before 'return'	my_first_proj...	/my_first_project	line 6	C/C++ Probl...
make.exe[2]: *** [my_first_project.o] Error 1	my_first_proj...			C/C++ Probl...
make.exe[2]: Target `all' not remade because	my_first_proj...			C/C++ Probl...
missing terminating " character	my_first_proj...	/my_first_project	line 5	C/C++ Probl...
Warnings (2 items)				
Infos (1 item)				

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

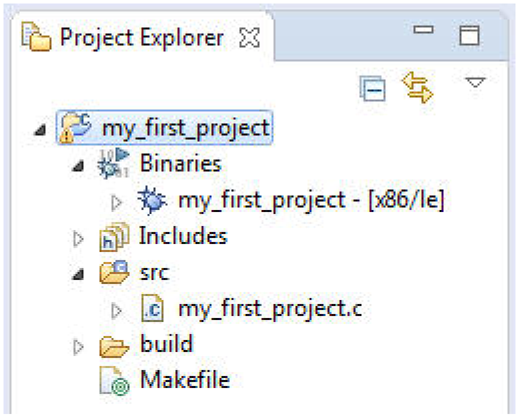


```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("Welcome to the QNX Momentics IDE\n);
    return EXIT_SUCCESS;
}
```

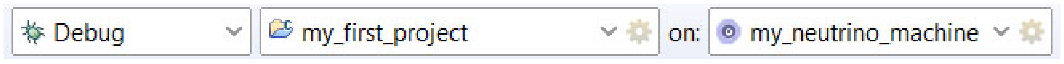
missing terminating " character

After the build operation, your binaries will be displayed in the **Binaries** folder. Physically, they're located in the **build** directory for your project. The IDE automatically created the corresponding makefiles.



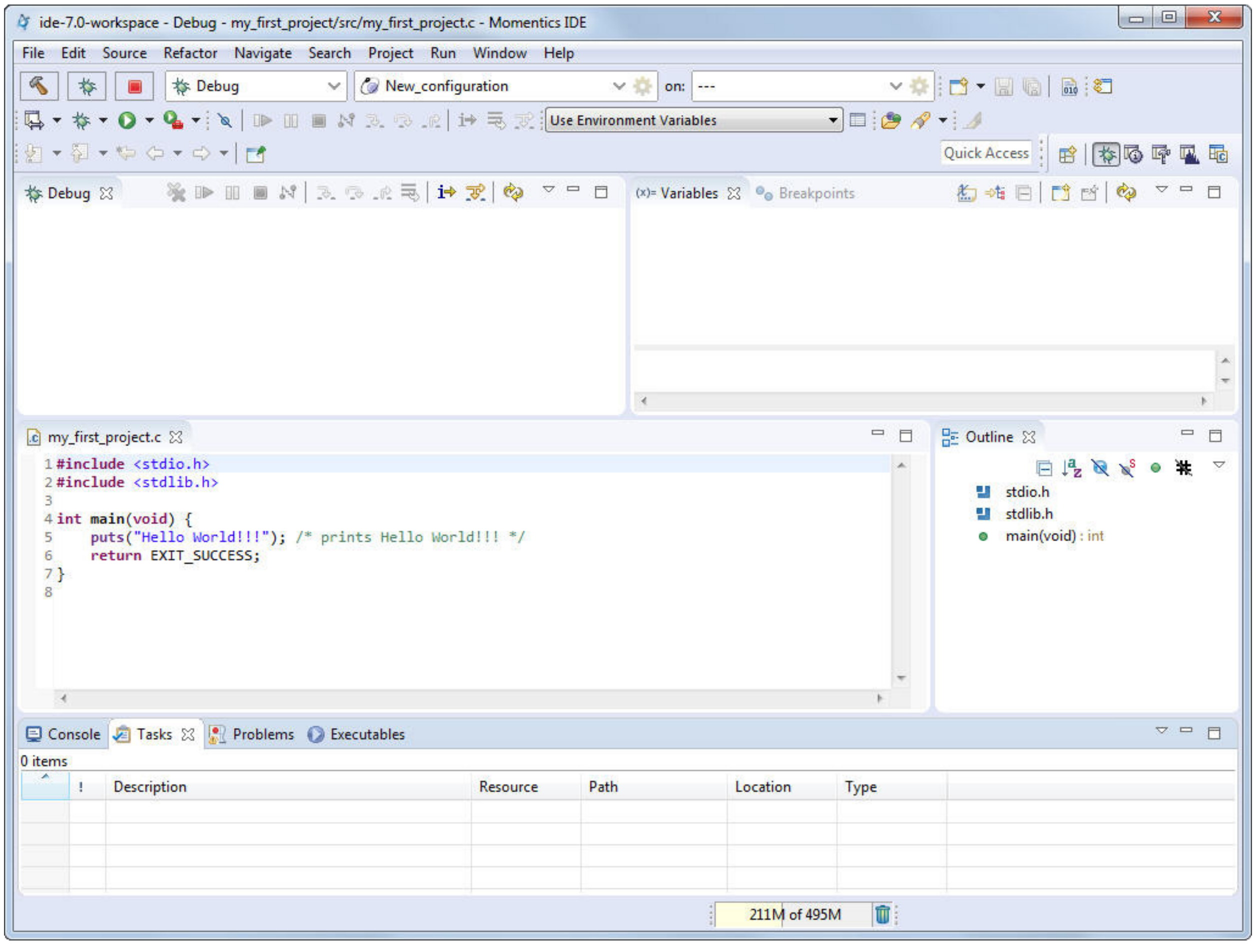
7. Launching and debugging the program

To run and debug the newly built program on your target system, you simply need to select three things in the launch bar: your project, the target you'd like to run it on, and **Debug**:



Then click the Debug button in the launch bar ( ).

The IDE now switches to the Debug perspective and transfers your program from your development machine across the network to your target system, and then starts it under the control of the debugger. 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.

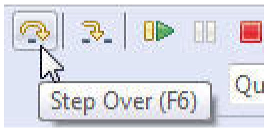


```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    puts("Hello World!!!"); /* prints Hello World!!! */
    return EXIT_SUCCESS;
}
```

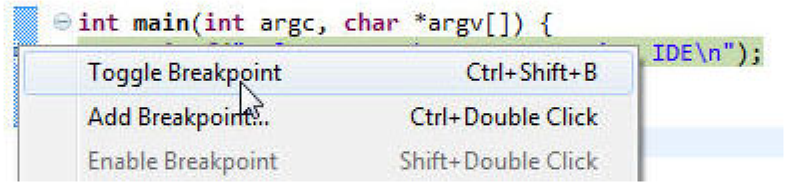
When you run or debug your application from the IDE, any input is read from the IDE's console, and any output goes to it. Once execution has passed the line that calls *puts()*, you should see the “Hello World!!!” message in the Console window.

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).

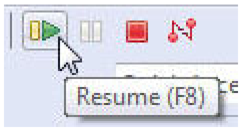
To set a breakpoint, place the mouse pointer over the left border of the source display, right-click and choose **Toggle Breakpoint** from the context menu. The breakpoint is shown as a little circle, which you can also set or remove while you write your code.



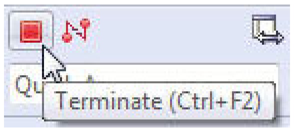
```
int main(int argc, char *argv[]) {
                                                 IDE\n");
```

Toggle Breakpoint Ctrl+Shift+B
Add Breakpoint... Ctrl+Double Click
Enable Breakpoint Shift+Double Click

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 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), select **Run** in the launch bar, and then click the Run button ( ).

You can also use the System Information perspective's Target File System Navigator (accessible through **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**).

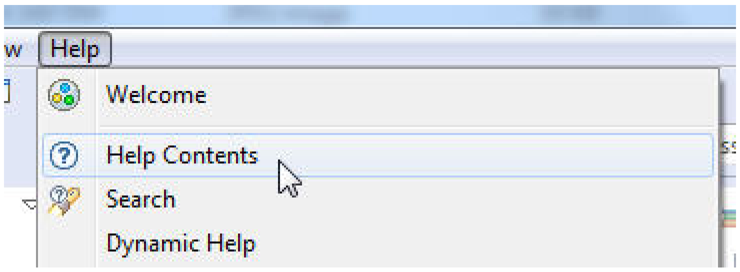
It's also possible to leave the binary on a shared network drive on your development host, mount the drive on your QNX Neutrino target (see the entry for `fs-cifs` in the QNX Neutrino *Utilities Reference*), and run the binary from there.

Making the program your own

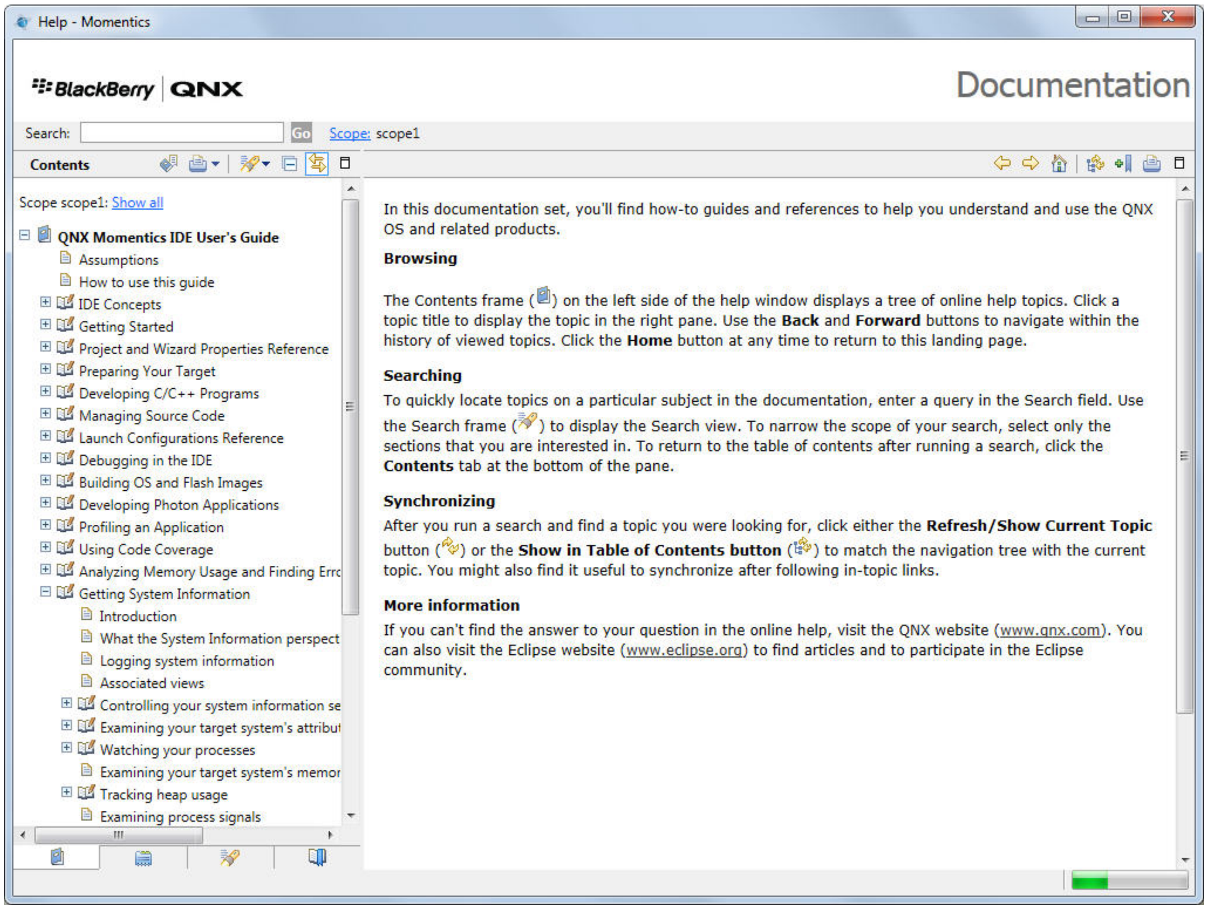
To turn this default program into your own first QNX Neutrino program, you can modify and extend the source code we just created. Try some of our sample programs and copy code from them into your project. And now that you've started, you'll probably want a lot more information, such as how to create 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*. If you've used earlier versions of QNX SDP, see also *Migrating to QNX SDP 7.0*.



Scope scope1: Show all

- QNX Momentics IDE User's Guide
  - Assumptions
  - How to use this guide
  - IDE Concepts
  - Getting Started
  - Project and Wizard Properties Reference
  - Preparing Your Target
  - Developing C/C++ Programs
  - Managing Source Code
  - Launch Configurations Reference
  - Debugging in the IDE
  - Building OS and Flash Images
  - Developing Photon Applications
  - Profiling an Application
  - Using Code Coverage
  - Analyzing Memory Usage and Finding Errc
  - Getting System Information
    - Introduction
    - What the System Information perspect
    - Logging system information
    - Associated views
    - Controlling your system information se
    - Examining your target system's attribut
    - Watching your processes
    - Examining your target system's memor
    - Tracking heap usage
    - Examining process signals

In this documentation set, you'll find how-to guides and references to help you understand and use the QNX OS and related products.

Browsing

The Contents frame ( ) on the left side of the help window displays a tree of online help topics. Click a topic title to display the topic in the right pane. Use the **Back** and **Forward** buttons to navigate within the history of viewed topics. Click the **Home** button at any time to return to this landing page.

Searching

To quickly locate topics on a particular subject in the documentation, enter a query in the Search field. Use the Search frame ( ) to display the Search view. To narrow the scope of your search, select only the sections that you are interested in. To return to the table of contents after running a search, click the **Contents** tab at the bottom of the pane.

Synchronizing

After you run a search and find a topic you were looking for, click either the **Refresh/Show Current Topic** button ( ) or the **Show in Table of Contents button** ( ) to match the navigation tree with the current topic. You might also find it useful to synchronize after following in-topic links.

More information

If you can't find the answer to your question in the online help, visit the QNX website (www.qnx.com). You can also visit the Eclipse website (www.eclipse.org) to find articles and to participate in the Eclipse community.

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!