Quickstart Guide

GNX

©2005–2014, QNX Software Systems Limited, a subsidiary of BlackBerry. All rights reserved.

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/

QNX, QNX CAR, Neutrino, Momentics, Aviage, and Foundry27 are trademarks of BlackBerry Limited that are registered and/or used in certain jurisdictions, and used under license by QNX Software Systems Limited. All other trademarks belong to their respective owners.

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 (p. 4)
- 2. Installing QNX SDP on the development host (p. 6)
- 3. Installing the QNX Neutrino RTOS on the target system (p. 7)
- 4. Networking with the QNX Neutrino RTOS (p. 8)
- 5. Creating a program project (p. 9)
- 6. Communicating with the QNX Neutrino RTOS (p. 12)
- 7. Compiling and linking (p. 14)
- 8. Preparing to launch the program (p. 17)
- 9. Starting and debugging (p. 19)
- 10. Making the program your own (p. 22)

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:



QNX Momentics Tool Suite

If you don't have the QNX Software Development Platform, you can download an evaluation version from www.gnx.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.gnx.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 procento 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
100: flags=8049 <up,loopback,running,multicast> mtu 33192</up,loopback,running,multicast>
inet 127.0.0.1 netmask 0xff000000
en0: flags=80008843 <up,broadcast,running,simplex,multicast,shim> mtu 150</up,broadcast,running,simplex,multicast,shim>
address: 00:0c:29:03:51:5a
media: Ethernet 10baseT full-duplex
status: active
inet 192.168.153.132 netmask 0xffffff00 broadcast 192.168.153.25

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 Deply from 192.168.153.132: bytes=32 time<1ms TTL=255
Reply from 192.168.153.132: bytes=32 time<1ms 11L=255
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss), Approximate round trip times in milli-seconds: Minimum = Oms, Maximum = Oms, Average = Oms

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 $New \rightarrow Project...$ In the New Project dialog, expand **QNX**, and then select **QNX C Project**:



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

New Project		
QNX New Proje This wizard creat	ct Wizard es a new QNX project	G
Project name:	ny_first_project	
Location: C:\Us	ers\stever\ide-5.0-workspace\my_first_project	Browse
Type Application Stand-alone exect Generate defa Working sets	m :utable which is supposed to be launched. sult file	v
Add project	to working sets	×] Select

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:

File Edit Source Relator Nevigete Search Project Run Window Help Image: Search Project Run Help Image: Search Project Run Help <	File Edit Source Reactor Navigets Search Project Run Window Help Image: Source Run Window Help Image: Source Reactor Navigets Search Project Run Window Help Image: Source Run Window Hel	C/C++ - my_first_project/my_first_project.c	- QNX Momentics IDE					
Image: Source and Source Development Platform 6.6 Image: Source Construct and Source Construct Cons	Image: Section of Section and Development History 6.6 Image: Section of Sec	File Edit Source Refactor Navigate S	earch Project Run Window	Help				
Image: Solution of the solution	Image: Solution in the second seco	📑 🕶 📰 🕼 🍙 🛛 🕶 🔨 🕶 🔜 QNX S	oftware Development Platform 6.6	5 - 🔍 🔌	💣 🕶 🚳	• 🖻 • 🞯 •	□□ - 参 - (D = 94 = 🍅 🛷 •
Project Explorer IX Image: Section in the section in t	Project Explorer IX Image: my_first_project.clip.h> Image: my_first_project.clip.h> Image: main(int argc, char "argv[]) { Image: my_first_project.clip.h> Image: main(int argc, char "argv[]); Image: main(int argc, char "argv[]); Image: main(int char(]):int argc, char "argv[]); Image: main(int argc, char "argv[]); Image: main(int char(]):int argc, char "argv[]); Image: main(int argc, char "argv[]); Image: main(int char(]):int argc, char "argv[]); Image: main(int char(]):int argc, char "argv[]); Image: main(int char(]):int argc, char "argv[]); Image: main(int argc, char "argv[]); Image: main(int char(]):int argc, char "argv[]); Image: main(int argc, char "argv[]); Image: main(int char(]):int argc, char "argv[]); Image: main(int argc, char "argv[]); Image: main(int argc, char "argv[]); Image: main(int argc, char "argv[]); Image: main(int argc, char "argv[]); Image: main(int argc, char "argv[]); Image: main(int argc, char "argv[]); Image: main(int argc, chargv[]); </th <th>3 ■ ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ●</th> <th>1</th> <th></th> <th></th> <th>Q</th> <th>uick Access</th> <th></th>	3 ■ ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ●	1			Q	uick Access	
		 Project Explorer S S S	my_first_project. wy_first_project. transformed and the state of the	> char *argv[]) { e to the QNX Momentics ccess; Ccess; Console Properties R	IDE\n"); esource	Path	Location	S M H stdlib.h stdlib.h p main(int, char*[]) : ir Type Type

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 gconn program. If you didn't see it earlier in the output of pidin, you can start gconn 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** \rightarrow **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:

lease specify the d	tails of your QNX target.	
Target Name		
Same as hostnar	ne	
Target Name: my	Nto_machine	
Connection		
Hostname or IP: 1	92.168.153.132	Port: 8000

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**.

Edit Source Refactor Navigate Search	Project Kun Wi	ndow	meip					
QNX Software Development	t Platform 6.6 🔻 🗔				🖳 + 💁 ·	• 😂 🔗 • 🌙	N + 🖓 + 🏠	\$ * \$ * B
							Quick Access	E 1
Target Navigator 🛛 🗖 🐻 S	System Summary 🔀	So, Pro	cess Info	ormati	Remory Inform	at 🐼 Malloc	Information 皆 Ta	arget File Syste
월 🚮 🗞 📮 🔹 🔍 🗸								
my_Nto_machine/localhost my_	Nto_machine - Last	Updated	:Thu Jan	16 16:14	:04 EST 2014			
6	system specification				CDUD	11-		
	Hostname: localnost				x86 @ 291	BMhz		
	soard: x80pc		21.04.25	TOT N				
	Jo version: 0.0.0 (201	4/01/10-	21:04:33	(11)				
	soot Date: Thu Jan It	11:07:08	EST 201	.4				
	System Memory							
U	sed: 50M Free: 206M	Total: 2	56M					
	Total Processes: 20					12		
	All Processes Appli	cation Pr	ocesses	Server	Processes			
	Process Name	Code	Data	Stack	Data Usage Delta	CPU Usage	CPU Usage Delta	Start Time
	procnto-smp-ins	628K	195	0	0	6m 53s 609	4s 976ms	Thu Jan 16 11:07:08
	slogger (2)	12K	96K	8192	0	3ms	0	Thu Jan 16 11:07:08
	dumper (3)	108K	160K	36K	0	2ms	0	Thu Jan 16 11:07:08
	pci-bios (4100)	52K	96K	8192	0	831ms	0	Thu Jan 16 11:07:08
	devb-eide (4101)	88K	39M	160K	0	317ms	3ns	Thu Jan 16 11:07:08
	io-usb (4102)	124K	168K	24K	0	3ms	5ns	Thu Jan 16 11:07:09
	io-hid (4103)	36K	128K	32K	0	9ms	0	Thu Jan 16 11:07:09
	devc-pty (4104)	52K	160K	8192	0	3ms	0	Thu Jan 16 11:07:09
	devc-con-hid (41	88K	132K	12K	0	149ms	0	Thu Jan 16 11:07:09
	devc-ser8250 (41	56K	128K	8192	0	4ms	0	Thu Jan 16 11:07:09
	inetd (45067)	44K	124K	8192	0	2ms	0	Thu Jan 16 11:07:12
	sh (4108)	152K	132K	8192	0	3ms	0	Thu Jan 16 11:07:09
	sh (4109)	152K	132K	8192	0	3ms	0	Thu Jan 16 11:07:09
	sh (4110)	152K	132K	8192	0	3ms	0	Thu Jan 16 11:07:09
	sh (4111)	152K	132K	8192	0	999us 979ns	0	Thu Jan 16 11:07:09
	io-pkt-v4-hc (41	128	640K	72K	0	556ms	22ms	Thu Jan 16 11:07:09
	pipe (4113)	20K	96K	44K	0	1ms	0	Thu Jan 16 11:07:09
	Contraction of the second second							
	•							

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):

🔐 Problems 🖉 Tasks 📮 Console 🛛 🗔 Properties 🛛 🕹 🏠 🔛 🔹 📬 🖛 🕄 🔹	- 0
CDT Build Console [my_first_project]	
<pre>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.0] Error 1 make.exe[2]: **** [my_first_project.0] Error 1</pre>	^
make.exe[1]: [all] Error 2 (ignored)	-

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:

errors, 2 warnings, 1 other					
Description	Resource	Path	Location	Туре	
a 🔞 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	
b 🚯 Warnings (2 items)					
i Infos (1 item)					

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** \rightarrow **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:

Q	NX C/C++ Project
	🧐 Options 🗱 Build Variants 📄 Gene
	Category Extra libraries
	Name
	航 m

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 <code>qconn</code> 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:

Debug Configurations			2
eate, manage, and run configu	rations) (
ŷ 📄 🗙 │ 🖻 🍰 ▼ type filter text	Name: my_first_project Configuration	onment @ Unload 🎋 Debugger 🗐 Source	e) 🔲 Common 🐺 Tools
C C/C++ Application	C/C++ Application:	onnent 🕑 opiota ip bebegger i ip boare	
C/C++ Postmortem Debugg	x86/o-g/my_first_project_g		
C/C++ QNX Attach to Remc C/C++ QNX PDebug (Serial) C/C++ QNX Postmortem De	Project:	Variables	rch Project Browse
🖉 C/C++ QNX QConn (IP)	my_first_project		Browse
my_first_project Configu	Build (if required) before launching		
GDB Hardware Debugging	Puild anoficiation	Castinuation	
Launch Group	build conliguration:	Select configuration	listian!
📴 QNX File Transfer		Select configuration using C/C++ App	Jication
	Enable auto build Itse workspace settings	O Disable auto build	ings
	Priority/Scheduling Algorithm Priority 10 - Scheduling SCHE	D_RR V	
	Ville terminal emulation on targe	*	
	Filter targets based on C/C++ A	oplication selection	
	my_Nto_machine (Neutrino/x	86)	Add New Target Remove Target Target Properties
ter matched 12 of 12 items	Using DSF Debugging Framework (N	ew) Launcher - <u>Select other</u>	Apply Revert
?			Debug Close

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.

Debug - my_first_project/my_first_project.c - QNX Momentics IDE	-		
File Edit Source Refactor Navigate Search Project Run Window Help			
📬 👻 📄 🕞 QNX Software Development Platform 6.6 💌 🗖 📚 🔖	₩ 12 A 🔊	3. D II I N	🖫 + 🎄 + 🜔 + 💁 +
☞ ৵ ▪ ☑ 옷 ↓ ▼ ☆ ▼ ☆ ▼ ☆ ▼ ↓		Quick Access	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
🏂 Debug 💥 🙀 🚺 🙀 😵 🔽 🗖	(X)= Variables 💥 💁 Br	eakpoints 👭 Registers 🛋	Modules 🗖 🗖
▲ 🧑 my_first_project Configuration [C/C++ QNX QConn (IP)]		倍 4 (3 🧇 8 x % 🔂 🖻 🔻 🔻
🔺 🎲 my_first_project_g [77844]	Name	Туре	Value
Thread [1] (STOPPED) (Suspended : Breakpoint)	(x)= argc	int	1
main() at my_first_project.c:5 0x8048/9a	> 🔿 argv	char **	0x8047eb4
C:\qnx0bU\host\win32\x8b\usr\bin\ntox8b-gdb.exe			
In the second se			
			* *
۲. III. ۲.	*		*
imy_first_project.c 🛛			tline 🛿 🗖 🗖
<pre>#include <stdlib.h></stdlib.h></pre>			🗖 🖡 🔊 🔊 e 💥 🗸
<pre>#include <stdio.h></stdio.h></pre>			stdlib.h
wint main(int accc, chan *army[]) {			stdio.h
<pre>printf("Welcome to the ONX Momentics IDE\n");</pre>			main(int, char*[]) : int
return EXIT_SUCCESS;			A 0 65
}			
		-	
Console 32 A Tasks R Problems O Executables 1 Memory		🔳 🗙 🖗 📄 🗔 🕅	
my first project Configuration IC/C++ ONV OConn (ID)) my first project a		- ee sel un Du G	
my_msc_project comganation (c/c++ Qrix Qcom (tr)) my_msc_project_g			
			-
4			5 E
	12 32 31		
Writable	Smart Insert 5 :	1	

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.

Toggle Breakpoint	Ctrl+Shift+B	IDE (n);
Add Breakpoints.	Ctrl+Double Click	1
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 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** \rightarrow **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** \rightarrow **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**:

w	Help		
]	3	Welcome	
	0	Help Contents	5
Þ	??	Search 6 Dynamic Help	

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*.

4 Help - QNX Momentics IDE				
# GINX		Documentation		
earch: Go <u>Scope:</u> All topics				
Contents	<u></u>			
	er Guide sment User Guide sment User Guide moentions, Support, and Licensin tation es JDE User's Guide Development Platform. Here you'll find a comprehensive library of titles to help you understand the OS and its tools so you can develop and deploy superior realtime embedded systems. If you're new to the QNX Neutrino [®] OS, you should start with the <u>System Architecture</u> guide. OS Core Components			
	Document	Description		
	Adaptive Partitioning User's Guide	Describes how to use adaptive partitioning to control the allocation of resources among competing processes.		
	Building Embedded Systems	Tells you how to get the OS running on your target embedded system, write an IPL, customize a startup program, etc.		
	Core Networking Stack User's Guide	Describes how to use io-pkt-* for networking on QNX Neutrino.		
	Device Publishers Developer's Guide	Provides a reference of all PPS objects written by device publishers and lists the command options you can set for publishers.		
	High Availability Framework Developer's Guide	How to build robust high-availability software running on the QNX Neutrino realtime operating system.		
	Instant Device Activation	How to set up a "minidriver" to start devices quickly when the system boots.		
HTTP FRROR: 500	Multicore Processing User's Guide	How to use symmetric multiprocessing to get the most performance possible out of a multiprocessor system.		
Done		· · · · · · · · · · · · · · · · · · ·		

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** \rightarrow **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.gnx.com). We want to be with you from the start, because we are successful only if you are!