Networking Source#

The most important thing to do first is to set up a staging directory to allow a) the source to build and b) prevent the build / install from overlaying your "standard" installation. This is always good practise when building QNX source but is particularly important when building the networking tree since a 'make install' generally has to be done from the root to build components in the correct order to satisfy dependencies.

This link, OS Source Build, covers some **Very Important Details**, most notably the description of creating a staging area and a qconf-override.mk file (See steps 2,3 and 4). Just to re-emphasize the importance of this... If you don't properly set things up, you stand a good chance of overwriting your base installation and that would not be a good thing.

We recommend that you build the source with a 6.4.0 installation. The source code minimally requires the Momentics 6.3.2 development suite. Although we don't officially support the new core networking product on 6.3.2, we are still making every effort to ensure that people who are using it on 6.3.2 can continue to do so. The 6.3.0 upgrade versions of Momentics (6.3.0 SP1, SP2 or SP3 plus the 6.3.2 Core OS upgrade) don't give you everything that you need to build successfully. Of particular importance are updates to some of the make files to properly handle SVN meta-info when you do a "make install" (Yup... If you see all of those "svn" files getting copied into your stage, then you aren't running the proper version of Momentics).

Just a couple of notes on the QCONF_OVERRIDE environment variable. On windows, in a DOS shell, use the SET command (there is no export). You can make the env var "stick" by right clicking on "My Computer" (select Properties). Select the "Advanced" tab and click on the "Environment Variables" button (bottom left in XP). Click on the "New" button and add in QCONF_OVERRIDE and the value and "OK" your way out and, the next time you start a DOS shell, QCONF_OVERRIDE will already be set up for you. If you're running self-hosted (or under Linux), you can add the export command to your .profile script and it will also automatically get set when you start a new terminal session.

Source Tree Organization#

The source is laid out with a top level similar to other projects. There is the "trunk" directory which is where the most up to date source code lives that is being worked on. We expect that all code in the trunk directory will work its way into the general product at some point in time. At any point in time, we can't guarantee that the trunk source will a) build and b) if it *does* build, work. While we will be making every effort to ensure that the trunk source will indeed build and operate, we all make mistakes from time to time, so I'm sure that code will slip in from time to time that won't have everyone's best interests in mind. Hey, that's what you get when you sign up to live development!

The branches directory contains stable release code that a) builds and b) DOES work. This is where source code used for official releases will live. It may also contain source for highly experimental development which would adversely affect the trunk branch or "specialized" code which may not be included as part of general product. For 6.4.0, a maintenance release branch (branches/6.4.0/trunk)) has been created. This branch contains the source used to create 6.4.0 plus critical maintenance fixes (which will be released in upcoming 6.4.0.x upgrade patches).

The tag directory is used to "mark" source that exists in another location (branch or trunk). Integration ("smoke test") builds, milestone builds, release candidate builds and, in fact, anything that we post "official" binaries for will have a corresponding entry in the tags branch so that we can full identify the source base used to make the binaries. It may also contain specific source for priority support builds or other such "non-standard" builds. The tags directory is not a development directory. It won't contain any code that doesn't exist in either the branch or trunk. The exact source used to create the 6.4.0 release is located under tags/6.4.0/GA.

Some relevant portions of the tree are:

- **/trunk/**

- **lib** - Libraries
    - **crypto** - OpensSSL libcrypto build files / directory
    - **ipsec** - libipsec code from ipsec-tools
    - **nbutil** - NetBSD utilities library (used for porting NetBSD code)
    - **pcap** - Packet Capture library
    - **socket** - Socket library for networking applications
    - **ssl** - OpenSSL libopenssl build files / directory
    - **wlconfig** - QNX WiFi configuration library abstraction layer
- **services** - Networking daemons and other service
- **sys** - Main code base for the stack
    - **dev** - NetBSD ported drivers
    - **dev_qnx** - Native Drivers
    - **target** Stack build directories
    - **kern** -
    - **lib** - Build directory for stack libraries
    - **lkm** - stack "kernel" modules
        - **net**
            - **pf**- Packet Filter build directory
    - **lsm** - stack "shared object" modules
        - **autoip** - Auto IP configuration module
        - **qnet** - QNET / Transparent Distributed Processing source
        - **nraw** - Sample resource manager / filter implementation showing how to filter raw packets out of the stack
    - **net**
    - **net80211** - WiFi layer stack source
    - **netinet**
    - **netipsec** - I~PSec stack source
    - **netkey**
    - **secmodel**
- **utils** - Networking utilities (Grouped alphabetically)

How do I get the source?#

The source code is available on the Networking repository located on http://community.qnx.com/sf/scm/do/listRepositories/projects.networking/scm

To download the lateste source code into your source directory:

svn checkout --username <userid> http://community.qnx.com/svn/repos/core_networking/trunk

Where **<userid>** is the email address used to create your account on the QNX site.

To check out the 6.4.0 GA release branch source, use

svn checkout --username <userid> http://community.qnx.com/svn/repos/core_networking/tags/6.4.0/GA

To check out the 6.4.0 maintenance branch source (which includes critical fixes to the GA release), use

svn checkout --username <userid> http://community.qnx.com/svn/repos/core_networking/branches/6.4.0/trunk

But where's the io-net code?#

The io-net code is in "maintenance only" mode with no new features planned as part of a GA release. We will continue to be doing maintenance patches as required, but new features and networking capabilities are being released in the Core Network 6.4 version of the stack. As such, the source to io-net and related protocols will not be part of this project.

How do I build the source?#

If you don't properly create a staging area and suitable qconf-override.mk file, you risk the chance of corrupting your standard installation. ("What I tell you three times is true." : Lewis Carroll)

All versions prior to 6.4.2

The latest <unistd.h> is needed for the getpeereid() prototype. This can be obtained by doing a 'make hinstall' of lib/c from the core os project. Alternatively you can add it manually to the version in your staging area.

Neutrino 6.3.2

If you haven't already done so, install the srcversion patch for QNX 6.3.2 from the download section of the core os project as root.

```
# cd $QNX_TARGET/../..
# tar -xpf srcversion-patch-6.3.2.tar
```

From the trunk directory of where you have the core_networking repo checked out:

- make install

Or if you prefer:

- make CPULIST=x86 install

Neutrino 6.4.0

If you haven't already done so, install the srcversion patch for QNX 6.4.0 from the download section of the core os project as root.

```
# cd $QNX_TARGET/../..
# tar -xpf srcversion-patch-6.4.0.tar
```

On a 6.4.0 system, you need to specify the CPU variants since support for some of the CPU architectures was deprecated. To build all supported targets, type:

- make CPULIST="x86 ppc arm sh" install

That's it! All the associated header files for the build and binaries created will get installed in your staging directory.

One important point to note: Some of the utilities (e.g. "ping") need to have root privileges in order to run (they do things like open raw sockets, for example). In order for these utilities to be given proper permissions under QNX or Linux you need to change the "op" utility to be "suid" (i.e. run at the privilege level of the owning user). By default in Neutrino, op is owned by root but isn't setuid since this is considered to be a security issue. To do this on Neutrino, log in as root and simply "chmod u+s /usr/bin/op" (on Linux, you need to figure out where the "op" utility lives). The other thing that you can do is perform the initial "make install" as a standard user (to ensure that your staging area is properly set up and that the install isn't going to overwrite files that it shouldn't) and do a final "make install" as root. Standard operating procedure is NOT to do ANYTHING casually as root since this can often have undesired consequences.

Note: building from the top level directory builds everything in the correct order so that all dependencies are satisfied from your stage. Simply doing a (cd <subcomponent> && make) probably won't work until you've built the world at least once.

Where can I get pre-built binaries?#

If you don't want the latest and greatest source base, you can grab a set of pre-built binaries from the Project Downloads Page. This page is where we'll be placing integration builds (raw builds that might not have undergone much, if any, testing), Milestone builds (which have had, at a minimum, sanity testing performed) and Release Candidate builds (which have had full regression and feature testing done on them).

How do I use what I built?#

If you're running self hosted, the easiest thing to do is to modify your PATH and LD_LIBRARY_PATH environment variables to point to the appropriate build directories before going to the "normal" location. Don't forget to slay io-net before starting io-pkt.

Here's a sample script (iopkt.sh) that you can cut and paste to change these variables for you. It assumes that your staging area is in $HOME/stage:

```
#Execute with ". ./iopkt.sh"
#!/bin/sh
export LD_LIBRARY_PATH=$HOME/stage/x86/lib:$HOME/stage/x86/lib/dll:$LD_LIBRARY_PATH
export PATH=$HOME/stage/x86/bin:$HOME/stage/x86/sbin:$HOME/stage/x86/usr/bin:$HOME/stage/x86/
usr/sbin:$PATH
```

On embedded targets, you need to include the appropriate binaries from the given CPU directory in your image. At a minimum, you will need:

- io-pkt-v4-hc (or whatever variant you want)
- libsocket.so
- ifconfig
- Relevant driver (plus devnp-shim.so if you're running an io-net driver binary)

Back to Main Wiki#