

GnuRadio Installation Notes

GnuRadio Website - <http://gnuradio.org/trac/wiki>

The packages for download can be found at
<ftp://ftp.gnu.org/gnu/gnuradio/gnuradio-3.0.2.tar.gz>

Or from SVN repository
svn co <http://gnuradio.org/svn/gnuradio/trunk> gnuradio

Document History:

Version	Modified/Creation date	Description
1.0	3/16/2005	Created document for GNURadio 2.4
1.1	6/1/2005	* Added notes for USRP board installation. * Updated to GNURadio 2.5
1.2	6/16/2005	* Updated Python and wxPython Installation notes. * Added Dell Truemobile Wireless card NDISWRAPPER installation instructions.
1.3	5/22/2006	* Updated to GNURadio 2.8 * Added Troubleshooting * Added gmsk2 examples explanation * Added CVS install instructions
1.4	11/20/2006	* Updated to GNURadio 3.0 * Updated tunnel example explanation * Added SVN install instructions

Here are some notes on the installation of GnuRadio ver.3.0

The list of available packages are: (New packages keep getting added to the repository)

- [gnuradio-core] The main library. Contains the underlying runtime system and most of the hardware independent signal processing blocks.
- [gnuradio-examples] Simple examples that exercise GNU Radio.
- [gr-audio-alsa] Support for sound cards using the ALSA (Preferred on GNU/Linux).
- [gr-audio-jack] Support for sound cards using the JACK (experimental: GNU/Linux?).
- [gr-audio-oss] Support for sound cards using the Open Sound System.
- [gr-audio-portaudio] Preliminary support for sound cards using the portaudio V19 library (work-in-progress).
- [gr-audio-osx] Support for OSX audio
- [gr-audio-windows] Support for audio-sink to gnuradio for windows.
- [gr-wxgui] GUI framework built on wxPython. Includes blocks for displaying realtime FFT and oscilloscope.
- [gr-gsm-fr-vocoder] GSM 06.10 13kbit/sec voice encoder/decoder.
- [gr-radio-astronomy] Radio astronomy application files
- [gr-trellis] Implementation of trellis-based encoding and decoding algorithms
- [gr-howto-write-a-block] Documentation and examples of how to write a new signal processing block in GNU Radio. Can be downloaded from <ftp://ftp.gnu.org/gnu/gnuradio/gr-howto-write-a-block-3.0.2.tar.gz>
- [usrp] The non-GNU Radio specific part of the Universal Software Radio Peripheral code base. This contains the host libs, firmware and fpga code.
- [gr-usrp] The glue that ties the usrp library into GNU Radio.

How to Build GNU Radio:

(1) Ensure that you've satisfied the external dependencies listed below.

With the exception of SDCC, the following GNU/Linux distributions are known to come with all required dependencies pre-packaged: Ubuntu 6.06, SuSE 10.0 (the pay version, not the free download), Fedora Core 2,3,4,5. Other distributions may work too. The required packages may be contained on your installation CD/DVD, or may be loaded over the net. The specifics vary depending on your GNU/Linux distribution.

On systems using pkgsrc (e.g. NetBSD/Dragonfly), build meta-packages/gnuradio, which will build a previous release and force installation of the dependencies. Then `pkg_delete` the gnuradio and usrp packages, which will leave the dependencies.

See the wiki at <http://gnuradio.org/trac/wiki> for more details.

(2) Do the following steps if installing from the SVN repository

```
$ ./bootstrap      # not reqd when building from the tarball
$ ./configure
$ make && make check
$ sudo make install
```

If any package is missing, `./configure` will print out the error message with the filename.

Brief instructions for installing the tarball packages: (I work on Fedora Core, So I have listed only packages I needed to install, to get GNURadio up and running)

1. [gnuradio-core]:

Prerequisites:

(1) The "autotools"
autoconf 2.57 or later
automake 1.7.4 or later
libtool 1.5 or later

If your system has automake-1.4, there's a good chance it also has automake-1.7 or later. Check your install disk and/or (on GNU/Linux) try:

```
$ man update-alternatives
```

for info on how some distributions support multiple versions.

(2) pkgconfig 0.15.0 or later <http://www.freedesktop.org/Software/pkgconfig>

From the web site:

pkgconfig is a system for managing library compile/link flags that works with automake and autoconf. It replaces the ubiquitous *-config scripts you may have seen with a single tool.

(3) FFTW 3.0 or later <http://www.fftw.org>

IMPORTANT!!! When building FFTW, you MUST use the --enable-single and --enable-shared configure options. This builds the single precision floating point version which we use. You should also use either the --enable-3dnow or --enable-sse options if you're on an Athlon or Pentium respectively.

(4) Python 2.3 or later <http://www.python.org>

Python 2.3 or later is now required. If your distribution splits python into a bunch of separate RPMS including python-devel or libpython you'll most likely need those too.

- `./configure --enable-unicode=ucs4`
`make clean`
`make`
`make install`

(5) Numeric python library <http://numeric.scipy.org>

Provides a high performance array type for Python.

http://sourceforge.net/project/showfiles.php?group_id=1369&package_id=1351

(6) The Boost C++ Libraries <http://www.boost.org>

We use the Smart Pointer library. Fedore Core 2 has a package for this, boost-devel-1.31.0-7. Otherwise download the source and follow the build instructions. They're a bit different from the normal `./configure && make`

(7) cppunit 1.9.14 or later. <http://cppunit.sourceforge.net>

Unit testing framework for C++.

Some of the other utilities which may be required are:

(8) SWIG – Simplified Wrapper and Interface Generator

<http://www.swig.org/>

These versions are known to work: 1.3.23, 1.3.24, 1.3.25, 1.3.27, 1.3.28, 1.3.29

(9) SDCC – Small Device C Compiler

<http://sdcc.sourceforge.net/>

Use version 2.4.0 or later.

This includes a C compiler and linker for the 8051. It's required to build the firmware for the USRP. If you don't have a USRP, don't worry about it.

Optional, but nice to have:

(10) wxPython. Python binding for the wxWidgets GUI framework.

Use version 2.5.2.7 or later. Again, almost all systems have this available.

As a last resort, build it from source (not recommended!)

<http://www.wxpython.org>

Also, as noted on the website, GNU Radio exercises bugs in certain versions of g++ 3.3.x on the x86 platform. If you are using g++ 3.3 and make check fails, please either upgrade to 3.4 or downgrade to 3.2. Both are known to work.

2. [gnuradio-examples]

Set your PYTHONPATH environment variable so that the GNU Radio toolkit and optional packages can be found by python.

PYTHONPATH should include the path of the local site-packages directory.

If the above packages were installed using the default prefix (/usr/local) and you're using python 2.3, this should work:

```
$ export PYTHONPATH=/usr/local/lib/python2.4/site-packages
```

You may want to add this to your ~/.bash_profile or similar file.

Once PYTHONPATH is set, you should be able to run any of the examples for which you have the required i/o devices.

To ensure that your setup is sane, try this:

```
$ python
>>> from gnuradio import gr
```

If this works, your PYTHONPATH is set correctly.

3. [gr-audio-alsa] & [gr-audio-oss]

These 2 packages are audio packages which are needed to interface with the audio device on your computer.

4. [gr-howto-write-a-block] : The documentation with descriptions of how to write signal processing blocks for GNU Radio

If you've got doxygen installed and provide the --enable-doxygen configure option, the build process creates documentation for the class hierarchy etc. Point your browser at gnuradio-core/doc/html/index.html

The online version can be found at :

<http://www.gnu.org/software/gnuradio/doc/howto-write-a-block.html>

5. The online documentation for GNU Radio with descriptions of all the modules, class

hierarchy and file list can be found at : <http://www.gnu.org/software/gnuradio/doc/>

6. [gr-wxgui] : GUI framework built on wxPython. Includes blocks for displaying realtime FFT and oscilloscope. The modules wxPython and Numerical Python also need to be installed.

http://wiki.wxpython.org/index.cgi/Getting_Started is a good place to look for information about wxPython and its installation guidelines.

A short summary of the instructions is as below:

- a. Install Python (<http://www.python.org/download/releases/2.4.4/>) (This is the version I used)
- b. Install wxPython: (requires glib and gtk+ libraries installed - <http://www.wxpython.org/download.php#prerequisites>)
Download the source code of the last wxPython release: [wxPython website](#).
The website has listings for different platforms. There is no separate version for Fedora Core 3, instead use the version for Fedora Core 2, it works.

The default installation happens in “/usr/lib/python<version>/site-packages/”. But the Python installation happens in “/usr/local/lib/python<version>/site-packages/”. This may prevent the “wx” module from being accessed correctly. To solve this, in “/usr/local/lib/python<version>/site-packages/” create the path file “wx.pth” and in that file give the full path to the wxPython installation i.e “/usr/lib/python<version>/site-packages/wx-<unicode-version>”. This will enable python to find the wx module.

- c. Test the installation with a small program in python:

```
import wx
app = wx.PySimpleApp()
frame = wx.Frame(None, -1, "Hello World")
frame.Show(1)
app.MainLoop()
```

Here is what you should get with wx:



After importing wxPython GUI, we instantiate a new wxPySimpleApp and a new wxFrame. A frame in wxPython is a window with its titlebar, reduction and close buttons, etc... We make this Frame appear by "showing" it. Eventually, we start the application's MainLoop whose role is to handle the events.

- d. Install Numerical Python: Numerical Python adds a fast array facility to the Python language (<http://numeric.scipy.org/>).

Download it from <http://sourceforge.net/projects/numpy> . The install happens within the python installation directory, hence requires no changes to the path file.

To verify, look under “/usr/local/lib/python<version>/site-packages/” and ensure that the files “Numeric.pth” and “wx.pth” exist and contain valid paths.

<http://www.pfdubois.com/numpy/html2/numpy.html> gives a nice reference for programming with Numerical Python.

- e. Install NumArray: http://www.stsci.edu/resources/software_hardware/numarray
Download it from [Sourceforge Numarray Download Page](#)

7. [usrp] : The non-GNU Radio specific part of the Universal Software Radio Peripheral code base. This contains the host libs, firmware and fpga code.

8. [gr-usrp] : The glue that ties the usrp library into GNU Radio.
The USRP hardware needs to setup and checked for correct operation.

<http://comsec.com/wiki?UsrpInstall> gives a quick walkthrough to check the working of the USRP board.

USRP hardware schematics and associated files can be obtained with the following command:

```
$ svn co http://gnuradio.org/svn/usrp-hw/trunk usrp-hw
```

- **Running the example files:**

To run examples, “cd” into the directory “gnuradio-examples/python/audio”. You should find some *.py files which can be run as executables. Run “./dialtone.py”. It should produce a dial tone of frequency 32KHz.

Also in the "usrp" subdirectory. Run ./usrp_oscope.py . This should bring up something that looks like an oscilloscope. Grab the corner of the window and resize it so you can read the labels on the buttons. It looks pretty dull until you get the triggering working -- set it from "Pos" to "Auto". Then you'll start seeing a bunch of noise on the screen -- a red line and a green line. If you attach a piece of wire to the inner conductor of your "RX-A" input on your "RXA" daughterboard, the green line will start to wiggle a lot.

A more interesting example is testing the FM reception. Even without a tuner, the ADC's on the USRP should be able to digitize a narrowband signal in the range up to

about 200MHz, by adjusting the built-in digital downconverter (DDC). Think of a strong broadcast FM station in your area, and try to receive it:

```
./ usrp_wfm_rcv.py --freq 104.5
```

The argument is the station's center frequency, in megahertz. A window will pop up which will show the signal at various stages of processing; and the radio station should be audible on your computer's speakers.

- **Digital Communication examples: (.../gnuradio-examples/python/digital)**

GNURadio uses the universal TUN/TAP drivers to tunnel the packets from the USRP via USB to the kernel.

Universal TUN/TAP Driver (<http://vtun.sourceforge.net/tun/> and [/usr/src/linux/Documentation/networking/tuntap.txt](http://usr/src/linux/Documentation/networking/tuntap.txt))

1. Description

TUN/TAP provides packet reception and transmission for user space programs. It can be seen as a simple Point-to-Point or Ethernet device, which, instead of receiving packets from physical media, receives them from user space program and instead of sending packets via physical media writes them to the user space program.

In order to use the driver a program has to open `/dev/net/tun` and issue a corresponding `ioctl()` to register a network device with the kernel. A network device will appear as `tunXX` or `tapXX`, depending on the options chosen. When the program closes the file descriptor, the network device and all corresponding routes will disappear.

Depending on the type of device chosen the userspace program has to read/write IP packets (with `tun`) or ethernet frames (with `tap`). Which one is being used depends on the flags given with the `ioctl()`.

The package from <http://vtun.sourceforge.net/tun> contains two simple examples for how to use `tun` and `tap` devices. Both programs work like a bridge between two network interfaces.

`br_select.c` - bridge based on `select` system call.

`br_sigio.c` - bridge based on `async io` and `SIGIO` signal.

However, the best example is VTun <http://vtun.sourceforge.net> :))

2. Configuration

Create device node:

```
mkdir /dev/net (if it doesn't exist already)
```

```
mknod /dev/net/tun c 10 200
```

Set permissions:

```
e.g. chmod 0700 /dev/net/tun
```

if you want the device only accessible by root. Giving regular users the right to assign network devices is NOT a good idea. Users could assign bogus network interfaces to trick firewalls or administrators.

Driver module autoloading

Make sure that "Kernel module loader" - module auto-loading support is enabled in your kernel. The kernel should load it on first access.

Manual loading

insert the module by hand:
`modprobe tun`

If you do it the latter way, you have to load the module every time you need it, if you do it the other way it will be automatically loaded when `/dev/net/tun` is being opened.

3. Program interface

- o Network device allocation:

`char *dev` should be the name of the device with a format string (e.g. `"tun%d"`), but (as far as I can see) this can be any valid network device name. Note that the character pointer becomes overwritten with the real device name (e.g. `"tun0"`)

```
#include <linux/if.h>
#include <linux/if_tun.h>

int tun_alloc(char *dev)
{
    struct ifreq ifr;
    int fd, err;

    if( (fd = open("/dev/net/tun", O_RDWR)) < 0 )
        return tun_alloc_old(dev);

    memset(&ifr, 0, sizeof(ifr));

    /* Flags: IFF_TUN - TUN device (no Ethernet headers)
     *        IFF_TAP - TAP device
     *
     *        IFF_NO_PI - Do not provide packet information
     */
    ifr.ifr_flags = IFF_TUN;
    if( *dev )
        strncpy(ifr.ifr_name, dev, IFNAMSIZ);

    if( (err = ioctl(fd, TUNSETIFF, (void *) &ifr)) < 0 ){
        close(fd);
        return err;
    }
}
```

```
    }  
    strcpy(dev, ifr.ifr_name);  
    return fd;  
}
```

- Frame format:

If flag IFF_NO_PI is not set each frame format is:

Flags [2 bytes]

Proto [2 bytes]

Raw protocol(IP, IPv6, etc) frame.

Universal TUN/TAP device driver Frequently Asked Question.

1. What platforms are supported by TUN/TAP driver ?

Currently driver has been written for 3 Unices:

Linux kernels 2.2.x, 2.4.x

FreeBSD 3.x, 4.x, 5.x

Solaris 2.6, 7.0, 8.0

2. What is TUN/TAP driver used for?

As mentioned above, main purpose of TUN/TAP driver is tunneling.
It is used by VTun (<http://vtun.sourceforge.net>).

Another interesting application using TUN/TAP is pipsecd
(<http://perso.enst.fr/~beyssac/pipsec/>), an userspace IPSec
implementation that can use complete kernel routing (unlike FreeS/WAN).

3. How does Virtual network device actually work ?

Virtual network device can be viewed as a simple Point-to-Point or
Ethernet device, which instead of receiving packets from a physical
media, receives them from user space program and instead of sending
packets via physical media sends them to the user space program.

Let's say that you configured IPX on the tap0, then whenever
the kernel sends an IPX packet to tap0, it is passed to the application
(VTun for example). The application encrypts, compresses and sends it to
the other side over TCP or UDP. The application on the other side decompresses
and decrypts the data received and writes the packet to the TAP device,
the kernel handles the packet like it came from real physical device.

4. What is the difference between TUN driver and TAP driver?

TUN works with IP frames. TAP works with Ethernet frames.

This means that you have to read/write IP packets when you are using tun and
ethernet frames when using tap.

5. What is the difference between BPF and TUN/TAP driver?

BPF is an advanced packet filter. It can be attached to existing network interface. It does not provide a virtual network interface.

A TUN/TAP driver does provide a virtual network interface and it is possible to attach BPF to this interface.

6. Does TAP driver support kernel Ethernet bridging?

Yes. Linux and FreeBSD drivers support Ethernet bridging.

Tunnel.py (Communication example which TUN/TAP adapters)

This program provides a framework for building your own MACs. It creates a "TAP" interface in the kernel, typically gr0, and sends and receives ethernet frames through it. See /usr/src/linux/Documentation/networking/tuntap.txt and/or Google for "universal tun tap". The Linux 2.6 kernel includes the tun module, you don't have to build it. You may have to "modprobe tun" if it's not loaded by default. If /dev/net/tun doesn't exist, try "modprobe tun".

To run this program you'll need to be root or running with the appropriate capability to open the tun interface. You'll need to fire up two copies on different machines. Once each is running you'll need to ifconfig the gr0 interface to set the IP address.

This will allow two machines to talk, but anything beyond the two machines depends on your networking setup. Left as an exercise...

On machine A:

```
$ su
# ./tunnel.py --freq 423.0M --bitrate 500k
## in another window on A, also as root...
# ifconfig gr0 192.168.200.1
```

On machine B:

```
$ su
# ./tunnel.py --freq 423.0M --bitrate 500k
## in another window on B, also as root...
# ifconfig gr0 192.168.200.2
```

Now, on machine A you should be able to ping machine B:

```
$ ping 192.168.200.2
```

and you should see some output for each packet in the tunnel.py window if you used the -v option.

Likewise, on machine B:

```
$ ping 192.168.200.1
```

This now uses a carrier sense MAC, so you should be able to ssh between the machines, web browse, etc.

- **Troubleshooting:**

- *AttributeError: 'PySwigObject' object has no attribute 'max_streams'*

This error stumped me for quite a while, when I upgraded to a new GNURadio version. Later it turned out be a problem with SWIG installation and I just had to reinstall SWIG to get rid of the error.

```
$ cd gnuradio-core/src/lib/swig
$ make clean
$ make install
```

- *Another handy trick if for example your fftw includes and libs are installed in, say ~/local/include and ~/local/lib, instead of /usr/local is this:*

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/local/lib
$ make CPPFLAGS="-I$HOME/local/include"
```

SVN Install:

The new repository organization simplifies a lot of the build system. You no longer need to go into the individual directories and compile separately.

To checkout the latest code from the development trunk, enter this on the command line:

```
$ svn co http://gnuradio.org/svn/gnuradio/trunk gnuradio
```

To instead checkout the latest stable release code, enter this on the command line:

```
$ svn co http://gnuradio.org/svn/gnuradio/branches/releases/3.0 gnuradio
```

First, ensure that you've fulfilled the dependencies specified in the top-level [README](#). Most GNU/Linux systems come with our dependencies already packaged. You may need to install them off of your install CD/DVD or over the net. See below for Operating System specific notes.

To compile, there are 5 steps. Start by cd'ing to the gnuradio directory, then complete the following commands:

```
$ ./bootstrap      # Do NOT perform this step if you are building from a tarball.
$ ./configure
$ make
$ make check
$ sudo make install
```

This will perform all configuration checks and select for build, test, and installation all components that pass.

For finer control, read the instructions at [BuildConfiguration](#).

Current Known Build Problems**#50**

make -j2 on Cygwin fails when make -j1 completes (gnuradio-core)

#99

Errors building gnuradio-3.0.2 using MinGW/MSYS

DELL TrueMobile Wireless Adapter:

The package was installed and tested on Linux Fedora Core 2 (kernel version 2.6.10-1.771_FC2), Fedora Core 3 (kernel version 2.6.9-1.667) AND Fedora Core 4 (kernel version 2.6.16-1.2111_FC4). Fedora Core 2, 3 & 4 recognized most of my system hardware and also my wireless PCI card.

The Dell Laptop's may have a problem with the Dell TrueMobile wireless PCMCIA cards, since there is no linux driver for those cards.

The way to activate these cards is by using the NDISWRAPPER (<http://ndiswrapper.sourceforge.net/>). The ndiswrapper package can be downloaded from <http://sourceforge.net/projects/ndiswrapper/> and the installation instructions can be found at <http://ndiswrapper.sourceforge.net/mediawiki/index.php/Installation>.

Follow all the steps as is, expect the "step 4: Configure interface", the easier way to do it is first identify the interface id by running "iwconfig". Suppose it is "wlan0",

```
vi /etc/sysconfig/network-scripts/ifcfg-eth1
```

```
DEVICE=eth1
BOOTPROTO=dhcp
ONBOOT=yes
TYPE=Wireless
ESSID=linksys
KEY=5136AS7F81
CHANNEL=1
GATEWAY=192.168.2.1
(Choose the appropriate ESSID, KEY and GATEWAY)
```

Then automate the ndiswrapper module to load on boot, "ndiswrapper -m" as listed in Step 5.

Other versions of Linux may require more pre-requisite modules to be installed.

Any version of Linux can be used, there is no recommended version of Linux. Newer versions have most of the packages needed preinstalled, whereas the older ones may require a manual install of the packages.

Updates listed on the webpage:**Current Defects:**

- [#29](#) templates are not expanded automatically in gr-trellis/src/lib
- [#31](#) plot.py _ticks routine picks bad strategy for producing tick mark labels
- [#44](#) portaudio build/link problems on Windows/Cygwin
- [#48](#) abort when no device specified on portaudio source or sink; Cygwin std::string problem
- [#50](#) make -j2 on Cygwin fails when make -j1 completes (gnuradio-core)
- [#56](#) make check in gnuradio-core fails
- [#71](#) audio.py doesn't look for audio_windows
- [#78](#) USRP compile using SDCC 2.6 fails on Intel-Mac
- [#85](#) Memory tests fail on NetBSD
- [#99](#) Errors building gnuradio-3.0.2 using MinGW/MSYS
- [#100](#) u_long and u_char not defined in MinGW, used in gr-error-correcting-codes
- [#101](#) sys/mman.h not defined on MinGW, used in gr-radar
- [#103](#) gr_throttle does nothing on MinGW; patch attached
- [#104](#) gr_oscope_guts.cc outputs initial garbage; patch attached

Current Enhancements:

- [#4](#) make doc should traverse whole tree
- [#5](#) Generate python-centric docs from Doxygen xml output
- [#9](#) Speed up compilation of gnuradio_swig_python.cc
- [#49](#) Add --with-prereqs or similar to configure
- [#57](#) on OSX, make ("ar") breaks when path name contains whitespace characters
- [#58](#) gr-radar doesn't perform any qa tests
- [#66](#) hw/sw closed loop AGC
- [#67](#) Standardized digital transmit levels for USRP
- [#69](#) Split up gnuradio.i / gnuradio_swig_python.cc to shorten incremental compilations
- [#72](#) Try shortening preamble from 32-bits to 8 or 16.
- [#73](#) Consider supporting flowgraphs with cycles
- [#74](#) Consider NOT requiring Python
- [#75](#) rewrite hierarchical blocks so that they are first class objects
- [#86](#) Get jcooley's OpenGL waterfall display working again
- [#89](#) Integrate gr-rds component into SVN trunk
- [#90](#) Better error message on USB version mis-matching
- [#91](#) Write plotting widget in C++ to improve performance
- [#102](#) examples that import powermate fail on MinGW
- [#105](#) Allow either NumPy or Numeric to be used

Current Tasks:

[#62](#) copy www.gnu.org web site to gnuradio.org

Discussion Mailing Lists:

<http://www.gnu.org/software/gnuradio/maillinglists.html>

Suggested Projects:

Make Windows port work as well as Linux

Full duplex audio

Faster USRP transfers

Easier to build

Audio on OS X

Portaudio

More example programs

OFDM

D-Star

MIMO

Ham radio demos

USRP/RFX IQ balance correction

USRP/RFX AGC

Better plotting widgets (OpenGL?)

Improved FFT, Scope, Waterfall

IQ plots

Eye diagrams

Collect all code out there that hasn't gotten into repository

DAB

POCSAG

GPS

802.11

IIR filter design

More FIR filter design code

GUI for putting together flowgraphs

Documentation

Logo

Website improvements

More wiki pages

Automated binary package generation