**Build and install GNSS-SDR**

**ON THIS PAGE**

* [**INSTALLING EVERYTHING THROUGH SOFTWARE PACKAGE MANAGERS**](https://gnss-sdr.org/build-and-install/#installing-everything-through-software-package-managers)
  + [GNU/LINUX DISTRIBUTIONS BASED ON DEBIAN](https://gnss-sdr.org/build-and-install/#gnulinux-distributions-based-on-debian)
  + [MACOS / MAC OS X USING MACPORTS](https://gnss-sdr.org/build-and-install/#macos--mac-os-x-using-macports)
* [**BUILDING FROM SOURCE**](https://gnss-sdr.org/build-and-install/#building-from-source)
  + [INSTALL DEPENDENCIES](https://gnss-sdr.org/build-and-install/#install-dependencies)
    - [INSTALL ALL DEPENDENCIES USING PACKAGES](https://gnss-sdr.org/build-and-install/#install-all-dependencies-using-packages)
    - [INSTALL DEPENDENCIES USING PYBOMBS](https://gnss-sdr.org/build-and-install/#install-dependencies-using-pybombs)
  + [CLONE, BUILD AND INSTALL GNSS-SDR](https://gnss-sdr.org/build-and-install/#clone-build-and-install-gnss-sdr)
* [**ALTERNATIVE TOOLS**](https://gnss-sdr.org/build-and-install/#alternative-tools)
  + [USING NINJA](https://gnss-sdr.org/build-and-install/#using-ninja)
  + [USING CLANG](https://gnss-sdr.org/build-and-install/#using-clang)
* [**OTHER BUILDS**](https://gnss-sdr.org/build-and-install/#other-builds)
  + [SNAP PACKAGE](https://gnss-sdr.org/build-and-install/#snap-package)
  + [DOCKER](https://gnss-sdr.org/build-and-install/#docker)
  + [GNSS-SDR IN EMBEDDED PLATFORMS](https://gnss-sdr.org/build-and-install/#gnss-sdr-in-embedded-platforms)

This page describes several ways to build and install GNSS-SDR.

**Installing everything through software package managers**[**Permalink**](https://gnss-sdr.org/build-and-install/#installing-everything-through-software-package-managers)

**GNU/Linux distributions based on Debian**[**Permalink**](https://gnss-sdr.org/build-and-install/#gnulinux-distributions-based-on-debian)

Starting from Debian 9 and Ubuntu 16.04, you can install GNSS-SDR just by doing:

$ sudo apt-get install gnss-sdr

However, even on very recent distributions, the latest packaged GNSS-SDR version is often at least months behind the latest developments, and the same applies to its dependencies. In order to get access to the most recent features and bug fixes, you might want to build the software from the source code.

**The good part:** This is the easiest and fastest way to get GNSS-SDR installed in your system. Just a single line in your terminal will do all the work for you.

**The downside:** GNSS-SDR package updates need to undergo an acceptance process before they are included in the different distributions, so it might not be the ultimate version of the source code. The same applies for the dependency libraries.

If everything went fine, you can directly jump into how to get your [first position fix](https://gnss-sdr.org/my-first-fix/). If your Operating System release still does not include the GNSS-SDR package, you can [build it from source](https://gnss-sdr.org/build-and-install/#source).

**macOS / Mac OS X using Macports**[**Permalink**](https://gnss-sdr.org/build-and-install/#macos--mac-os-x-using-macports)

If you are using macOS 10.15 Catalina (or Mac OS X 10.9 and above), and the [Macports](https://www.macports.org/) package manager, GNSS-SDR can be installed by typing in a Terminal:

$ sudo port install gnss-sdr

This will install the latest stable release of GNSS-SDR.

**You have more options here!**

Instead of installing the latest stable release, you can install the code found on the master branch, which might contain some bug fixes with respect to the latest stable release:

$ sudo port install gnss-sdr-devel

or the context of the next branch, which might contain fixes and new features with respect to the latest stable release:

$ sudo port install gnss-sdr-next

If everything went fine, you can directly jump into how to get your [first position fix](https://gnss-sdr.org/my-first-fix/). Or maybe you might prefer other options, such as using [Homebrew](https://brew.sh/) instead of Macports as a software package manager, or to build the source code manually. Fore more details of GNSS-SDR on macOS, please check the [README.md](https://github.com/gnss-sdr/gnss-sdr/blob/master/README.md#macos-and-mac-os-x) file.

**Building from source**[**Permalink**](https://gnss-sdr.org/build-and-install/#building-from-source)

**Is it better to compile from source or to install from a package?** Here we provide some guidelines. The rule of thumb is: Always consider installing from standard packages of your Operating System first; only compile from sources if you know exactly why you need to do that.

GNSS-SDR and its software dependencies can all be installed either by downloading their source code, building and installing them in your system, or by installing the corresponding software package. Under GNU/Linux, the package manager may vary according to the distribution you are using (.deb for Debian-based distros, such as Ubuntu, Linaro and Raspbian; .rpm for Fedora / SUSE / Mandriva; etc.). In Mac OS X, most popular open source package managers are Macports and Homebrew. In all cases, a possible option for a quick jump into GNSS-SDR is to install all the dependencies from packages, and then building GNSS-SDR from the source code.

But maybe this approach does not fit your needs. Maybe you already have some dependency already built from source and want to use it, or your setup requires some specific flag somewhere. This is a building-time *vs*. fine-tuning trade-off. In order to take the adequate approach, just remember this basic rule:

*Software packages require that all its dependencies must be also installed from packages.*

and take into account the following considerations:

**The good part:** The advantage of compiling from source is that you can compile packages with certain flags/options which may be missing or disabled in standard packages. Also, it makes it easy to have multiple versions of the same program installed. Also, you can choose an exact version of a package which may be already removed from, or not yet present in, your OS repositories.

**The downside:** The disadvantage of compiling from source is that the usual cmake .. && make && sudo make install procedure keeps your package manager completely unaware of the changes you are making, so you are not going to get any automatic updates for the manually-compiled software; and it is possible that package manager will later override/break your package if you are not careful to install it in a separate location.

Some highly automated tools that can do some of the work for you are described below.

**Install dependencies**[**Permalink**](https://gnss-sdr.org/build-and-install/#install-dependencies)

**Install all dependencies using packages**[**Permalink**](https://gnss-sdr.org/build-and-install/#install-all-dependencies-using-packages)

If you are using Debian 8, Ubuntu 14.10 or above, this can be done by copying and pasting the following line in a terminal:

$ sudo apt-get install build-essential cmake git libboost-dev \

libboost-date-time-dev libboost-system-dev libboost-filesystem-dev \

libboost-thread-dev libboost-chrono-dev libboost-serialization-dev \

libboost-program-options-dev libboost-test-dev liblog4cpp5-dev \

libuhd-dev gnuradio-dev gr-osmosdr libblas-dev liblapack-dev \

libarmadillo-dev libgflags-dev libgoogle-glog-dev libhdf5-dev \

libgnutls-openssl-dev libmatio-dev python-mako python-six \

libpugixml-dev libpcap-dev libprotobuf-dev protobuf-compiler \

libgtest-dev googletest

**Note for Ubuntu 14.04 LTS users:** you will need to build from source and install GNU Radio manually, as explained below, since GNSS-SDR requires gnuradio-dev >= 3.7.3, and Ubuntu 14.04 came with 3.7.2. Install all the packages above BUT EXCEPT libuhd-dev, gnuradio-dev and gr-osmosdr (and remove them if they are already installed in your machine), and install those dependencies using PyBOMBS, as explained below.

Once you have installed these packages, you can jump directly to [clone, build and install GNSS-SDR](https://gnss-sdr.org/build-and-install/#build).

**Install dependencies using PyBOMBS**[**Permalink**](https://gnss-sdr.org/build-and-install/#install-dependencies-using-pybombs)

This option is adequate if you are interested in development, in working with the most recent versions of software dependencies, want more fine-tuning on the installed versions, or simply in building everything from the scratch just for the fun of it. In such cases, we recommend to use [PyBOMBS](https://www.gnuradio.org/blog/pybombs-the-what-the-how-and-the-why) (Python Build Overlay Managed Bundle System), GNU Radio’s meta-package manager tool that installs software from source, or whatever the local package manager is, that automatically does all the work for you. Please take a look at the [configuration options and general PyBOMBS usage](https://github.com/gnuradio/pybombs). Here we provide a quick step-by-step tutorial.

First of all, install some basic packages:

$ sudo apt-get install git python3-pip

Download, build and install PyBOMBS:

$ sudo pip3 install --upgrade git+https://github.com/gnuradio/pybombs.git

Apply a configuration:

$ pybombs auto-config

Add list of default recipes (*i.e.*, instructions on how to install software dependencies):

$ pybombs recipes add-defaults

Download, build and install GNU Radio, related drivers and some other extra modules into the directory /path/to/prefix (replace this path by your preferred one, for instance $HOME/sdr):

$ pybombs prefix init /path/to/prefix -a myprefix -R gnuradio-default

This will perform a local installation of the dependencies under /path/to/prefix, so they will not be visible when opening a new terminal. In order to make them available, you will need to set up the adequate environment variables by sourcing the setup\_env.sh script:

$ cd /path/to/prefix

$ . ./setup\_env.sh

Now you are ready to use GNU Radio and to jump into [building GNSS-SDR](https://gnss-sdr.org/build-and-install/#build) after installing a few other dependencies. Actually, those are steps that PyBOMBS can do for you as well:

$ pybombs install gnss-sdr

By default, PyBOMBS installs the ‘next’ branch of GNSS-SDR development, which is the most recent version of the source code. This behaviour can be modified by altering the corresponding recipe at $HOME/.pybombs/recipes/gr-recipes/gnss-sdr.lwr

In case you do not want to use PyBOMBS and prefer to build and install GNSS-SDR step by step (i.e., cloning the repository and doing the usual cmake .. && make && sudo make install dance, as explained below), there are still some missing dependencies (*i.e.*, Armadillo, GFlags, Glog, GnuTLS and Protocol Buffers) that can be installed either by using PyBOMBS:

$ pybombs install armadillo gflags glog gnutls protobuf

or manually, just downloading, building and installing them. More details are available in the [README.md](https://github.com/gnss-sdr/gnss-sdr/blob/master/README.md#manual-installation-of-other-required-dependencies) file.

**Clone, build and install GNSS-SDR**[**Permalink**](https://gnss-sdr.org/build-and-install/#clone-build-and-install-gnss-sdr)

Once all the dependencies are installed in your system, you are ready to clone the repository, build the source code and install the software in your system:

$ git clone https://github.com/gnss-sdr/gnss-sdr

$ cd gnss-sdr/build

$ git checkout next

$ cmake ..

$ make

$ sudo make install

$ git clone https://github.com/gnss-sdr/gnss-sdr

$ cd gnss-sdr/build

$ git checkout next

$ cmake ..

$ make

$ sudo make install

* df command – Shows the amount of disk space used and available on Linux file systems.
* du command – Display the amount of disk space used by the specified files and for each subdirectory.
* btrfs fi df /device/ – Show disk space usage information for a btrfs based mount point/file system. Read more

# My first position fix

#### ON THIS PAGE

* [**STEP 1: VERIFY THAT GNSS-SDR IS INSTALLED**](https://gnss-sdr.org/my-first-fix/#step-1-verify-that-gnss-sdr-is-installed)
* [**STEP 2: DOWNLOAD A FILE OF RAW SIGNAL SAMPLES**](https://gnss-sdr.org/my-first-fix/#step-2-download-a-file-of-raw-signal-samples)
* [**STEP 3: CONFIGURE GNSS-SDR**](https://gnss-sdr.org/my-first-fix/#step-3-configure-gnss-sdr)
* [**STEP 4: RUN GNSS-SDR**](https://gnss-sdr.org/my-first-fix/#step-4-run-gnss-sdr)

**This page is the “Hello, world! ” for GNSS-SDR**. It will guide you from the scratch up to getting position fixes with GNSS-SDR, in one of its simplest configurations. The signal source will be a file (freely available on the Internet) containing raw signal samples, so this procedure does not require the availability of a radio frequency front-end nor a powerful computer executing the software receiver. The only requirement is GNSS-SDR installed in your computer, and an Internet connection to download the file containing the raw signal samples.

## Step 1: Verify that GNSS-SDR is installed[Permalink](https://gnss-sdr.org/my-first-fix/#step-1-verify-that-gnss-sdr-is-installed)

This guide assumes that GNSS-SDR and its software dependencies are already installed on your system. In order to check whether it is correctly installed, open a terminal and type:

$ gnss-sdr --version

you should see something similar to:

$ gnss-sdr --version

gnss-sdr version 0.0.11

$

Please check that your installed version is 0.0.11 (or something like 0.0.11.git-branchname-githash if you built the code from a source code snapshot). Older versions could not work for the example shown here. If you installed GNSS-SDR by doing sudo apt-get install gnss-sdr and you got a version earlier to 0.0.11, please do sudo apt-get remove gnss-sdr and [build it from source](https://gnss-sdr.org/build-and-install/#build).

If you see something like:

$ gnss-sdr --version

gnss-sdr: command not found

$

please check out the [building guide](https://gnss-sdr.org/build-and-install/) and the [README.md](https://github.com/gnss-sdr/gnss-sdr/blob/master/README.md) file for more details on how to install GNSS-SDR.

In order to take advantage of the SIMD instruction sets present in your processor, you will need to run the profiler tools of the VOLK and VOLK\_GNSSSDR libraries (these operations only need to be done once, and can take a while):

$ volk\_profile

and

$ volk\_gnsssdr\_profile

########################################################

Needed to change sysctl.conf

/etc/sysctl.conf

Added

##################################################################

# KRL: Increasing the size of the network read and write buffers

#################################################################

net.core.rmem\_max=24266666

net.core.wmem\_max=24266666

klieberschnitzel@ubuntu18-04-3:/usr/local/lib/uhd/examples$ ./benchmark\_rate --rx\_rate 1e3 --tx\_rate 1e3

[INFO] [UHD] linux; GNU C++ version 7.4.0; Boost\_106501; UHD\_3.14.1.HEAD-0-g0347a6d8

[WARNING] [UHD] Unable to set the thread priority. Performance may be negatively affected.

Set MTU=1500 to start

Edited /etc/security/limits.conf

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

klieberschnitzel@ubuntu18-04-3:/usr/local/lib/uhd/examples

location of benchmark\_rate

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

klieberschnitzel@ubuntu18-04-3:/usr/local/lib/uhd/examples$ sysctl net.ipv4.udp\_wmem\_min=8192

sysctl: permission denied on key 'net.ipv4.udp\_wmem\_min'

klieberschnitzel@ubuntu18-04-3:/usr/local/lib/uhd/examples$ sudo sysctl net.ipv4.udp\_wmem\_min=8192

[sudo] password for klieberschnitzel:

net.ipv4.udp\_wmem\_min = 8192

klieberschnitzel@ubuntu18-04-3:/usr/local/lib/uhd/examples$ sudo sysctl net.ipv4.udp\_rmem\_min=8192

net.ipv4.udp\_rmem\_min = 8192

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

uhd\_usrp\_probe

located at /usr/

./benchmark\_rate --tx\_rate 16.666667e6 --rx\_rate 16.666667e6 --args addr=192.168.10.2 --duration 120

The Trump inquiry, at a minimum, is presenting to the people of the United States by ranking member Mr. Nunes, that the Republican Party supports a President’s autonomy in divulging and gathering information from people who are outside the US government and do not necessarily have a secret clearance. I work for the military and have a secret clearance and I am trained yearly to never speak of what I do to anyone about my work even if they have a secret clearance but are restricted because of “need-to-know” laws. I can be jailed if I break this rule.

$ sudo sysctl -w net.core.rmem\_max=26214400  
net.core.rmem\_max = 26214400  
$ sudo sysctl -w net.core.rmem\_default=26214400  
net.core.rmem\_default = 26214400

Run sysctl.conf

Sudo sysctl -p

Run gnss-sdr

$ gnss-sdr --config\_file=./my\_GPS\_receiver.conf

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

MTU testing

### Check the configured MTU

You can verify that an interface has the intended MTU using the -i option of the netstat command:

netstat -i

The MTU of each interface is listed in the second column of the output:

Iface MTU Met RX-OK RX-ERR RX-DRP RX-OVR TX-OK TX-ERR TX-DRP TX-OVR Flg

eth0 1500 0 9658 0 0 0 308 0 0 0 BMRU

lo 16436 0 12952 0 0 0 12952 0 0 0 LRU

Alternatively you can obtain essentially the same information using the ifconfig command:

ifconfig eth0

The MTU is then listed after the interface address (or addresses) but before the packet counters:

eth0 Link encap:Ethernet HWaddr 02:00:00:00:00:00

inet addr:192.168.0.2 Bcast:192.168.0.255 Mask:255.255.255.0

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:9659 errors:0 dropped:0 overruns:0 frame:0

TX packets:309 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:1000

RX bytes:2783810 (2.6 MiB) TX bytes:25318 (24.7 KiB)

Remember that the value listed is not necessarily the one that would apply following a reboot.

### Check connectivity using ping

You can verify that the path between two machines has at least the expected MTU using the ping command:

ping -M do -c 4 -s 8972 192.168.0.1

This transmits an ICMP echo request to the specified destination then waits for an ICMP echo reply. As the intention here is to check the behaviour of eth0, it would be best to choose a destination on the same subnet in the first instance. Both machines must have a sufficiently large MTU if the test is to succeed.

The -M do option causes the DF (don't fragment) flag to be set, meaning that the packet should be dropped if it cannot remain in one piece at any point in its journey. The -c option sets the number of pings.

The -s option specifies the number of bytes of padding that should be added to the echo request. In addition to this number there will be 20 bytes for the internet protocol header, and 8 bytes for the ICMP header and timestamp. The amount of padding should therefore be 28 bytes less than the network-layer MTU that you are trying to test (9000 − 28 = 8972).

If the test is successful then you should see a list of echo replies that were received:

PING 192.168.0.1 (192.168.0.1) 8972(9000) bytes of data.

1480 bytes from 192.168.0.1: icmp\_seq=1 ttl=64 time=0.241 ms

1480 bytes from 192.168.0.1: icmp\_seq=2 ttl=64 time=0.583 ms

1480 bytes from 192.168.0.1: icmp\_seq=3 ttl=64 time=0.140 ms

1480 bytes from 192.168.0.1: icmp\_seq=4 ttl=64 time=0.123 ms

Note that the IP packet size (9000 bytes) is listed on the first line. You can use this to check that you requested the correct amount of padding.

If the test is unsuccessful then you should see an error in response to each echo request:

PING 192.168.0.1 (192.168.0.1) 8972(9000) bytes of data.

From 192.168.0.2 icmp\_seq=1 Frag needed and DF set (mtu = 1500)

From 192.168.0.2 icmp\_seq=1 Frag needed and DF set (mtu = 1500)

From 192.168.0.2 icmp\_seq=1 Frag needed and DF set (mtu = 1500)

From 192.168.0.2 icmp\_seq=1 Frag needed and DF set (mtu = 1500)

Note that the MTU for the hop that failed is listed in the error message. This is not necessarily the MTU for the entire path, only the first MTU that was found to be too small.

Path MTUs are recorded in the routing cache. This can interfere with testing, and in particular, can make a remote MTU restriction appear to be a local one. You can clear the cache using the ip route command:

ip route flush cache

%%%%%%%%%%%%%%%%%%%%%%%%%%%

GNSS-SDR

# Configurations

#### ON THIS PAGE

* [**GPS L1 C/A RECEIVER USING A USRP**](https://gnss-sdr.org/conf/#gps-l1-ca-receiver-using-a-usrp)
  + [REQUIRED EQUIPMENT](https://gnss-sdr.org/conf/#required-equipment)
  + [SETTING UP THE FRONT-END](https://gnss-sdr.org/conf/#setting-up-the-front-end)
  + [SETTING UP THE SOFTWARE RECEIVER](https://gnss-sdr.org/conf/#setting-up-the-software-receiver)
  + [RUN IT!](https://gnss-sdr.org/conf/#run-it)
  + [IF SOMETHING GOES WRONG](https://gnss-sdr.org/conf/#if-something-goes-wrong)
* [**GPS L1 C/A RECEIVER USING A HACKRF ONE**](https://gnss-sdr.org/conf/#gps-l1-ca-receiver-using-a-hackrf-one)
  + [REQUIRED EQUIPMENT](https://gnss-sdr.org/conf/#required-equipment-1)
  + [SETTING UP THE FRONT-END](https://gnss-sdr.org/conf/#setting-up-the-front-end-1)
  + [SETTING UP THE SOFTWARE RECEIVER](https://gnss-sdr.org/conf/#setting-up-the-software-receiver-1)

Obtaining position fixes from a file is nice and useful, but the real deal for a software-defined receiver is to play with live GNSS signals in real-time. This page describes examples of hardware setups, software configurations and general tips for obtaining position fixes (and a collection of side data, delivered in standard formats) with GNSS-SDR.

**Important note:** Either if you are capturing signal in a file for post-processing, or if you are configuring a receiver to work in real-time with live signal samples delivered by a radio frequency front-end, **you will need an active antenna** (that is, an antenna with an integrated low noise amplifier delivering a gain in the band of interest greater than 2020 dB, with a noise figure below 22 dB), **and you will need to feed it**.

Some radio frequency front-ends have jumpers, or some other configurable mode, for antenna feeding. If this feature is not available, you will need a bias-T in the antenna cable or some other kind of protection in order to inject power to the antenna without harming the front-end, plus an adequate power source (GNSS antennas use to be fed in a range from 2.5 VDC to 5.5 VDC, please check your model’s datasheet).

## GPS L1 C/A receiver using a USRP[Permalink](https://gnss-sdr.org/conf/#gps-l1-ca-receiver-using-a-usrp)

This in an example of an eight-channel GPS L1 C/A receiver, working at 4 MS/s (baseband, i.e. complex samples), and using a device from the [USRP family](https://www.ettus.com/product) as the “air-to-computer” interface.

### Required equipment[Permalink](https://gnss-sdr.org/conf/#required-equipment)

In order to get real-time position fixes, you will need:

* **An active GPS antenna**. Any model will fit, just check that you can plug it to the USRP’s SMA-male connector.
* **A USRP**. All models will do the job. Depending on the specific USRP model you are using, the configuration may vary (see below).

| **USRP Model** | **RF bandwidth (MHz @ 16 bits per sample)** | **RX gain (dB)** | **ADC Processing Bandwidth (MS/s)** | **Interface** | **Host Sample Rate (MS/s @ 16-bit I&Q)** |
| --- | --- | --- | --- | --- | --- |
| USRP 1, B100 | Up to 88 MHz | See daughterboard | 6464 MS/s | USB 2.0 | 88 MS/s |
| B200, B210, B200mini, B205mini | From 200200 kHz to 5656 MHz | Up to 7373 dB | 61.4461.44 MS/s | USB 3.0 | 61.4461.44 MS/s |
| N200, N210 | Up to 2525 MHz | See daughterboard | 100100 MS/s | Gigabit Ethernet | 2525 MS/s |
| X300, X310 | Up to 120120 MHz | See daughterboard | 200200 MS/s | 10 Gigabit Ethernet | 200200 MS/s |

Some USRP models and features. The ADC processing bandwidth is the sample rate provided by the ADCs on the USRP motherboard, and the host sample rate refers to the sample stream between the FPGA of a USRP device, and a host PC. Some USRP models also provide the option to stream 8-bit samples, effectively doubling the host-bandwidth in terms of samples/second. Source: [*Ettus Research Knowledge Base*](https://kb.ettus.com/About_USRP_Bandwidths_and_Sampling_Rates).

* The USRP family features a modular architecture with interchangeable daughterboard modules that serve as the RF front end. In case of using a USRP without an embedded transceiver, you will need **a daughterboard** allowing the reception of signals around 1.5 GHz. That is: DBSRX2, WBX, SBX, CBX and UBX daughterboards can work for you. You will not need a daughterboard if you are using USRP B200, B210 or E310, which ship an Analog Devices AD9361 RFIC as an integrated wideband transceiver.

| **Daughterboard** | **Frequency coverage** | **Analog bandwidth** | **RX gain** |
| --- | --- | --- | --- |
| WBX-120 | 50 MHz - 2.2 GHz | 120 MHz | 00 to 31.531.5 dB |
| SBX-120 | 400 MHz - 4.4 GHz | 120 MHz | 00 to 31.531.5 dB |
| CBX-120 | 1.2 GHz - 6 GHz | 120 MHz | 00 to 31.531.5 dB |
| UBX-160 | 10 MHz - 2.2 GHz | 160 MHz | 00 to 31.531.5 dB |
| WBX | 50 MHz - 2.2 GHz | 40 MHz | 00 to 31.531.5 dB |
| SBX | 400 MHz - 4.4 GHz | 40 MHz | 00 to 31.531.5 dB |
| CBX | 1.2 GHz - 6 GHz | 40 MHz | 00 to 31.531.5 dB |
| UBX-40 | 10 MHz - 6 GHz | 40 MHz | 00 to 31.531.5 dB |
| DBSRX2 | 800 MHz - 2.3 GHz | Configurable: 8 MHz to 80 MHz | GC1 from 00 to 7373 dB, and BBC from 00 to 1515 dB |

Daugtherboards allowing GNSS signal reception. Note that WBX-120, CBX-120 and SBX-120 daughterboards were designed to work with USRP X300/X310 and any future products with sufficient ADC/DAC sample rates. These boards are not compatible with devices that incorporate lower rate ADC/DAC below 200 MS/s. Source: [*Ettus Research Knowledge Base*](https://kb.ettus.com/About_USRP_Bandwidths_and_Sampling_Rates).

* **A computer** connected to the USRP and with GNSS-SDR installed.

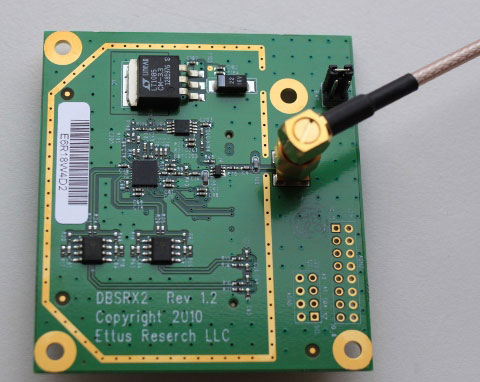
### Setting up the front-end[Permalink](https://gnss-sdr.org/conf/#setting-up-the-front-end)

The first thing to do is to install a suitable daughterboard into the USRP. As a example, you can check Ettus Research’s detailed [step-by-step guide to install a daughterboard into the USRP N200/N210](https://kb.ettus.com/USRP_N_Series_Quick_Start_(Daughterboard_Installation)). In USRPs with two receiving slots, please check in which one you are inserting the daughterboard (they are usually labeled as “RX A” and “RX B”). This is something that you will need to specify in the configuration file (via the subdevice parameter, see below).

Then, you will need to feed your GNSS active antenna.

**Important:** Never apply more than -15 dBm of power into any RF input.

In case of using a DBSRX2 daughterboard, you will need to adjust the J101 jumper in order to feed the antenna.

DBSRX2 daughterboard. The J101 jumper in the upper right corner allows the injection of current towards the antenna. Source: [*Radio Adventures*](http://yo3iiu.ro/blog/).

If this feature is not available (e.g., WBX daughterboards), you will need a bias-T between the USRP and the antenna, and to connect it to a power source delivering the voltage required by your antenna (usually, 3 V or 5 V).

Bias-T allowing the injection of DC voltage to the antenna. Source: [*Radio Adventures*](http://yo3iiu.ro/blog/).

USRP N210 with the bias-T and the GPS antenna. Source: [*Radio Adventures*](http://yo3iiu.ro/blog/).

Depending on the specific USRP model you are using, the connection to the host computer in charge of the execution of the software receiver can be through USB (2.0 or 3.0) or Ethernet (1 GigE or 10 GigE). Once connected, every USRP device has [several ways of identifying it](https://files.ettus.com/manual/page_identification.html) on the host system:

| **Identifier** | **Key** | **Notes** | **Example** |
| --- | --- | --- | --- |
| Serial # | serial | globally unique identifier | 12345678 |
| IP Address | addr | unique identifier on a network | 192.168.10.2 |
| Type | type | hardware series identifier | usrp1, usrp2, b200, x300, … |
| Name | name | optional user-set identifier | lab1\_usrp3 |

Common device identifiers. Source: [*Identifying USRP Devices*](https://files.ettus.com/manual/page_identification.html).

Devices attached to your system can be discovered using the uhd\_find\_devices program. This program scans your system for supported devices and prints out an enumerated list of discovered devices and their addresses. If you type uhd\_find\_devices --help in a terminal, you should see something similar to this:

$ uhd\_find\_devices --help

linux; GNU C++ version 4.9.2; Boost\_105400; UHD\_003.010.git-0-2d68f228

UHD Find Devices Allowed options:

--help help message

--args arg device address args

Then, you can search your USRP in a specific IP address:

$ uhd\_find\_devices --args addr=192.168.50.2

linux; GNU C++ version 4.9.2; Boost\_105400; UHD\_003.010.git-0-2d68f228

--------------------------------------------------

-- UHD Device 0

--------------------------------------------------

Device Address:

type: x300

addr: 192.168.50.2

fpga: HGS

name:

serial: F5CA38

product: X300

or by typing:

$ uhd\_find\_devices --args type=usrp1

This is a good way to check if the USRP is correctly connected to your computer. After this check, we can proceed to configure the software receiver. The [USRP Hardware Driver and USRP Manual](https://files.ettus.com/manual/page_devices.html) provides more information about the configuration and usage of those devices.

### Setting up the software receiver[Permalink](https://gnss-sdr.org/conf/#setting-up-the-software-receiver)

Copy the configuration below into your favorite plain text editor and save it with a name such as my\_GPS\_receiver.conf in your working directory.

[GNSS-SDR]

;######### GLOBAL OPTIONS ##################

GNSS-SDR.internal\_fs\_sps=4000000

;######### SIGNAL\_SOURCE CONFIG ############

SignalSource.implementation=UHD\_Signal\_Source

SignalSource.device\_address=192.168.50.2 ; <- PUT THE IP ADDRESS OF YOUR USRP HERE

; OR LEAVE IT EMPTY FOR USB

SignalSource.item\_type=cshort

SignalSource.sampling\_frequency=4000000

SignalSource.freq=1575420000

SignalSource.gain=40

SignalSource.subdevice=A:0 ; <- Can be A:0 or B:0

SignalSource.samples=0

;######### SIGNAL\_CONDITIONER CONFIG ############

SignalConditioner.implementation=Signal\_Conditioner

;######### DATA\_TYPE\_ADAPTER CONFIG ############

DataTypeAdapter.implementation=Pass\_Through

DataTypeAdapter.item\_type=cshort

;######### INPUT\_FILTER CONFIG ############

InputFilter.implementation=Fir\_Filter

InputFilter.input\_item\_type=cshort

InputFilter.output\_item\_type=gr\_complex

InputFilter.taps\_item\_type=float

InputFilter.number\_of\_taps=11

InputFilter.number\_of\_bands=2

InputFilter.band1\_begin=0.0

InputFilter.band1\_end=0.48

InputFilter.band2\_begin=0.52

InputFilter.band2\_end=1.0

InputFilter.ampl1\_begin=1.0

InputFilter.ampl1\_end=1.0

InputFilter.ampl2\_begin=0.0

InputFilter.ampl2\_end=0.0

InputFilter.band1\_error=1.0

InputFilter.band2\_error=1.0

InputFilter.filter\_type=bandpass

InputFilter.grid\_density=16

InputFilter.sampling\_frequency=4000000

InputFilter.IF=0

;######### RESAMPLER CONFIG ############

Resampler.implementation=Pass\_Through

;######### CHANNELS GLOBAL CONFIG ############

Channels\_1C.count=8

Channels.in\_acquisition=1

Channel.signal=1C

;######### ACQUISITION GLOBAL CONFIG ############

Acquisition\_1C.implementation=GPS\_L1\_CA\_PCPS\_Acquisition

Acquisition\_1C.threshold=0.01

Acquisition\_1C.doppler\_max=8000

Acquisition\_1C.doppler\_step=500

;######### TRACKING GLOBAL CONFIG ############

Tracking\_1C.implementation=GPS\_L1\_CA\_DLL\_PLL\_Tracking

Tracking\_1C.pll\_bw\_hz=30.0

Tracking\_1C.dll\_bw\_hz=4.0

Tracking\_1C.early\_late\_space\_chips=0.5

;######### TELEMETRY DECODER GPS CONFIG ############

TelemetryDecoder\_1C.implementation=GPS\_L1\_CA\_Telemetry\_Decoder

;######### OBSERVABLES CONFIG ############

Observables.implementation=Hybrid\_Observables

;######### PVT CONFIG ############

PVT.implementation=RTKLIB\_PVT

PVT.positioning\_mode=Single

PVT.output\_rate\_ms=10

PVT.display\_rate\_ms=500

PVT.iono\_model=Broadcast

PVT.trop\_model=Saastamoinen

PVT.flag\_rtcm\_server=true

PVT.flag\_rtcm\_tty\_port=false

PVT.rtcm\_dump\_devname=/dev/pts/1

PVT.rtcm\_tcp\_port=2101

PVT.rtcm\_MT1019\_rate\_ms=5000

PVT.rtcm\_MT1077\_rate\_ms=1000

PVT.rinex\_version=2

You will need to adjust the values for at least two parameters:

* Check that SignalSource.device\_address points to the actual IP address of your USRP, if you are connected through Ethernet, or leave it empty for USB.
* Check that SignalSource.subdevice is set to the receiving slot in which you actually inserted your daughterboard with the antenna. In USRPs with only one receiving slot, leave it as A:0. Please check more details on [how to specify the subdevice](https://files.ettus.com/manual/page_configuration.html#config_subdev).

The [Signal Processing Blocks documentation](https://gnss-sdr.org/docs/sp-blocks/) provides definitions and more details about the configuration parameters.

### Run it![Permalink](https://gnss-sdr.org/conf/#run-it)

Once the hardware and the software configurations are ready, go to your favorite working directory where the file my\_GPS\_receiver.conf was stored and invoke the software receiver with this particular configuration:

$ gnss-sdr --config\_file=./my\_GPS\_receiver.conf

You should see something similar to:

$ gnss-sdr --config\_file=./my\_GPS\_receiver.conf

linux; GNU C++ version 4.9.2; Boost\_105400; UHD\_003.010.git-0-2d68f228

Initializing GNSS-SDR v0.0.11 ... Please wait.

Logging will be done at "/tmp"

Use gnss-sdr --log\_dir=/path/to/log to change that.

-- X300 initialization sequence...

-- Determining maximum frame size... 8000 bytes.

-- Setup basic communication...

-- Loading values from EEPROM...

-- Setup RF frontend clocking...

-- Radio 1x clock:200

-- Initialize Radio0 control...

-- Performing register loopback test... pass

-- Initialize Radio1 control...

-- Performing register loopback test... pass

Sampling Rate for the USRP device: 4000000.000000 [sps]...

UHD RF CHANNEL #0 SETTINGS

Actual USRP center freq.: 1575420000.010133 [Hz]...

PLL Frequency tune error 0.010133 [Hz]...

Actual daughterboard gain set to: 37.500000 dB...

Setting RF bandpass filter bandwidth to: 2000000.000000 [Hz]...

Check for front-end LO: locked ... is Locked

Using Volk machine: avx2\_64\_mmx

Starting a TCP Server on port 2101

The TCP Server is up and running. Accepting connections ...

...

Of course, file my\_GPS\_receiver.conf can be wherever (--config-file accepts both relative and absolute paths), and the data displayed at the terminal output might vary according to your setup.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Fixed Issue by setting Windows 10 side to Jumbo Frame and set MTU to 4000 (max my usb3 ethent adabper could handle) and tried MTU 4000 on Ubuntu side but I was getting

WARN: USRP Source Block caught rx error code: 15

[ERROR] [RX FLOW CTRL] Error unpacking packet: ValueError: Bad CHDR or packet fragment

Then I set MTU on Ubuntu side to 1500 and it works.

