Gpsd ou Kplex

Entre les 2 mon cœur balance ; lequel choisir ?

<http://catb.org/gpsd/> ou <http://www.stripydog.com/kplex/index.html>

Kplex, car il me semble que Gpsd n’accepte qu’une seule entrée ?

# Exemple d’usage Kplex / OpenCPN / Raspberry

## A Raspberry Pi Wireless NMEA-0183 Multiplexer

(USB GPS, Digital Compass, and Sailboat Instrument Input)

Dans le lien <https://kingtidesailing.blogspot.com/2016/08/example-nmea-0183-sentences.html> , je trouve :   
The "PFEC" sentence is a proprietary sentence used so that my Furuno RD30 display can read roll and pitch. I use the free NKE Marine Electronics app on my phone to display everything, so I include a few XDR sentences so that it recognizes them (same with GPVTG).

### Step 6: Setup a Wireless Kplex Multiplexer

Kplex is a fantastic tool, but to be totally honest, the documentation kind of sucks. It isn't written for computer morons like me, so I had to do a lot of testing and searching to figure it out. First, install it by doing this (in the kts directory of course):

git clone https://github.com/stripydog/kplex.git

cd kplex

make

sudo make install

Here is my kplex.conf file that accepts UDP inputs from the previous python scripts, and prints it all to a TCP port that my laptop/phone can connect to (as well as to the display I got on eBay and an additional port that can print to my Autopilot, or receive from another instrument). This kplex.conf file is also included in the github download from my page.

#### kplex.conf

# From GPS

[udp]

address=**127.0**.**0.1**

port=**5005**

direction=**in**

# From IMU

[udp]

address=**127.0**.**0.2**

port=**5005**

direction=**in**

# From DST

[udp]

address=**127.0**.**0.3**

port=**5005**

direction=**in**

# From BME

[udp]

address=**127.0**.**0.7**

port=**5005**

direction=**in**

# To TCP

[tcp]

mode=server

port=**10110**

# To/From RD2030

[serial]

filename=/dev/rd

direction=both

baud=**4800**

# To/From Autopilot

[serial]

filename=/dev/ap

direction=both

baud=**4800**

Simply save this as kplex.conf in the kts/scripts folder (where a copy already is if you downloaded my github) and copy it into the /etc/ directory with the following command in the terminal:

sudo cp kplex.conf /etc/

But we're not done. What address should you connect to on your phone or laptop? Find out with the following command:

ifconfig

and write down the number next to inet address of wla0 (mine is 10.0.0.1). When you add a connection in OpenCPN or the NKE Marine Electronics iOS app, use that number for the address, and use the port number that is in your kplex.conf file (if you used mine, the port number is 10110). Now, when you connect, you'll have a steady stream of NMEA data!  
  
But let's make sure everything is working before we proceed. We're going the run the imu.py script, which will output NMEA data to the kplex port, and then run kplex as well with the raw text output:

python imu.py &

sudo kplex file:direction=out

That will output a text string of whatever kplex is receiving. Great. So now our instruments are outputting data, kplex is receiving, if you connect to the WiFi network and type in the inet address on your device (OpenCPN/NKE, with the port number specified in kplex.conf) you can receive over WiFi. So now, we just need to run the monitor script on its own so all this will happen automatically.

## Step 3: Configure WiFi Network and Ethernet Bridge (optional)

<http://andersonsabroad.com/blog/raspberry-pi-marine-computer/step-3-configure-wifi-access-point-and-internet-gateway/>

Setting up a wireless access point on your Raspberry Pi is only necessary if you have a wired network but don’t have a wireless network.  This allows devices like iPad’s and laptops to connect wirelessly receive NMEA data for navigational apps.  It also gives these devices access to the rest of your network.

Even if you have a wireless network, you might also want to do this if you want to have a dedicated wireless network for navigational purposes only.

Okay, that said, let’s get to it:

First let’s make sure we have the most up-to-date list of the available software packages..  
**sudo apt-get update**

Now let’s install the 3 packages we need for this task  
***sudo apt-get install hostapd bridge-utils iw***

Next we need to edit some configuration files to set things up correctly:

***sudo nano /etc/default/hostapd***Uncomment and edit the following line as shown..      **DAEMON\_CONF=”/etc/hostapd/hostapd.conf”**Save and close the file (***CTRL-X***)

***sudo nano /etc/hostapd/hostapd.conf***Edit the file as follows:  
**interface=wlan0**  
**bridge=br0**  
**driver=rtl871xdrv**  
**country\_code=US**  
**ssid=<TypeInYourWiFiNetworkNameHere>**  
**hw\_mode=g**  
**channel=6 #you can pick any channel from 1 to 11**  
**macaddr\_acl=0**  
**auth\_algs=1**  
**ignore\_broadcast\_ssid=0**  
**wpa=2**  
**wpa\_passphrase=<TypeInYourPassPhraseHere>**  
**wpa\_key\_mgmt=WPA-PSK**  
**wpa\_pairwise=TKIP**  
**rsn\_pairwise=CCMP**  
**ieee80211n=1**Save and close the file (***CTRL-X***)

Backup the network interfaces config file and edit it:  
***sudo cp /etc/network/interfaces /etc/network/interfaces.bak***  
**sudo nano /etc/network/interfaces**Edit the file as follows:  
**auto lo**  
**auto br0**  
**iface lo inet loopback**  
**iface br0 inet static**  
**address (IP address you want your RaspBerry Pi device to use)**  
**netmask 255.255.255.0**  
**bridge\_fd 1**  
**bridge\_hello 3**  
**bridge\_maxage 10**  
**bridge\_stp off**  
**bridge\_ports eth0 wlan0**

**auto eth0**  
**allow-hotplug eth0**  
**iface eth0 inet manual**

**auto wlan0**  
**allow-hotplug wlan0**  
**iface wlan0 inet manual**  
Save and close the file (***CTRL-X***)

Now we need to change to a different version of the hostapd package to support the EDIMAX WiFi USB device

Reference: <http://blog.sip2serve.com/post/48420162196/howto-setup-rtl8188cus-on-rpi-as-an-access-point>  ***Note: I used this as a starting point as well as the replacement file, but the instructions I am providing are slightly different from the referenced page.  This is because I had trouble getting it all to work and had to modify a few things.  The instructions I am posting are from what I actually did to make it work.***

Backup the current executable file  
***cd /usr/sbin***  
***sudo cp /usr/sbin/hostapd /usr/sbin/hostapd.bak***

Delete the original file  
**sudo rm –f hostapd**

Download the replacement file to support the EDIMAX WiFi  
**sudo wget** [***http://dl.dropbox.com/u/1663660/hostapd/hostapd***](http://dl.dropbox.com/u/1663660/hostapd/hostapd)

Now set permissions on the file so it will run  
***sudo chown root:root hostapd***  
***sudo chmod 755 hostapd***

Now that all the files are configured and ready, let’s restart the rPi again  
**reboot**

The network should appear on other devices now.

## Step 4: Install kPlex NMEA Multiplexer

<http://andersonsabroad.com/blog/raspberry-pi-marine-computer/step-4-install-kplex-nmea-multiplexer/>

kPlex is the software that we’ll use to receive and forward NMEA network data between all of the NMEA0183 (2 USB adapters like [this one](http://www.digitalyachtamerica.com/index.php/en/products/pc-systems/pc-accessories/product/67-usb-to-nmea-adaptor)) and TCP/IP (Ethernet and WiFi) interfaces.   It will also serve as a source of data for OpenCPN running directly on the Raspberry Pi.

kPlex Website: <http://www.stripydog.com/kplex/index.html>

First we need to install kPlex itself.  There is a package already built for Raspbian, but it’s not in the APT repository, so we will just need to download that package and install it manually.

Download the package for Raspbian Wheezy  
***cd ~***  
***sudo wget http://www.stripydog.com/download/kplex\_1.3.4-1\_armhf.deb***

Then we will install the package we just downloaded  
  ***sudo dpkg -i ./kplex\_1.3.4-1\_armhf.deb***

Next we can configure kPlex to automatically start when the rPi boots up  
***sudo update-rc.d kplex defaults***

Now we will set up a basic NMEA Server by setting the appropriate parameters in the kplex config file.  
***sudo nano /etc/kplex.conf***  
Edit the file as follows

**[serial]  
filename=/dev/ttyUSB0  
direction=in  
baud=4800  
name=mux**

**[serial]  
filename=/dev/ttyUSB1  
direction=in  
baud=38400  
name=ais**

**[tcp]  
mode=server  
port=10110  
direction=out**

**[udp]  
port=10110  
address=255.255.255.255  
type=broadcast  
coalesce=no**

**#[broadcast]  
#direction=out  
#device=br0**

**[global]  
failover=GP\*\*\*:0:mux:60:ais**

Save and Exit (***CTRL-X***)

Okay, let’s explain what this config file actually does.  You will likely need to adjust this based on your configuration

The first **[serial]** section defines an NMEA to USB interface that I’ve named “**mux**” because the data coming in that port is coming from a [Digital Yacht NMEA MUX100](http://www.digitalyachtamerica.com/index.php/en/products/interfacing/multiplexers/product/78-mux100) device.  It’s set to 4800 because that’s the speed that the MUX100 device’s output is configured for.  This port is configured to receive NMEA data but not send any.

The second **[serial]** section defines another NMEA to USB interface named “**ais**” because that connection is attached to the NMEA output port of the West Marine AIS-1000.  Since AIS NMEA ports run at 38400 baud that is defined in this configuration as well.   Again, this port receives data but does not send out.

In both of the above cases, I’m using a low cost USB to Serial Adapter (Trendnet TU-S9) which goes for about $10 on Amazon.com

Then we have a **[tcp]** section which defines the network server on port 10110 for devices like iPad’s and laptops to query kPlex for NMEA data.  This is configured to send NMEA data out but not receive it.

Following that there is the **[udp]** section which defines a broadcast server on port 10110 for devices that merely listen for NMEA data coming in rather than asking for it. This is also using port 10110 (if not specified, kPlex uses that port by default anyway) and it’s configured to send to the all-hosts broadcast address of 255.255.255.255.

The **[broadcast]** section is an older version of the [udp] section and I’ve included it in case you run in to any issues with UDP.  You can comment out the [udp] section and uncomment the [broadcast] section if you need to.

Lastly because both the MUX100 and the AIS will be sending GPS position data, I configured a **failover** rule in the **[global]** section that makes the mux100 the primary source and the AIS a failover source for just the GPS data.   As long as GPS position data is coming in on USB0, all of the GPS data on USB1 will be ignored.  If GPS data stops coming in on USB0, then kPlex will start accepting it from USB1. All other data coming in on both ports is unaffected by this rule.

There are also ways to configure filters so certain NMEA data is not accepted but I am not configuring that right now until I have a chance to see what data actually shows up when everything is connected.

Full documentation for the kplex.conf file is here: <http://www.stripydog.com/kplex/configuration.html>

I started with the quickstart guides and used the configuration guide above to enhance and adjust.  
<http://www.stripydog.com/kplex/quickstart.html>  
<http://www.stripydog.com/kplex/examples/nmeaserver.html>

Finally, assuming the file is configured correctly and all the devices and interfaces are connected and installed (ie: the USB devices need to be plugged in and seen by Raspbian) you can start the kplex daemon.

***sudo service kplex start***

# Kplex

What is Kplex?

kplex is a multitransport software data multiplexer which runs on GNU/Linux, Mac OS X, FreeBSD, NetBSD and OpenWRT. It works with data which conform to the NMEA-0183 standard, but has been written without reference to any proprietary documentation not available in the public domain. Kplex multiplexes data inputs from sources such as serial lines, pseudo terminals and network interfaces and send to any (reasonable) number of outputs. kplex can perform filtering of inputs (so you only get the data you want, or don't get the data you don't want from a given source) and outputs (so you only send what you want where you want) and can perform fine-grained failover so that for any given type of data, you specify a priority order of the source you would like to take it from. For further details see the kplex [overview](http://www.stripydog.com/kplex/kplex.html).

## Kplex Overview

kplex is a program for combining and routing NMEA-0183 data to and from multiple sources and destinations. kplex inputs and outputs may be any of

Serial lines

The traditional transport for NMEA-0183. Ideally for marine electronics an opto-isolated RS422 interface, although some devices produce outputs designed for PCs (either RS232 or USB). Many people exploit wide tollerances and accept the risk of damage to plug un-isolated NMEA-0183 outputs into their computers either directly or via a serial to usb converter.   
kplex makes no distinction between usb and rs232 serial lines. The operating system hides the distinction between the two.

Pseudo Terminals (ptys)

Pseudo Terminals (or "pty"s) are "virtual" serial ports. kplex can read or write data to/from an existing pty just like a "real" serial port. kplex can also create ptys. This presents an interface to, for example, a navigation program which only knows about serial input, allowing it to read data which has actually been received over an IP network. By creating multiple ptys, kplex can "own" serial inputs allowing it to route data where you want, but also allow multiple navigation programs to co-exist, each of which needs to "own" the interface on which it sends and receives data.

Network Interfaces

kplex can create a network server process allowing programs to connect to it over a TCP/IP network. IPv4 and IPv6 are supported. This option is especially useful in conjunction with a wireless access point, allowing devices with only wireless connectivity such as ipads running navigation programs to send and receive nmea data. kplex can also connect to a remote tcp server such as a physical nmea-to-wireless box or another instance of kplex, allowing you to, for example, receive nmea data wirelessly on your laptop.   
kplex can also send and receive nmea data over unicast, multicast or broadcast UDP. Broadcast is not a concept used by IPv6, so that is an IPv4-only feature.

Files

kplex can read or write data to or from a file. Using a regular file can be useful to log sentences received or to send test data. This option is most useful when the "file" is the user's terminal: sentences can be monitored in real time and manually input.

The exact number of inputs and outputs kplex can support depends on the way it is configured, but unless you need many tens of thousands of interfaces, you will be limited only by your system's capacity to process the data.

### Advanced Features

kplex does more than just copying all data input to all of its outputs.

#### Checksumming

kplex can be instructed to discard any packets whose nmea checksum is incorrect.This feature can be enabled or disabled globally or on an interface-by- interface basis.

#### Sentence Filtering

kplex can filter sentences on input, accepting or discarding those which match a particular sentence type or talker on a per-interface basis. kplex also allows you to select what data you do or do not pass to each output.

#### Sentence Source Failover

Sometimes you have more than one source for the same type of data. You want to use one source if it is available, but fail to a different source if the primary stops working. kplex allows you to specify, independently for any talker ID or sentence type, a priority list of input sources. For each sentence type sepcified in a failover list, when that data type has not been observed on the primary source interface for a specified period of time, kplex will allow that data type to be received from the next interface in the priority list. Multiple ordered failover sources may be specified.

#### Rate Limiting

kplex can limit the rate at which sentences matching a particular pattern enter or leave an interface. This can be useful in data logging where you wish to record your position once per hour (for example).

## Kplex Quickstart

### Purpose

Kplex has a lot of options. The main [configuration page](http://www.stripydog.com/kplex/configuration.html) is now long and off-putting for new users. Getting kplex to do what most people want to do is relatively easy. This document attempts to simplify configuration basics. What follows is the easiest way to get kplex working, although not necessarily the best.

### Installation

This is covered on the [installation page](http://www.stripydog.com/kplex/kplex.html). If you ultimately want kplex to run automatically on boot, don't set that up just yet: we want to configure kplex first..

### Configuration

kplex's behaviour can be defined either in a configuration file or on the invocation command line. Any directives given on the command line will be added to (if compatible) or override (if not) directives given in the configuration file. The default location for the configuration file is /etc/kplex.conf. A kplex.conf is installed with the .deb file, but this file does nothing: it is the equivalent of a blank file: kplex ignores everything on a line after a "#" character. You can replace this file, remove all the lines and add new ones, or simply write your configuration at the end.

The configuration file is used to define kplex's inputs and outputs. Inputs and outputs are referred to as "interfaces". An interface can be an input, an output, or both.

kplex has many global options which can be defined in the configuration file but in this quickstart document we will accept the defaults. We do however need to define some interfaces to make kplex do anything useful. Each interface can be defined by writing its type in square brackets. To declare a serial interface (i.e. a data connection over an attached serial line) you would put:

[serial]

into the configuration file. Underneath this you then add statements in the format "variable=value" to tell kplex things about the interface such as the device name to use and the baud rate to set it to. Everything you write under an interface declaration in square brackets is considered to be part of the same interface declaration until kplex sees a declaration for another interface in square brackets. As elsewhere, everything after a "#" is ignored. Here's a basic declaration for a serial input interface on /dev/ttyUSB0, running at 38400 baud:

[serial]

# This is a comment and ignored

filename=/dev/ttyUSB0

direction=in

baud=38400

Where possible kplex assumes sensible defaults for values. The standard baud rate for NMEA connections is 4800 so if you don't specify "baud=" this is what kplex will use. kplex assumes your connections are bi-directional unless you tell it otherwise with "direction=in" or "direction=out". It doesn't hurt to explicitly state the default values (e.g. "direction=both"). Some parameters need stating ("filename=" must be stated explicitly for serial interfaces).

kplex has quite a few options for the various interface types though not many obligatory ones. They're all documented on the [configuration page](http://www.stripydog.com/kplex/configuration.html).

### Serial to Network Conversion

This is what most people want to use kplex for and it's easy to set up. We've already seen how to configure a serial interface for receiving data. To transmit your data over a network your computer needs to have a network interface. This might be an ethernet conection (perhaps connected to a wifi router) or a wireless connection. kplex can interact with networks in many different ways but here we'll only talk about the two methods most commonly used.

kplex works happily with IPv6 (the "next generation" Internet Protocol ("IP")) but here we'll only talk about IPv4 which most users are more familiar with.

Marine applications most commonly want to receive data either by connecting to a [TCP Server](https://en.wikipedia.org/wiki/Transmission_Control_Protocol) or by listening for [UDP](https://en.wikipedia.org/wiki/User_Datagram_Protocol) packets. TCP and UDP are methods for data transmission over IP networks, the distinction (which is better described in many places) being that TCP involves a sender and a receiver communicating with each other to ensure that all data are reliably delivered in the correct order, whereas UDP involves a sender transmitting messages which may be received by one or more receivers but which (in the absence of something explicitly added by a programmer) receivers do not acknowledge, so if a message is not seen by one or more of the receivers, the sender is not aware and does not attempt to re-transmit it.

UDP messages ("packets") may be sent to a single receiver, a group of receivers or all devices on a network according to the address used. Marine data applications which use UDP generally use the latter method which is referred to as "UDP broadcast".

Which is the best method? "ça dépend". UDP is much lower overhead. A single packet transmitted may be received by many devices whereas TCP requires data to be transmitted over separate connections for each sender/receiver pair. TCP has overheads in the communication between sender and receiver to ensure that no data are lost or mis-ordered. Often though it doesn't matter if occasional transducer updates are lost. On ethernet networks I prefer UDP for marine data, but for various reasons UDP broadcast over wireless networks can be very unreliable. If practice shows that packet loss is unacceptable with UDP broadcast on your network, use TCP.

You can configure kplex to send data over UDP but also to accept TCP connections (as we shall see).

Modern computers run many processes at once. If a network packet just had an IP address the receiving computer wouldn't know which process to deliver the packet to. A "port" is like a mailbox number in a building. It tells the mail room who the end recipient is. Both TCP and UDP packets contain a "port" number. You can use pretty much any "port" you like so long as sender and receiver agree although ports below 1024 should be avoided as some of these are used by system processes. "10110" is the port reserved with the [IANA](http://www.iana.org) for NMEA 0183 data. You don't have to use that and some commercial marine serial-to-network devices use other ports, but this is kplex's default unless you specify something else, which generally speaking there should be no need to do.

To configure kplex to transmit data over TCP to anything that requests it, add this to your kplex.conf:

[tcp]

mode=server

direction=out

"mode=server" tells kplex to listen for requests for data from other devices rather than connecting to a remote device. We could tell kplex to listen on a particular network interface but here we haven't so kplex will listen on all its network interfaces. As we haven't told kplex which port to listen on it will listen on the default port, 10110 (we could have appended "port=2000" to make it listen on port 2000 instead). "direction=out" tells kplex only to transmit (not to receive) data. If you omit this kplex will both send and receive data on tcp connections, although most apps you want to connect won't be sending data, only receiving.

One declaration of a TCP server interface for kplex will allow many devices to connect: you don't need a separate declaration for each device you wish to support.

To configure an app to connect to this interface, specify protocol to be TCP, address to be the IP address of the computer running kplex, and port to be 10110.

To configure kplex to send data via UDP broadcast to all devices on a connected network we first need to know the broadcast address to send the data to. kplex can work that out for you if you tell it the name of the network interface you wish to broadcast from. If the network interface is called "wlan0", append this to your kplex.conf:

[udp]

device=wlan0

direction=out

If you want to use a port other than the default 10110 you can specify it by adding "port=" as with the TCP declaration described above. Note that prior to version 1.3.1 the above declaration would not work as desired on some platforms if "wlan0" was also configured for IPv6. In those cases it would be necessary to explicity state the broadcast address to use (e.g. "address=192.168.1.255").

To configure an app to connect to this interface, specify protocol to be UDP and port to be 10110.

Putting it all together, here's a configuration file which takes two serial inputs, one from /dev/ttyUSB0 at 38400 baud and another from /dev/ttyUSB1 at 4800 baud (the default, which is why there's no "baud=4800" statement, although you could put that in if you wanted). Data is transmitted over UDP port 10110 and a TCP server supports TCP connection on port 10110:

[serial]

filename=/dev/ttyUSB0

baud=38400

direction=in

[serial]

filename=/dev/ttyUSB1

direction=in

[tcp]

mode=server

direction=out

[udp]

device=wlan0

direction=out

## ystem Requirements

Kplex compiles and runs on Gnu/Linux MacOS X, FreeBSD and NetBSD. It should be possible to get it to compile on any POSIX-compliant system without too much effort. As kplex tries not to be hardware dependent and should run on 32 or 64 bit, big or little-endian systems.

kplex has been compiled on some router OSes such as OpenWRT and Tomato.

The primary development and testing platforms are GNU/Linux on x86-64 and ARMv6 (Raspberry Pi).

## Obtaining kplex

kplex is currently being distributed as a source tar file which should compile on all supported platforms, or for selected platforms as a binary Debian package. The .debs are initially for the two primary development platforms: x86\_64 (compiled on Ubuntu 12.04) and armhf (compiled on Raspbian "Wheezy" on a version 1 Pi model B but works happily on other Pis including the v2).

If enough requests are received for binaries in a specific format (e.g. rpm) these will be considered. MacOS dmgs will not be worked on until some sort of GUI has been constructed. All downloads can be found [here](http://www.stripydog.com/kplex/download.html).

## Installing

### From .deb

dpkg -i ./kplex\_1.0-1\_amd64.deb (substitute the name of the platform-specific package you downloaded as necessary)

There is no compiling required when installing from a binary package, so you're ready to skip to [configuration](http://www.stripydog.com/kplex/configuration.html).

### From Source

#### Unpacking

kplex source is distributed as a compressed tar file. Open a terminal window, change directory to the place where you downloaded kplex and unpack the flie:   
tar zxf *<filename>*   
where *<filename>* is the kplex .tgz file you just downloaded, e.g.:   
$ tar zxf kplex-1.0.tgz  
Now change into the diretory created:  
$ ls  
kplex-1.0 kplex-1.0.tgz  
$ cd kplex-1.0  
$

#### Compiling

kplex requires no special libraries in order to build or run other than what should already be installed on your system.

On MacOS X or GNU/Linux just type "make". On FreeBSD and operating systems where "make" is not GNU make use "gmake" wherever "make" appears in these instructions (gmake is not part of the FreeBSD base system and may need installing).   
$ make  
cc -Wall -c -o kplex.o kplex.c  
cc -Wall -c -o fileio.o fileio.c  
cc -Wall -c -o serial.o serial.c  
cc -Wall -c -o bcast.o bcast.c  
cc -Wall -c -o tcp.o tcp.c  
cc -Wall -c -o options.o options.c  
cc -Wall -c -o error.o error.c  
cc -Wall -c -o lookup.o lookup.c  
cc -Wall -o kplex kplex.o fileio.o serial.o bcast.o tcp.o options.o error.o lookup.o -pthread -lutil  
There should be no errors or warnings reported during compilation. If there are, please let me know details of your system and compiler.

#### Installing

"make install" will install the kplex binary and should be done with root privileges. The default installation location is /usr/bin on Linux systems, /usr/local/bin on other systems. These can be modified using the BINDIR variable. For example, to install kplex in /usr/sw/bin:   
sudo make -DBINDIR=/usr/sw/bin install

#### Invocation at boot time

For portable computers It is not usually required to have kplex start automatically on system boot, but if this is required (e.g. for a server with permanently connected serial lines) a sample debian/ubuntu init script is provided with the source distribution (as kplex.init). An example init script for RedHat/CentOS systems is [here](http://www.stripydog.com/kplex/kplexinit.txt). If kplex is run from init scripts, you are strongly advised not to run it as root and ideally to run in a chroot-ed/jailed environment.

If you install kplex from .deb an init script /etc/init.d/kplex is installed as part of the package. You can make kplex start at boot time by issuing the following command which adds appropriate start/stop links::

sudo update-rc.d kplex defaults

The above change can be backed out at a later point as follows:

sudo update-rc.d -f kplex remove

A future enhancement to this page will be documenting adding kplex startup to launchd (OS X), systemd (newer linuxen) etc.

## Configuring and Running kplex

Full instructions on how to configure and run kplex are given in the README file in the kplex source distribution which is also availble online [here](http://www.stripydog.com/kplex/configuration.html). There is a lot of detail in there so users with basic requirements may prefer to start with the [quickstart instructions](http://www.stripydog.com/kplex/quickstart.html)

## Kplex Configuration

* [Invocation](http://www.stripydog.com/kplex/configuration.html#Invocation)
* [Interface Types](http://www.stripydog.com/kplex/configuration.html#InterfaceTypes)
  + [Serial](http://www.stripydog.com/kplex/configuration.html#serial)
  + [File](http://www.stripydog.com/kplex/configuration.html#file)
  + [TCP](http://www.stripydog.com/kplex/configuration.html#tcp)
  + [UDP](http://www.stripydog.com/kplex/configuration.html#udp)
  + [Broadcast](http://www.stripydog.com/kplex/configuration.html#bcast)
  + [Multicast](http://www.stripydog.com/kplex/configuration.html#mcast)
  + [GoFree](http://www.stripydog.com/kplex/configuration.html#gofree)
  + [Pseudo Terminal (pty)](http://www.stripydog.com/kplex/configuration.html#pty)
* [Filtering](http://www.stripydog.com/kplex/configuration.html#filtering)
* [Failover](http://www.stripydog.com/kplex/configuration.html#failover)
* [Termination](http://www.stripydog.com/kplex/configuration.html#stopping)
* [TAG blocks](http://www.stripydog.com/kplex/configuration.html#tag)
* [Examples](http://www.stripydog.com/kplex/configuration.html#examples)
* [Configuration File Syntax](http://www.stripydog.com/kplex/configuration.html#config)
* [To Do](http://www.stripydog.com/kplex/configuration.html#todo)
* [Q&A](http://www.stripydog.com/kplex/configuration.html#qanda)

### Invocation

kplex -V | [-d 1..9] [-f <filename>] [-o <option>] <interface> <interface> [<interface> ...]

Where:  
The "-d" flag, if specified, will cause kplex to print debugging information, the verbosity of which is controlled by the flag's argument with "1" producing minimal additional debugging information and "9" producing verbose debug information.

<filename> is the path of a configuration file to use where default options and interfaces are defined. If not specified, kplex will look for a configuration file in your home directory (~/.kplex.conf on GNU/Linux, ~/Library/Preferences/kplex.ini on Mac), then /etc/kplex.conf and if nothing is found will use only command line directives. Specifying filename as "-" disables looking for the default configuration files.

<option> is an option specifier for the form:   
<variable>=<value>   
These correspond to configuration options which may otherwise be specified in "global" section of the configuration file (discussed in the [Configuration File](http://www.stripydog.com/kplex/configuration.html#config) section below). Command line configuration options always override options given in the configuration file.

<interface> is an interface specifier which takes the form:

* <interface\_type>:<option>[,<option> ...] where:
  + <interface\_type> is one of:
    - "serial": serial nmea 0183 data
    - "file": file (or standard in/out)
    - "tcp": nmea over TCP
    - "udp": nmea over UDP (unicast/broadcast/multicast)
    - "broadcast": nmea over UDP broadcast (DEPRECATED)
    - "bcast": synonym for "broadcast" (DEPRECATED)
    - "multicast": nmea over UDP multicast (DEPRECATED)
    - "mcast": synonym for "multicast" (DEPRECATED)
    - "pty": serial nmea data over a pseudo terminal
    - "gofree": nmea over TCP announced by Navico's "GoFree" protocol
  + <option> is one or more options of the form <var>=<val>. Some options are not optional. Options applicable to all interface types are:
    - "direction": May normally be one of "in" specifying an input, "out" specifying an output, or "both" specifying a bi-directional interface. Not all directions are applicable to all interface types. The default is "both" if an interface type permits bi-directional communication. It is more efficient to specify "in" or "out" if that is all an interface needs to do.
    - "qsize": Size of the interface's output queue. Not used for input only interfaces. Defaults should be fine. This should only need to be increased from default in the case of a bursty high-speed input feeding a slow ouput.
    - "checksum": May be "yes" to enable checksumming of incoming sentences on an interface or "no" to disable it. This option overrides the global checksum option.
    - "strict": May be "yes" to enable strict parsing of incoming sentences on an interface or "no" to disable it. This option overrides the global strict parsing option.
    - "loopback": May be "yes" to enable sentences read from an interface to be written back to it when communication is bi-directional or "no" (the default) to ensure sentences are not looped back to the interface they were read from. This option should be used with extreme caution and generally not at all with broadcast or multicast interfaces. This option has no effect on unidirectional interfaces.
    - "ifilter": Specifies an input filter (see below)
    - "ofilter": Specifies an output filter (see below)
    - "name": Attaches a symbolic name to an interface. This is only required if you intend to use the interface for failover (see below) but can be helpful for debugging. The value associated with "name" can be any string consisting of letters and or numbers which is not used as the "name" for another interface. If no "name" is assigned kplex will autogenerate one starting with an underscore ('\_'). For this reason names starting with an underscore should not be manually assigned. Names are NOT case sensitive, so do not call one interface "Serial" and another "serial".
    - "srctag": Specifies that an NMEA-0183v4 TAG block containing a source identifier be prepended to sentences output on the interface if the value is "yes" or "input". The src identification string is the interface name (see the "name" option above) truncated to 15 characters or the string "kplex" if the interface name starts with an underscore as is the case with autogenerated (non-user-specified) names. A TAG block prepended by the "srctag=yes" option looks like: \s:kplex\*23\ If "srctag=no" is specified, no source identifier is prepended. This is the default.  
      If "srctag=input" is specified, the src identification string is the name of the interface on which the sentence arrived truncated to 15 characters or the string "kplex" if no "name" is specified for the input interface.
    - "timestamp": Specifies that an NMEA-0183v4 TAG block containing UNIX time (for example "\c:1423133110\*5C\" should be prepended to sentences output on the interface. The timestamp is in seconds if the value is "s" or milliseconds if the value is "ms". Note that NMEA-0183v4 timestamps do not take account of leap seconds.
    - "optional": If "optional=no" is specified or this option is not given, kplex will exit if it cannot initialize the interface. If "optional=yes" is specified, failure of the interface to initialize will only cause kplex to exit if, as a result of it failing, kplex has no inputs or no outputs.

If source identifier and timestamps are both requested for an interface, they are combined into a single TAG block, source identifier first, e.g.: \s:kplex,c:1423133048\*5F\

The -V flag instructs kplex to print its version number and exit. It should not be used with any other options.

### Interface Types

#### Serial Interfaces

This is the traditional way of getting nmea data into your computer. kplex doesn't care whether your device is connected to via a traditional serial port or via a USB to serial converter so long as the device on which data are presented looks like a character special device and it can be configured with baud rate and a minimal set of other parameters. Note that you can't normally just plug your nmea tx/rx into a serial port. NMEA is RS422 whereas serial ports normally want RS232 input.

Interface-specific options:

* filename=<device>
* baud=<baud>

Where

* <device> is the serial device (e.g. /dev/ttyS0)
* <baud> is the baud rate. Defaults to 4800 if unspecified. Supported baud rates are: 4800, 9600, 19200, 38400, 57600, 115200

You must minimally specify a device name for a serial interface. usb to serial converters often use /dev/ttyUSB0. Check your /var/adm/messages file and/or udev rules. Note that normal users are often not permitted to open serial devices. This may mean adding your user to a group which \*is\* allowed to read the device (e.g. "dialout", "uucp" or whatever).

#### "File" interfaces

This specification covers regular files, FIFOs and terminal i/o via standard input and standard output.

Interface-specific options:

* filename=<file>
* persist=[yes|no]
* append=[yes|no]
* eol=[rn|n]
* owner=<user>
* group=<group>
* perm=<permissions>

Where

* <file> is either the file name to read from or write to or "-". In the latter case, standard input is used for inputs, standard out for outputs, and standard in and standard out for "BOTH". If not specified defaults to "-".
* <user> is the user to set ownership of a created output file to.
* <group> is the group to set a created output file to.
* <permissions> are the file access permissions, in octal form, to set a created output file to.

"File" interfaces are slightly different from other interfaces in that by default sentences are terminated by <LF> rather than <CR><LF>. Because this is \*nix and we don't want to muck about with redundant '\r's. Sentences input with just a terminating '\n' (ie <LF>) are converted to <CR><LF> for output to other interface types (i.e. serial, tcp, broadcast etc.). Sentences terminated by the standard NMEA-0183 <CR><LF> will be rejected on input unless "strict=no" is specified, although this behaviour may change in future releases.

If the option "eol=rn" is specified, file interfaces will output sentences terminated by <CR><LF>. Input sentences will be discarded if they are terminated with <LF> not preceded by <CR> and will not have additional <CR>s added.

"eol=n" specifies that the default behaviour for file interfaces (<LF> only as sentence delimiter) should be used.

"BOTH" is not supported as an I/O direction for the "file" type except for standard in / standard out.

If no "filename" option is given, standard in/out are used.

If "append=yes" is specified for an interface outputting to a regular file, output is appended to the file. If "append=no" (the default) is specified, the file will be truncated before output is written. This option may not be specified for input files, FIFOs or terminals.

"persist=yes" may only be specified on an interface connected to a FIFO. If specified it will re-open a FIFO when the reader or writer at the other end closes the pipe. If "persist=no" (the default) is specified, an input interface will exit on receipt of EOF. An output interface will exit when the reader at the other end of the pipe exits.

FIFOs block on open for read until something opens the FIFO for writing, and block on open for write until something opens them for reading. To avoid hanging kplex's initialisation thread, opening of FIFOs is delayed until individual reader and writer threads have been created. Output to file interfaces is line buffered.

For output to regular files, if the specified filename does not exist it will be created if permissions allow. If kplex creates an output file, it will be owned by the user of the kplex process unless the "owner=" option is specified, in which case the file's ownership will be set to the specified username. On most systems, kplex can only set username to something other than the owner of the kplex process when run as root. The group of an output file created by kplex will be set to the kplex process's primary group unless the "group=" option is specified in which case the file's group will be set to the specified group. The group must exist and the kplex process must have permission to set a file to that group. Normally this requires the kplex process's owner to be a member of the specified group. An output file created by kplex is normally readable and writable by user and group and readable by "other", modified by the user's umask. If "perm=" is specified, the permissions on a file created by kplex are set to the option's argument in octal \*unmodified\* by the processes's umask. Thus "perm=0666" will make the file readable and writable by user, group and other regardless of the kplex process's umask.

For output files which pre-exist and for all input files, the user,group and perm options are ignored.

#### TCP Interfaces

kplex can act either as a tcp server (allowing other programs on the same or other machines to connect to it) or as a tcp client, connecting to servers on the same or other systems. In "client" mode, kplex will attempt to connect to the server running on the port and address you specify. This may be a kplex tcp server interface or a commercial product, such as an nmea to wifi interface box. You must specify an ip address (IPv4 and IPv6 supported) for a tcp client connection. You may specify a port. If not specified, the port used will be the one your system associates with "nmea-0183", or the IANA assigned 10110 if your system doesn't know about "nmea-0183". If using kplex, the server port defaults to 10110. Consult manufacturer's documentation for the port to use for other products.

In Server mode, kplex listens for incoming connections. These may be from programs like iNavX or other instances of kplex. No data flows until a connection is made. kplex can accept many client connections simultaneously. The exact number will be system dependent but it will certainly be "enough".

Interface-specific options:

* mode=<mode>
* address=<address>
* port=<port>
* persist=[yes|no|fromstart]
* retry=<seconds>
* preamble=<preamble>
* gpsd=yes|no]
* timeout=<timeout>
* sndbuf=<sndbuf>
* nodelay=[yes|no]
* keepalive=[yes|no]
* keepidle=<keepidle>
* keepintvl=<keepinterval> ( Not Mac OS X < 10.9 )
* keepcnt=<count> ( Not Mac OS X < 10.9 )

Where:

* <mode> is either "server" or "client". If not specified, defaults to "client".
* <address> is the server address to bind to for output interfaces and the remote tcp server to connect to for inputs. This can be:
  + A symbolic hostname (which must be resolvable on the host)
  + An IPv4 address in dotted decimal format
  + An IPv6 address

For input interfaces, a remote host MUST be specified. For outputs, if address is not specified, or specified as "-" a wildcard address will be used and the server run on all available interfaces. Whether this is IPv4 only or IPv4 and IPv6 will depend on system configuration. IPv6 can be forced (if your system supports it) using a wildcard address of "0::0". An IPv6 server can accept IPv4 connections on a dual stacked host. You can happily ignore all mention of IPv6 if you want to. The address, if specified for an output, must correspond to one assigned to a interface on the host

* <port> is the tcp port to run the server on. If not specified, defaults to the tcp port returned by a lookup of the service "nmea-0183" and if that fails the IANA assigned port for nmea-0183 10110 is used.
* <seconds> is the number of seconds Default 5) to wait before each attempt at reconnecting a lost tcp connection. The "retry" option is only valid in conjunction with "persist=yes" or "persist=fromstart"
* <preamble> is a string of characters to send after connecting to a remote server and before sending data, as described below.
* <timeout> is the number of seconds to wait for an output operation to complete before assuming a connection has died, abandoning the data send operation and attempting to reconnect the interface. Only valid with output or bi-directional interfaces and only in conjunction with "persist=yes" or "persist=fromstart". Note that timeouts will not occur while data can still be passed to TCP, so the size of TCP buffers has an impact on how quickly a hung connection will be timed out.
* <bufsize> is the size in bytes to set the TCP output buffer to. This option is valid only with "persist=yes" or "persist=fromstart" and defaults to 2048. Without "persist=yes" or "persist=fromstart" system default TCP buffer size is used. A buffer size of 2k should not negatively impact performance in this application. A smaller output buffer size generally results in hung output connections being detected faster.
* <keepidle> is the number of seconds of inactivity on a tcp connection to wait before sending the first keepalive probe (see below). Only valid with "keepalive=yes".
* <keepinterval> is the number of seconds to wait between each keepalive probe after the first (see below). Only valid with "keepalive=yes" and not available for Mac OS X prior to Mavericks.
* <count> is the number of un-replied to keepalive probes (see below) before a tcp connection is considered lost. Only valid with "keepalive=yes" and not available for Mac OS X prior to Mavericks.

For most purposes you can just specify "tcp:direction=both,mode=server" to create a bi-directional tcp server.

If a client ("mode=client", the default) tcp connection is lost for any reason, kplex will not attempt to reconnect if "persist=no" (the default) is specified. The interface will shut down, but other interfaces will continue to operate. If "persist=yes" is specified for a client connection, kplex will attempt to reconnect when the connection is lost. When attempting to reconnect an outbound or bi-directional connection, kplex will discard all but the last message in its queue to minimise the amount of potentially stale data arriving at the server. The delay in seconds between successive attempt at reconnection may be specified using the "retry" option. "persist=yes" only tells kplex to reconnect a lost connection. If the first connection attempt fails it will not be re-tried and initialisation of that interface will fail. If persistent attempts to connect an initially failed connection are desired, "persist=fromstart" should be specified. Note that this option should be used with care to avoid repeated attempts to connect to a mis-typed hostname or address.

kplex will detect a dropped connection if the other end closes down "cleanly", i.e. the program it is connecting to shuts down or the machine it is running on is gracefully shut down. kplex may also detect a failed endpoint if it gets no response to data it is attempting to write. It may not, however, be able to detect a failed endpoint if it is purely reading from the other end and it is not notified that the data source has gone away. This may frequently happen if connecting from behind NAT (NAT mappings are lost) or if the other computer crashes or has power removed. In these cases it is useful to specify the "keepalive=yes" option (the default is "no"). This will cause tcp to send probes to the remote end point to check that it is still "alive". The first probe is sent after seconds of inactivity. The default value will be system dependent but is usually 2 hours. If no reply is received to this probe, further probes are sent <keepinterval> seconds apart. If <count> probes are sent without reply, kplex considers the connection dropped and will attempt to reconnect. The default values for the interval between keepalive probes and the number sent before the connection is considered lost are system dependent, but invariably higher than desirable for kplex's purposes. On Mac OS X versions prior to Mavericks, only the delay before the initial probe is configurable from within kplex, but the other two values are configurable on a system-wide basis with sysctl(8).

If "persist=yes" is specified for a tcp client connection but no "keepalive=" option is given, keepalives will default to "on" and unless explicitly specified otherwise, the following values will be set:

* keepidle=30
* keepintvl=10 (Not Mac OS X prior to 10.9)
* keepcnt=3 (Not Mac OS X prior to 10.9)

If "nodelay=yes" is specified or the nodelay option is not used, kplex will disable the nagle algorithm on an outbound tcp connection. This results in fractionally faster delivery of data to clients at the expense of slightly more network traffic and is generalyl desirable on a local area network. Where minimising network use is a priority (such as sending data over a mobile data connection with a per-megabyte charge) specifying "nodelay=no" can reduce network traffic at the expense of a slight increase in latency.

The "preamble" option is used to send a set of characters to a remote server to identify a sending station before transmitting data. It is not part of the NMEA-0183 specification (for standards-compliant source identification, use source TAGs instead) but is used by some AIS aggregation sites as an alternative to per-station dedicated TCP ports. If specified with the "persist" option, the preamble string will be sent after each reconnection following connection loss. Non-ASCII characters may be specified either by a backslash followed by the (exactly) 3-digit octal representation of the character or the sequence "\x" followed by the (exactly) 2-digit hexadecimal representation of the character. The NULL character must be specified as "\000" or "\x00" and may not be abreviated to "\0" or "\x0". Standard escape sequences (\a,\b,\f,\r,\n,\t,\v) are recognised. Other escaped characters are sent literally without the leading backslash. Any string termination must be explicitly stated, so a NULL- terminated string must end with "\000" or "\x00". Care must be taken when using this option on the command line due to shell interpretation of escape characters. Only one "preamble" may be specified and this option may not be used with "mode=server".

If "gpsd=yes" is specified kplex will use port 2947 as a default if the "port" option is not specified and will set the preamble to: ?WATCH={"enable":true,"nmea":true} This will enable nmea output from an instance of gpsd connected to. This option may not be used with "mode=server" or the "preamble" option.

#### UDP Interfaces

**NOTE:** As of kplex 1.3 UDP interfaces are now preferred over the existing "broadcast" and "multicast" interface types. However the "udp" interface is new in this version. If you encounter problems, please report the issue and consider reverting to the older "broadcast" or "multicast" interface types while your problem is under investigation.

This method encapsulates nmea sentences within UDP datagrams which are sent via unicast, multicast or broadcast.

Interface-specific options:

* address=<address>
* port=<port>
* device=<interface>
* type=[unicast|broadcast|multicast]
* coalesce=[yes|no]

Where:

* <address> is the interface address to bind to for inbound kplex interfaces or the address to send to for outbound interfaces. If not specified for an input interface, kplex will receive broadcast and unicast traffic sent to the desired port on a given interface if one is specified, or all interfaces if non is specified. If no address is specified for an output or bi-directional interface, a system interface must be specified, in which case the broadcast or point-to-point destination address is assumed for outbound datagrams with inbound datagrams received for the address of the specified interface in the case of point-to-point system interfaces or the interface's broadcast address where the system interface is broadcast-capable. For sending and receiving of multicast datagrams an address must be specified. kplex should work out whether any provided address is unicast, multicast or broadcast and act accordingly. "group=" is a synonym for "address=" for backwards compatibility with deprecated mcast interfaces.
* <device> specifies the system interface (e.g. "wlan1", "eth0") to use. If the kplex interface is inbound and an interface is specified, kplex will attempt to bind to an address if one is specified which belongs to that system interface, or the first address found associated with that system interface if no address is specified. For bi-directional and outbound multicast and broadcast interfaces all traffic is sent and received using a specified system interface. For bi-directional and outbound unicast interfaces any received traffic will be on the interface specified. For outbound datagrams the source address will be set to an address associated with the specified interface. If no device is specified for inbound or outbound multicast interfaces, the routing table is used to determine which system interface to use. For broadcast and unicast interfaces inbound traffic will be received on the system interface corresponding to a specified broadcast or unicast address where no system interface is specified or any system address if no device or address is specified.
* <port> if specified is the udp port or service name. If not specified defaults to the udp port returned by a lookup of the service "nmea-0183" and if that fails the IANA assigned port for nmea-0183 10110 is used.

Note that broadcast is inherently IPv4 (it does not exist in IPv6) and inefficient, forcing all nodes on a network to process data which they are potentially uninterested in. Multicast is the superior technique but not as widely supported by other marine applications.

Bi-directional unicast interfaces do not necessarily mean what you think they mean. Outbound packets will be sent from an ephemeral port and will be received from \*any\* sender of UDP packets containing valid NMEA-0183 data on that port. This is invariably not what you want and it is normally better to use separate send and receive interfaces for bi-directional communication over UDP between two instances of kplex.

Generally kplex can work out whether an interface should use unicast, multicast or broadcast traffic from the supplied <address> so the "type" option should not normally be given and doing so to force an mode not consistent with the supplied address will result in an error. One use for the "type" option is when supplying a "device" but no "address" option to an inbound interface. In this case the receiving interface will expect unicast traffic unless "type=broadcast" is explicitly requested.

By default or if "coalesce=no" is specified, input sentences are always transmitted one per packet as soon as they can be. If "coalesce=yes" is specified, kplex will buffer parts of a multi-part AIS message so long as enough space is available (512 bytes, a minimum of 7 sentences depending on size, can be buffered). Buffered data will be transmitted if the last part of the message is received (even if all the intervening parts have not been), there is insufficient space to store another sentence, or a sentence arrives which is not part of the buffered message. In the latter cases the newly arrived sentence is buffered if it is part (but not the last fragment) of a multi-part AIS sentence, otherwise it is transmitted immediately. kplex does not re-order out of order fragments of a multi-part AIS message.

#### Broadcast Interfaces

Broadcast interfaces are now deprecated and will be removed from a future version of kplex. Use udp interfaces instead if possible.

This method involves nmea sentences encapsulated within UDP datagrams sent to a broadcast address.

Interface-specific options:

* device=<interface>
* address=<address>
* port=<port>

Where:

* <device> specifies the system interface (e.g. "wlan1", "eth0") to use. This must be specified for outbound or bi-directional interfaces. kplex will only broadcast out through one interface. If the kplex interface is inbound and an interface is specified, kplex will attempt to bind to that interface and only accept packets received on that interface. Unfortunately this is a privileged operation on most GNU/Linux systems and kplex will often silently fail to do this without root privileges. You can possibly achieve a similar effect without root privileges by use of the <address> specifier (see below). If interface is not specified (or given as "-"), kplex will listen on all interfaces unless an <address> is specified (see below).
* <port> if specified is the udp port or service name. If not specified defaults to the udp port returned by a lookup of the service "nmea-0183" and if that fails the IANA assigned port for nmea-0183 10110 is used.
* <address> is the IPv4 interface address to bind to for inbound kplex interfaces or the address to send to for outbound interfaces. If unspecified for an input, kplex will receive broadcast and unicast udp to the relevant port on any <interface> specified, or all system interfaces if none was specified or the user as insufficient privileges to bind to a specific system interface. If an address is specified for an inbound broadcast interface, kplex will receive only packets to that address. Note that if you specify the IP address of a system interface, you will NOT receive broadcast traffic. If you specify a broadcast address (either the subnet broadcast address or the "all hosts" broadcast address 255.255.255.255 you will ONLY receive that \*type\* of broadcast (ie subnet OR all hosts). For this reason, this parameter is best left unspecified by most users. For outbound connections, this parameter specifies the broadcast address to use. It must be a broadcast address appropriate for the system interface you have specified and will default to the subnet broadcast address associated with the first address found for the specified system interface. If your client programs are particularly stupid they may be expecting the all hosts broadcast address of 255.255.255.255. If things don't work with the default, try this in the <address>.

Note that broadcast is inherently IPv4 (it does not exist in IPv6) and highly inefficient, forcing all nodes on a network to process data which they are potentially uninterested in. Multicast is the superior technique but not supported by many (if any) marine navigation applications at present.

#### Multicast Interfaces

Multicast interfaces are now deprecated and will be removed from a future version of kplex. Use udp interfaces instead if possible.

Multicast interfaces are similar to broadcast interfaces, but available with IPv6 as well as IPv4 and more efficient. In an ethernet network, a broadcast packet will require all nodes on a network to pass the packet to their IP stacks to determine whether or not it is of interest to them. IP multicast addresses map to ethernet multicast addresses. An operating system tells its network interface to accept only multicast packets with hardware addresses it is interested in. The mapping is not 1:1 (many IP multicast addresses map to one ethernet multicast address) but on a busy network, use of multicast instead of broadcast can dramatically cut down the number of packets a given node's network stack needs to process.

Interface-specific options:

* group=<multicast address>
* device=<interface>
* port=<port>

Where:

* <multicast address> is the multicast group address. This must be specified.
* <interface> is the network interface to use (e.g., "eth0", "wlan1" etc.). If unspecified, if a bind address is specified, the system interface assigned that address will be used, otherwise the choice of interface will be left to the system and normally based on the routing table.
* <port> if specified is the udp port or service name. If not specified defaults to the udp port returned by a lookup of the service "nmea-0183" and if that fails the IANA assigned port for nmea-0183 (10110) is used.

A multicast group address to used must be specified for an "multicast:" interface. For link local IPv6 multicast addresses, an interface device must be specified. This may be done in one of two ways:

1. by appending the multicast group address with "%" followed by the interface name, for example:
2. group=ff02::a0:200%wlan0
3. by specifying the interface with the "device" option, e.g.:
4. device=wlan0

If a device is not specified in one of the above ways for multicast addresses other than IPv6 link and interface local groups, the routing table will be used to select the outgoing interface for multicast packets.

#### GoFree Interfaces

GoFree is Navico's service discovery protocol which allows applications to connect to a network services without knowing details of its address. A kplex gofree interfaces listens on the IPv4 multicast address (239.2.1.1) and port (2052) which Navico have specified for announcements of the "nmea-0183" service. If not currently connected to an announced "nmea-0183" service, a kplex gofree interface will attempt to initiate a connection to the unicast TCP IPv4 address/port found in the first appropriate service announcement it sees. If a gofree interface is currently connected to an nmea-0183 service, On receipt of an announcement for an alternate service location (i.e. the IPv4 address/port of an nmea-0183 service on another multifunction display ("MFD"), if the last announcement for the currently connected service was more than 2 seconds prior, the gofree interface will terminate the current service connection and reconnect to the newly announced service. If the last announcement for the current service was less than 2 seconds prior, kplex will only initiate a connection to the alternate service if the current connection has terminated.

Interface-specific options:

* device=<interface>

Where:

* <interface> specifies the system interface (e.g. "wlan1", "eth0") to use. If unspecified the system will select the interface to listen for service announcements on, normally defaulting to the first multicast-capable non-loopback device.

GoFree does not support bi-directional nmea-0183 connections so all gofree interfaces have an implicit "direction=in" option. It is an error to specify "direction=out" for a gofree interface. Any output filters specified for a gofree interface are ignored.

#### Pseudo Terminal (pty) interfaces

Pty interfaces are pretty much the same as serial interfaces except that the devices concerned do not correspond to physical input and output devices on your system. Actually, in the case of inputs it makes no difference whether you specify "serial:" "pty:": The code ends up going down the same path.

Where ptys come in handy with kplex is if you want to split a serial input between one or more programs running on a computer and possibly some outputs too.

Interface-specific options:

* mode=<mode>
* filename=<file>
* baud=<baud>
* owner=<user>
* group=<group>
* perm=<permissions>

Where

* <mode> is either "master" or "slave"
* <file> is either the pty to connect to in "slave" mode or, in "master" mode, a path name specifying a symbolic link that will be created pointing to the slave side of an output pty
* <baud> is the baud rate. Defaults to 4800 if unspecified. Supported baud rates are: 4800, 9600, 19200, 38400, 57600, 115200
* <user> is the username for the slave side of a master pty to be set to.
* <group> is the group to set the slave side of a master pty to.
* <permissions> are the permissions in octal form to set the slave side of a master pty to.

<file> must be specified in slave mode. In master mode, kplex creates a master/ slave pty pair. If you give kplex a <file> it will attempt to create a symbolic link with that pathname pointing to the slave side of the pty it creates. If the path given currently exists as a symbolic link it will be replaced. If it exists but is not a symbolic link (e.g. it's a regular file or device) kplex will exit with an error. If no pty name is given, or if it is given as "-", kplex just prints the name of the slave pty created without creating a symlink.

Specifying a pathname (and creating a symlink) is useful for providing a persistent interface.

Where kplex is directed to create a master pty interface with "mode=master", the slave side of the pty (which other processes will use to communicate with kplex) will be created using system default ownership and permissions. On many systems this will mean the owner of the device will be the owner of the kplex process, the group will be set to "tty" and the permissions will be 0620 (i.e. read/write by owner, write only by group and inaccessible to others).

if "owner=<user>" is specified, the owner of the slave side of a master pty will be set to <user> if <user> is a valid system user and the owner of the kplex process is permitted to change the file's ownership in this way. Normally this may only be used by a kplex process running as root.

If "group=<group>" is specified, the group of the slave side of a master pty will be set to <group> if <group> is a valid system group and the owner of the kplex process is permitted to change the file's group in this way. Normally only root or a member of the specified group is able to make this change.

If "perm=<permissions>" is specified where <permissions> are the desired permissions of the slave side of a master pty in octal format, permissions are set accordingly. Only file access bits are settable. Attempts to set setuid/setgid/sticky bits will be ignored. A leading "0" is allowed but not required for permissions. Note that "000" is neither a useful nor, in this case, permitted access mode.

As an example, Assume you wish to take AIS input from a serial port, make it available to opencpn but also create a tcp server to make the data available to inavX on an ipad. The user of OpenCPN is a member of the dialout group but not tty. You might invoke kplex like this:

kplex serial:direction=in,filename=/dev/ttyUSB0,baud=38400 pty:direction=out,mode=master,filename=/home/fred/.opencpn/ais,baud=38400,group=dialout,perm=640 tcp:direction=out,mode=server

And add the following to /home/fred/.opencpn/opencpn.conf:

[Settings/AISPort]  
Port=Serial:/home/fred/.opencpn/ais

This creates the /home/fred/.opencpn/ais, which opencpn will use for its AIS input, as a symlink to the slave of a pty which kplex opens at 38400 baud in addition to a tcp server.

### Filtering

kplex allows you specify two types of filter: Input and Output Input Filters dictate what sentences an input interface forwards Output filters dictate which sentences get passed out of an output interface. An input filter is used by input and bi-directional interfaces but ignored by output interfaces. Likewise, an output filter is used by output and bi-directional interfaces and ignored by input interfaces. Connections spawned by servers inherit their parent's filters, so connections to a tcp server will be filtered according to the server's filters. A filter consists of a series of filter rules. Each filter rule consists of either a "+", "-" or "~" (to specify an "ALLOW", "DENY" or "LIMIT" rule respectively) followed by a "match string" which is either the word "all" or 5 characters. The match string may optionally be followed by the "%" character and the name of an interface (which must have been given to an interface using the "name=" option). A "LIMIT" rule must additionally have a "/" character followed by a whole number (i.e. without a decimal point) representing the minimum number of seconds which must pass between successive sentences matching that rule being permitted to pass. Filter rules are separated by a colon (":" character). Filter rules are applied in the order they are specified to a sentence being filtered.

A filter rule which specifies the word "all" matches all sentences. If a filter rule specifies a 5 character match string, these are compared with the 5 character NMEA 0183 talker/message type of the sentence being filtered. The filter rule matches if each character is the same as the corresponding character in the sentence being filtered. A "\*" in a filter matches any character. Thus a filter rule specifying: GP\*\*\* would match any sentences produced by a GPS talker (excluding any proprietary sentences not specifying talker as "GP").

If a source interface has been specified for a rule, a given sentence must additionally have entered kplex from an interface with the specified "name" for the rule to "match". Thus a filter specified as GP\*\*\*%Serial1 would match sentences with a talker id of "GP" only if received on the interface with the name "Serial1".

When a filter rule "matches" a sentence, it "fires". If the rule was an "allow" rule (ie prepended by a "+", the sentence is allowed. If the rule was a "deny" rule (ie prepended by a "-"), the sentence is dropped. If the rule was a "limit" rule, the sentence is passed if and only if time in seconds since the last time a sentence matching this that rule was allowed to pass was equal to or greater than the number of seconds following the "/" in the rule specification.

If no rules are matched the sentence is allowed. Thus a filter such as: ifilter=+GP\*\*\*:+AI\*\*\*:+SDDBT is pointless. It allows all sentences as it denies none. To \*only\* allow AIS, GPS and Depth below transducer sentences, you need to deny what is not explicitly allowed by adding "-all" to the end of the filter specification: ifilter=+GP\*\*\*:+AI\*\*\*:+SDDBT:-all Obviously order is important. Putting "-all" at the beginning would simply deny all sentences.

Specifying an interface in a rule applied to an ifilter is generally pointless.

**IMPORTANT NOTE:**   
proprietary sentences start with "$P" followed by a three character vendor code, followed by a vendor-specified string which may be, and often is, more than one character in length. kplex can only filter on the 5 characters following the "$" and thus cannot precisely filter all proprietary sentences. Similarly kplex cannot completely filter query sentences which consist of "$" followed by the two letter talker id of the requester, the two letter talker id of the target talker, the character "Q", a comma and the three character sentence mnemonic. Only the 5 characters after the "$" (ie requester/target pair) can be filtered on.

### Failover

kplex allows you to specify special filters which are intended to allow you to use a particular source for one type of data if it is available, but allowing that type of data to be passed from another interface if it hasn't been seen on the preferred input interface for a specified period of time. This behaviour is specified using the "failover=" directive in the [global] section of the configuration file or as a -o option on the command line. The format is: failover=<filter>:<delay>:<interface>[:<delay>:<interface>]... Where:

* <filter> is a filter specifier as described in "Filtering" above
* <delay> is the number of seconds without seeing data which matches the filter on a higher priority interface before the datum is passed
* <interface> is the name of the interface to which the <delay> specifier applies. An interface must be given a "name=" option to be usable with failover.

Any number of :<delay>:<interface> specifiers may be added. Any interface not specified on a "failover" line will never pass sentences matching the filter.

"Primary" interfaces (ie those where the data should "normally" come from) should be specified with a <delay> of 0.

Example:   
You have 3 GPS sources available. The main GPS is fed via a serial connection you have named "serial1". A second is available from a USB GPS you have named "USBpuck". As a last resort you have your phone transmitting nmea over tcp on an input you have named "phone". You might specify: failover=GP\*\*\*:0:serial1:30:USBpuck:60:phone This will always pass sentences with a talker id of "GP" from the interface named "serial1". If no "GP" sentences are seen on serial1 for 30 seconds, kplex will pass sentences matching "GP\*\*\*" from USBpuck until a "GP\*\*\*" sentence is next seen on serial1. If no "GP\*\*\*" sentences are seen on either serial1 or USBpuck for 60 seconds, kplex will start passing such sentences from the "phone" connection until such point as those sentences are seen on either of the higher priority interfaces.

### Stopping

kplex closes down if it has no more outputs. If kplex has no more inputs, it closes down after all outputs have transmitted any buffered data. Interfaces shut down when the end of data input is reached (e.g. on end of file for file inputs or a network peer terminates its end of the connection) or on error. Outputs terminate when they are unable to write due to a network peer terminating or some error condition.

If the process receives a SIGTERM (e.g. from the kill command) or SIGINT (e.g. from ctrl-C pressed at the terminal kplex is running in), kplex will shut all its interfaces down, allowing any buffered data to be transmitted, before exiting.

To stop an instance of kplex which is running in the foreground, hold down the "Ctrl" key and hit "c".

To stop an instance of kplex running in daemon mode, send it the termination signal, e.g. "pkill kplex".

kplex should always clean up all of its interfaces, including restoring serial line settings to what they were when kplex started. If this doesn't happen it's a bug: Please report it.

### NMEA-0183v4 TAG block handling

kplex will strip all NMEA-0183v4 TAG blocks from the input stream and discard them. Correctly formatted following sentences will be multiplexed. kplex does not conform to NMEA-0183v4 and will ignore all query and control messages.

Timestamps and source identifiers conform, as far as can be determined from publicly available sources, to the NMEA-0183v4 TAG specification. Note however that at this time implementation of TAG block handling in various programs and devices is variable and other programs may discard sentences with correctly formatted TAG blocks prepended

### Example usage

Inputs from ais data on one serial port, other nmea data on a serial to usb interface. Output to broadcast udp and one usb to serial interface:

kplex serial:direction=in,filename=/dev/ttyS0,baud=38400 serial:direction=in,filename=/dev/ttyUSB0 tcp:direction=out,mode=server serial:direction=out,filename=/dev/ttyUSB1,baud=38400

Bi-directional communication with tcp server on 192.168.1.50, port 2200. Output to pseudo terminal, creating link for opencpn to read and write at /tmp/nmea, 38400 baud

kplex tcp:direction=both,mode=client,address=192.168.1.50,port=2200 pty:direction=both,mode=master,filename=/tmp/nmea,baud=38400

Input from GPS on usb to serial interface, outputting to tcp server, IPv6 multicast group ff05::10:110 on the default port and a data log file which appends an RMC sentence once per hour:

kplex serial:direction=in,filename=/dev/ttyUSB0 tcp:mode=server,direction=out mcast:group=ff05::10:110,direction=out file:filename=/var/tmp/datalog,direction=out,append=yes,ofilter=~GPRMC/3600:-all

### Configuration File syntax

The configuration file syntax is similar to the command line sytanx, but new lines are used to separate options instead of commas and the start of an interface specification is signalled by the interface type enclosed by square brackets as the only non-white space on a line. Everything that follows the start of an interface section is considered an option relating to that interface until the beginning of the next interface section or the end of the file.

Everything after a '#' character on a line is ignored.

A special "interface" type, "global", may be used to specify options not specific to a particular interface. "global" options currently supported are:

* qsize=<qsize>  
  Where:
  + <qsize> is the size (in sentences) of kplex's central multiplexing queue This should not normally need changing.
* mode=<mode>  
  Where:
  + <mode> is either "foreground" (the default) or "background", the former is the default. The latter tells kplex to detach form its controlling terminal and run as a daemon process.
* checksum=[yes|no]  
  Where
  + "checksum=yes" tells kplex to check the checksum all incoming nmea sentences (except for interfaces where per-interface configuration overrides this). Sentences which do not match their calculated checksums are discarded. The default is not to calculate checksums as it is assumed that this will be done by end consumer applications.
* strict=[yes|no]  
  Where
  + "strict=yes" (the default if "strict=" is not specified) tells kplex to require all sentences received to be correctly terminated with a <CR><LF> sequence unless overridden on a per-interface "strict" option or a per-interface "eol=n". If "strict=no" is specified all interfaces will default to a looser parsing strategy which will allow input sentences to be terminated by a <CR>, <LF> or NULL (0x00). This option applies to input sentences only and has no effect on interface output. However terminated on input when loose parsing ("strict=no") is in effect, sentences output from interfaces other than file interfaces will be <CR><LF> terminated.
* logto=<facility>  
  Where
  + <facility> is the syslog facility to use for logging. The default is "daemon", telling kplex to use the LOG\_DAEMON syslog facility. <facility> is the same string as would be used in a syslog.conf(5) file, so to log to LOG\_LOCAL7, specify "logto=local7".
* failover=<failover specification>  
  Where
  + <failover specification> is a series of colon-separated filter:delay:interface tuples as described in the "Failover" section above.

As an example, the first example from the "example usage" section above could be specified in a configuration file:

# This is a comment and will be ignored

[serial]

direction=in

filename=/dev/ttyS0

baud=38400 # baud will be read, but this comment ignored

# whitespace is ignored

[serial]

direction=in

filename=/dev/ttyUSB0

[tcp]

direction=out

mode=server

[serial]

direction=out

filename=/dev/ttyUSB1

baud=38400

# This is the end of the example file

### To Do

Possible future enhancements include:

* A set of commands to allow modifying kplex on the fly, for example to add, subtract and modify the current interface list. The structure of kplex means this would be a relatively straightforward addition
* A nice GUI. Obviously.

### Q&A

Question and Answers have now been moved to a dedicated [FAQ page](http://www.stripydog.com/kplex/faq.html).

## Frequently Asked Questions

Is this free software?

Yes. As in "Beer" and as in "Speech". It is released under the terms of GPLv3 (see the COPYING file which should be included with this software)

Can we use it in our product?

Subject to the conditions of GPLv3, yes. You can optionally hire me to customize and provide support for the code. See the [contact page](http://www.stripydog.com/kplex/contact.html) for how to get in touch

Why "kplex"?

Originally it was called "mplex", but that's the name of an mpeg stream multiplexer for linux. changing only one letter of the name simplified changing the file names. And "kplex" scores loads in french scrabble.

Do you have a git repository?

Yes but not a public one. That will change one day, probably after I've written a hacking guide to the code. Any version of kplex currently on github was not put there by me and probably isn't up to date

How can I contribute code?

Send me a patch but please don't be disappointed/angry if I don't include it.

How else can I help?

Send me a link to a page describing how you use kplex

Are you planning to support Raymarine seatalk?

The code was written for an earlier version of kplex but it wasn't included in a "production" release for a number of reasons which might make support difficult. These included:

* Requirement for additional hardware to convert seatalk to rs232
* Problems with handling of parity errors in many serial to USB converters. Kplex's interpretation of seatalk relies on correct handling of parity errors.
* Difficulty of writing data to the network. To do collision avoidance "correctly" requires real time processing which intriduces a significant level of complexity for the user. Reading is rather easier with less chance of chaos on the seatalk bus.
* Lack of data. I only have data from my own seatalk network. Conclusions I draw regarding timing and ordering of data sent out by various sorts of instrument may not generalize to other Raymarine products

The code may at some point be revived but if it is, it won't be guaranteed to work.

Does this run on a raspberry pi?

Yes. Note the 3.3v weirdness of the serial interface though if not using serial to usb adaptors.

Why is kplex so heavily multithreaded when the target systems are raspberry pis and laptops?

Multithreading suits the program's structure and allows blocking I/O without using poll/select and without resorting to non-portable event notification mechanisms like epoll or kqueue. Plus when it was first written I was thinking that I hadn't written anything with pthreads for a while...

Filtering etc. uses linear searches with linked lists. Isn't that inefficient?

Typically target systems have few network interfaces and filter lists don't extend into hundreds of rules. The lack of fixed overheard associated with hash tables makes linear search more efficient for the common case. I'd throw in a gag about it being distressing if N is over C, but that's the kind of crossover humour that rarely goes down well.

Can I use this to bridge to a tablet using just my laptop?

Not exactly. kplex does not create a wireless access point, it just uses a linux computer's existing wireless interfaces. If you want to create a wireless access point using a linux system, check out hostapd. An Alternative may be to use ad hoc networking to connect to your wireless device (The comar box does this).

# Examples and Tutorials

This is a collection of examples of kplex in use. the first section contains tutorials I have knocked together rapidly so may contain errors. Please [Let me know](http://www.stripydog.com/contact.shtml) if you spot any errors or wish to make suggestions for improvement.

The second section contains links to projects other people have integrated kplex into. These are far more interesting and better written. If you've written up a kplex-based project, please [Let me know](http://www.stripydog.com/contact.shtml) and I will link to it here.

## Tutorials

### [Raspberry Pi NMEA Server](http://www.stripydog.com/kplex/examples/nmeaserver.html)

Tutorial on using your Pi to take NMEA data from serial lines and distribute it via an existing boat network.

### [Raspberry Pi Wireless Access Point](http://www.stripydog.com/kplex/examples/piap.html)

Example of creating a standalone wireless network with a raspberry pi to distribute NMEA data

### [Using kplex to send AIS data to Marine Traffic](http://www.stripydog.com/kplex/examples/marinetraffic.html)

Using a Raspberry Pi (or, indeed, something else)

## Projects using kplex

### [websocketNMEA](http://www.hedmanshome.se/content/view/17/1/)

Erland Hedman's lovely "analogue" glass bridge instrument application (code [here](https://github.com/ehedman/websocketNmea)).

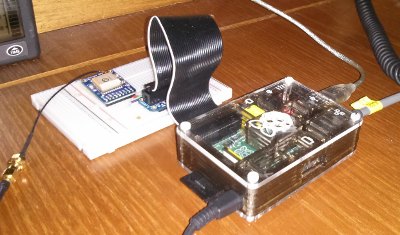
### [Openplotter](http://www.sailoog.com/en/openplotter)

The all-in-one chart plotter (OpenCPN), multiplexor, access point, SDR-AIS and general marine IoT solution for Raspberry Pi.

### [kplex on a MIPS router](http://syspringdawn.blogspot.co.uk/2013/08/disributing-nmea-through-router.html)

RobbieW has compiled kplex for his MIPS-based Asus RT-N16 router running Tomato.

# Examples : Raspberry Pi NMEA Multiplexer



This page describes how to use a [Raspberry Pi](http://en.wikipedia.org/wiki/Raspberry_Pi) and [kplex](http://www.stripydog.com/kplex/kplex/) to act as a multiplexer and network server for data in the NMEA-0183 format. This involves converting between one or more serial line feeds (e.g. from your GPS, AIS or other nav instruments) and an IP (Internet protocol) network. Use of the pi is described but obviously you can generalise this to any computer running Linux, OS X, FreeBSD or probably any POSIX-like OS.

## A note on Terminology

Transporting NMEA-0183 over a computer network is often referred to as "NMEA over Ethernet". This is slightly misleading as ["Ethernet"](http://en.wikipedia.org/wiki/Ethernet), although usually involved, doesn't need to be. There is nothing special in this context, either to the programmer or the user, about ethernet. NMEA data you receive from Internet AIS feeds will frequently travel part of the way to you over other media. The common technology that is being used in the transport of the data here is [Internet Protocol](http://en.wikipedia.org/wiki/Internet_Protocol), so "NMEA over IP" would be a more accurate description of what we are doing here.

## Disclaimer

The last of the wine from last year's trip to Cherbourg was consumed in the writing of this tutorial. Please alert me to the inevitable errors.

## Assumptions

For this tutorial we assume that you have an existing boat network. This may be wired or wireless but for the sake of simplicity we'll assume you are plugging your pi into an ethernet port on your router. The techniques here should work just the same if you connect your pi to a wireless network.

If you don't already have a network, take a look at the [tutorial on building a wireless access point](http://www.stripydog.com/kplex/examples/piap.html).

## Materials

You will need..

* Raspberry pi Model B. The A can be used but would require an additional USB hub. We assume you have added an SD card and installed Raspbian. There are many tutorials on the web as to how to do this
* The most recent version of kplex, installed according to the [instructions](http://www.stripydog.com/kplex/kplex.html).
* 12v power supply with micro USB connector. You can use a car phone charger pluged into a boat's cigarette lighter power socket or there are cheap 12v to 5v power converters on ebay.
* Ethernet cable (or wireless dongle if that is your preferred network connection method)
* Method of connecting your NMEA device (discussed below)

## Connecting Your Device

There are a number of ways you can get NMEA data into the pi. The easiest is if your device outputs NMEA-0183 data over USB. Simply plug the USB into the pi. Some devices have a 9 pin [RS 232](http://en.wikipedia.org/wiki/RS-232) serial connector. If that's the case, your simplest option is to purchase a serial to USB converter. If you have multiple devices to connect, dual and quad port serial to usb adaptors are available. Plug your serial connection into the converter and the converter's USB end into the pi.

If your device just has bare wires, your options depend on what those wires are. Devices following the data transmission standard set out in the first NMEA-0183 specification may have single wires labeled "Rx" (receive) and "Tx" (transmit). There will also be an electrical ground wire. The device sends and receives signals via changes in the potential of a signal wire with respect to ground. Whether it reads an incoming bit of information as a "1" or a "0" depends on the voltage of the "rx" wire relative to ground. With equipment such as this you can wire it to a female DB9 (PC serial) connector which you then connect to the pi via a serial to USB converter, or you can make use of the pi's console port (see below). If wiring a DB9 connector, connect the device ground ("GND") to pin 5 (ground), Tx to pin 2 ("RxD"), Rx to pin 3 ("TxD"). DB9 connectors are available from anywhere that sells electrics professional or hobby electronic supplies and will probably need soldering. Be especially careful to consider which side of the connector you are looking at when working from diagrams as it is easy to be confused between male/female front/back views.

If the device has wires labeled "NMEA OUT +" and "NMEA OUT -" (and possibly "NMEA IN +" and "NMEA IN -", the device treats signal voltages as relative to the "+" and "-" lines and will be using [RS 422](http://en.wikipedia.org/wiki/RS-422) as specified by more recent versions of the NMEA-0183 standards. "Differential" transmission as this is known is less subject to interference over longer wire runs. The most "correct" way to connect such devices to the pi is via an opto-isolated RS 422 connection. It is possible to buy opto-isolated RS422 to USB converters but these tend to be expensive (€80 or more). Non-opto-isolated RS422 to usb converters can obtained far more cheaply if you wish to take the small risk of a direct electrical connection between devices.

It is usually also possible to use an RS232 converter. Many people do this because RS232 to USB converters are cheaper than RS422 to USB converters and more readily available. To connect RS422 output from a device to RS232 input, connect "NMEA OUT +" on the RS422 device to "RxD" on the RS 232 input and connect ground and ground together. This usually (but not always) works because although the amplitude of the signal on the NMEA OUT + line will be half as great relative to ground as it would be relative to "NMEA OUT -", it is still big enough to be recognised by many RS232 receivers. There are, however, no guarantees.

### Using the UART

The pi has a usable onboard UART which is by default designated for use as a console port. The device in the file system associated with this device is /dev/ttyAMA0. There are a number of articles on the web describing how to connect an RS232 device to this UART. A popular method uses a MAX3232 IC and 5 capacitors to convert RS232 to the 3.3v line level signals expected by the pi.

It should also be possible to use an RS422 transceiver and opto isolation, but my enthusiasm for electronics has not stretched sufficiently far to try this.

If you are going to use /dev/ttyAMA0 for serial device input`you need to:

* Remove the reference to it as console device in /etc/cmd.txt
* Disable the terminal process assigned to monitor that device for logins

It is easiest to reboot your device after making these changes. Obviously you will no longer be able to use this port as a console port.

### Connecting Multiple Devices

kplex is a multiplexer so you can connect several NMEA-0183 devices to your pi and mix and match the way you connect them. If you have several several USB connections you may run into a problem where when the pi reboots your USB devices in the file system map to a different physical USB device. "Pinning" logical to physical device mappings is beyond the scope of this document, but if such problems are encountered, the user is encouraged to investigate [udev configuration](http://www.reactivated.net/writing_udev_rules.html).

### Device Permissions

Both the console port (/dev/ttyAMA0) and USB serial devices are not, by default, readable by everybody. If you plan on running kplex as root this is not an issue. If you wish to run kplex as a non-root user, make sure the user is in the "dialout" group. To add user "fred" to the dialout group, edit the /etc/group file and append "fred" to the comma separated list of users in that group. For example, change:

dialout:x:20:

to

dialout:x:20:fred

fred will be be able to access devices which are readable by group "dialout" after logging in subsequent to this change.

## Basic kplex configuration

The [kplex configuration page](http://www.stripydog.com/kplex/configuration.html) contains information on all the things you can do with kplex. There is a lot of information and it can appear quite complex for what is normally a very simple process.

### The kplex configuration file

kplex.conf is divided into sections, one for each "interface" which kplex uses. An interface can either be an input, an output, or for some types of interface, both. Each interface is defined by the "type" of the interface inside square brackets followed by a number of name/value pairs, one pair per line, with name and value speparated by "=". If we have an AIS receiver connected via /dev/ttyUSB0 and a GPS device connected via /dev/ttyUSB1 and we only want to read data from each of them we might define the relevant interfaces thus:

[serial]

filename=/dev/ttyUSB0

baud=38400

direction=in

[serial]

filename=/dev/ttyUSB1

baud=4800

direction=in

The definition of one interface continues until the start of the next (ie until a line with an interface type enclosed in square brackets is encountered). The order of the property/value pairs is not important.

Of course two inputs and no outputs is not particularly interesting. In fact kplex will tell you that and not bother running if you try the above configuration, so let's tell kplex to distribute the information it receives to the network, firstly by creating a tcp server which clients can connect to, and secondly by sending broadcast udp packets. We need to append the following to our configuration file:

[tcp]

mode=server

direction=out

[broadcast]

direction=out

device=eth0

This tells kplex to create a tcp server listening on the default NMEA port (10110) on all network interfaces which outputs data only (it does not receive). It will also broadcast any data it receives out of the ethernet interface "eth0" (change this to "wlan0" to use the first wifi interface). It will use the interface's broadcast address and the default port of 10110. Different ports can be specified using the "port=" property for an interface.

## Connecting to kplex

### TCP Server

An application wanting to connect to the TCP server kplex creates will want to know an address and port to connect to. If you haven't specifically specified a port, kplex uses tcp port 10110. If you haven't specified an address, kplex will listen on all network interfaces of the system it is running on. Note that you cannot specify an arbitrary address: it must be one assigned to the computer kplex is running on. Unless you have a very specific requirement and know what you're doing, you shouldn't tell kplex which address to listen on.

So what address to connect to? If your network is manually configured and you have assigned an address to the pi yourself use the pi's network address. If your pi is in its default configuration and receiving an address from a DHCP server on your network you have some options.

1. Determine what the pi's address is and use that. You can determine the pi's address by logging on and running the ifconfig command. If you are connected to the network by the pi's ethernet interface:
2. ifconfig eth0
3. eth0 Link encap:Ethernet HWaddr b8:27:eb:1c:5e:05
4. inet addr:192.168.1.2 Bcast:192.168.1.255 Mask:255.255.255.0
5. UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
6. RX packets:14007 errors:0 dropped:0 overruns:0 frame:0
7. TX packets:11267 errors:0 dropped:0 overruns:0 carrier:0
8. collisions:0 txqueuelen:1000
9. RX bytes:1737980 (1.6 MiB) TX bytes:4468006 (4.2 MiB)

The first wireless interface (for those connected via wireless) is usually "wlan0".

The address to connect to is the one after "inet addr:" in the example above, in this case 192.168.1.2.

Unfortunately the next time your pi or router reboots it may be given a different address. Logging on each time to find out the address is not very user friendly so we really need a method of giving the pi a fixed address.

1. The best way to do this if your DHCP server (normally implemented on a router) allows it is to tell the DHCP server to always assign a particular address to the pi. Most but not all domestic routers have this capability. Refer to your router's documentation to find out if this is possible. You will need to know the pi's hardware address, which the documentation may variously call "hardware address", "MAC address", "MAC" or "Ethernet address". This is shown in the ifconfig output and is specific to each interface, so "eth0" will have a different hardware address to "wlan0" (if you have a wlan0). In our output shown previously, hardware address is the 6 colon separated pairs of hexadecimal digits after "HWaddr". The IP address you tell the router to assign to that device must be valid for the network to which the pi is connected.
2. If you cannot get your DHCP server to assign the same address to your pi each time it boots you will have to assign an address manually to the pi. First determine the "DHCP range" which your DHCP server allocates dynamic addresses from (refer to your DHCP server's documentation). Choose an address which is valid on your network but outside the DHCP range and modify your /etc/network/interfaces to give the pi a static address, using your router as a gateway (there are many tutorials on this available). If you want to use symbolic addresses with kplex and other applications you will also need to modify the /etc/resolv.conf file to tell the pi's resolver where to find DNS servers.

### UDP Broadcast

For broadcasts it doesn't matter if the system's IP address changes. If you're only going to be using broadcast you can have a different IP address every time and your clients won't care. All you need to know is your network's broadcast address. You can find this out from your router or any device connected to the network (refer to the documentation for the relevant device). In the ifconfig output on the pi above, the broadcast address is shown after "Bcast:". In the above example the network broadcast address is 192.168.1.255. As with the TCP server, kplex will default to using port 10110 if none is explicitly specified.

## Getting things running

### User to run kplex as

kplex needs to be able to read and write data to any files or devices it uses and if you want to use ports below 1024 you need to run as root. There should be no need to use a priviledged network port for kplex and you can ensure that kplex can read devices by changing permissions on them or adding the user kplex runs as to the appropriate group (as discussed above). To run kplex as root is not ideal security practice but is used for the sake of brevity in this tutorial.

Before proceeding any further, check that you have input. When invoked from the command line, kplex uses its configuration file and any other parameters you give it. Assuming we have constructed our kplex.conf as previously described with two serial inputs, a tcp server and a broadcast udp server, to run kplex (as root) and output to the terminal as well as the network:

sudo kplex file:direction=out

If you intend running kplex as a non-root user, run kplex as that user but for now let's assume you are taking the simple option. Run the command as above.

This would use your configuration file and in addition send output to the terminal, which is the default output for the "file" interface type. If you see all the NMEA sentences you expect scrolling down the screen then check to see whether you can connect other programs to kplex over the network. If all is ok we're nearly done. If sentences from one or both of your devices are missing, check physical connectivity and most importantly baud rate. AIS devices are invariably 38400 baud. Standard NMEA devices are normally 4800 (which is kplex's default for serial devices) but some GPS units are 9600 baud. If you have two USB connections at different baud rates, try swapping the device names round to ensure that you haven't mistakenly assumed which physical device is connected to /dev/ttyUSB0 etc.

When you are satisfied that everything is working correctly, stop kplex by holding down the "Ctrl" key and at the same time hitting the "c" key. Now we're ready to set things up for kplex to start at boot time.

### Getting kplex to start at boot time

If you want kplex to be invoked as a non-root user, edit the start script /etc/init.d/kplex and change the variable "RUN\_AS\_USER" to the name of the user you wish to run kplex.

To have kplex start when the pi boots up, run the following:

update-rc.d kplex defaults

You should now be able to start and stop kplex using the "service" command. To start kplex now (without rebooting):

service kplex start

And if you wish to stop it:

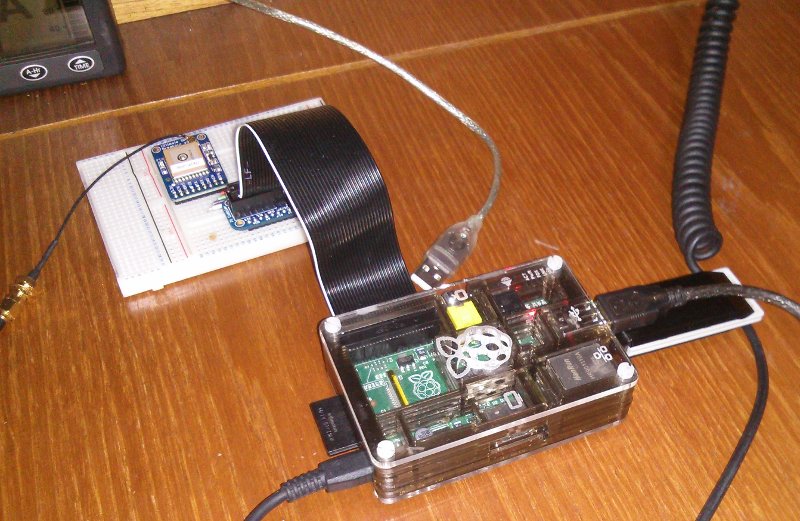
service kplex stop

If in the future you want to undo things such that kplex no longer starts at boot, it can be achieved thus:

update-rc.d kplex remove

see the man page for update-rc.d(8) for full details.

# Example : Raspberry Pi NMEA Wifi Multiplexer



This page describes how to use a [Raspberry Pi](http://en.wikipedia.org/wiki/Raspberry_Pi) to build a wireless access point which also acts as an NMEA-0183 router. I stress that what is described here is not necessarily the ideal way to do it. I've put this page up at the request of others as a starting point. Take some ideas from here and improve on them. If you search for "raspberry pi access point" you will find other ideas.

Some people will be wanting to build an access point which functions purely as an NMEA-0183 to wifi bridge, like some of the commercial offerings. Others will want to combine that functionality with a shared Internet connection. This page will start by describing how to use the pi to build an access point for an isolated network not connected to any other network. Later sections (not yet written) will develop the project, adding hardware (a GPS connected to the pi's console port) and an additional network connection to allow sharing a wifi or 3G connection.

Do be aware that in some cases open source linux drivers are written with incomplete information regarding the closed hardware they need to support. Although others will disagree, my experience with linux wireless over the past decade or so has been that it isn't always consistent or reliable. It goes without saying that this project is not recommended for safety critical systems. I have experienced numerous USB-related lockups when experimenting with the pi as an nmea server and access point.

## Assumed Knowledge

Basic knowledge of Linux including use of a text editor is assumed. There is a wealth of information available elsewhere on getting started with the Raspberry Pi that need not be replicated here. Different types of hardware (specifically, different wifi dongles) may require downloading and installing of firmware in order to get working. Because of the variety of different dongles that might be used it is not possible to give step by step instructions.

## Required Hardware

### Raspberry Pi

I was originally using one of the first generation model Bs with 256MB memory. The memory and processing power are fine for this application but I found the USB bus could be flaky with two wireless adapters connected. This was not a simple power problem as the same issue occurred with a powered hub. Not a problem when only using the pi for an access point but became an issue when also using it as a router to connect to the Internet. Reports from others suggest the B+ improved on this situation and my current Pi v2 is working fine.

### SD card

For the Pi's OS. 4GB is sufficient, but 8 or more gives you room to add more software or store more logged data. Note that the newer Pis use a micro SD card.

### Power supply

You want to convert the boat's 12v power supply to 5v and provide it to the pi via a micro usb. The easiest option is to use a phone charger designed for a lighter socket. I found a converter with a micro usb connector on one side and 1m of power cable on the other. Either solution should be less than €5 on ebay.

### USB wifi dongle which supports master mode

This is the hard part. When acting as an access point, your wifi device operates in what is known as "master mode" (or "AP" mode). Not all devices or their software drivers support this mode. Support for this mode in Linux depends on the device's chipset (ie the electronics inside the device). Unfortunately manufacturers sometimes change the chipset they use in a device without changing the name of their product. You may read in someone's blog post that a particular dongle supports master mode on Linux, go out and buy the "same" thing only to discover you've bought the New! Improved! version with a different chipset which doesn't even work on Linux. The best way to select a dongle is to borrow someone else's, try it, and if it works, don't give it back. Offer to buy them another newer one if you are honest. Alternatively for any device you are thinking about buying, try and find out what chipset it uses then attempt to determine whether that chipset's driver supports master mode under Linux. A summary list of driver capabilities is available on the [Linux Wireless](http://en.wikipedia.org/wiki/Comparison_of_open_source_wireless_drivers#Driver_capabilities) web site, although this is not always up to date and accurate. Worst case, be prepared to give several of your friends wifi dongles as birthday presents this year.

I'm currently using an old D-Link DWA-110 (RaLink RT2571W chipset). It's an 802.11g device which is plenty fast enough for boat data, although may not be your best option if you want to push high definition video about.

UPDATE: For my latest re-work of this project I used an Edimax EW-7811un adapter which is commonly sold for use with the Raspberry Pi. This uses the RTL8188CUS chipset which at time of writing requires a custom version of hostapd. The software can be downloaded from the [realtek website](http://www.realtek.com.tw/downloads/downloadsView.aspx?Langid=1&PNid=21&PFid=48&Level=5&Conn=4&DownTypeID=3&GetDown=false&Downloads=true#2742). I compiled and installed a replacement hostapd, added the declaration:

DAEMON\_SBIN=/usr/local/bin/hostapd

to /etc/default/hostapd in order to reference the custom binary, then replaced:

driver=nl80211

with

driver=rtl871xdrv

ieee80211n=1

wmm\_enabled=1

I will updated this document with more detail at a later date.

### Serial to USB converter(s)

The easiest way to get NMEA-0183 data into your pi is via USB. If the device you want to hook up to the pi outputs NMEA-0183 over USB, happy days. If not you will need a serial to usb adapter. Ideally this should be opto-isolated, but the cost of such adapters tends to be more than replacing the pi several times over in the unlikely event of it being blown up by a power surge. If your NMEA-0183 device outputs data over an RS232 adapter, use an RS232 to serial adapter (~€13 delivered from ebay). If it is compliant with a version of NMEA-0183 prior to v2, use an rs232 to serial adapter. Older equipment and cheap GPS units fall into this category. The device's TX should be connected to the adapter's RX, RX to TX, GND to GND. More modern devices (NMEA-0183 v2 upwards) use RS 422, a differential protocol. You can buy [rs422 to usb adapters](http://www.ftdichip.com/Products/Cables/USBRS422.htm), but these far harder to come by and generally more expensive than your average RS232 to USB adapters. For input only, many people will just use an RS232 to usb adapter, connecting TX+ to RX on the adapter and GND to GND. Tollerances are such that this will generally work although it is not guaranteed.

The pi only has two USB ports. If you use one for the wifi dongle that leaves you with one. In order to connect more than one serial device via usb we can use a dual serial-to-usb converter, or we can connect a usb hub to the pi, and connect other serial-to-usb adapters to that.

Later we'll look at the alternative method of getting data into the pi via the serial i/o pins normally used for the pi's serial console.

### Software

We will be using:

* An OS image. We will use the "standard" hard float Raspbian image from the [Raspberry Pi Foundation](http://www.raspberrypi.org/downloads)
* [hostapd](http://hostap.epitest.fi/hostapd/) for access point management
* [isc dhcpd](http://www.isc.org/software/dhcp), our DHCP server
* [kplex](http://www.stripydog.com/kplex/) for NMEA-0183 multiplexing

hostapd and dhcpd are most easily installed with apt-get once the pi is up and running. Dhcpd is probably not the best choice of dhcp server: It is a full-fat enterprise-ready piece of software that is undoubtedly overkill for your purposes, but it's what I have used for many years and am familiar with.

## Initial Setup

This section may be expanded but for now, follow the [Instructions](http://www.raspberrypi.org/quick-start-guide) on the Pi foundation's web site to get the pi running a basic OS. For the rest of this guide we're going to be working with the command line.

It is critical that you use an OS image which includes up to date firmware. Issues existed with USB handling code which could cause USB hangs when some serial to USB devices were opened. If your firmware is older than April 2013 and you experience such a problem, either use a more up to date image or upgrade the pi's firware. A popular way of doing that is with [rpi-update](https://github.com/Hexxeh/rpi-update).

## Hardware Configuration

For now we're going to assume you are only connecting one NMEA-0183 connection which carries combined AIS and other data at 38400 baud. After the initial setup described above, plug in your wireless dongle to one USB port and plug in the serial to USB converter to the other. If you're going to use several serial to USB converters, you'll need to use a USB hub.

At this point it pays to check that the wireless dongle really is the device you think it is. Here's the output of the "lsusb" command on my pi:

Bus 001 Device 002: ID 0424:9512 Standard Microsystems Corp.

Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub

Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp.

Bus 001 Device 004: ID 07d1:3c07 D-Link System DWA-110 Wireless G Adapter(rev.A1) [Ralink RT2571W]

Bus 001 Device 005: ID 0403:6001 Future Technology Devices International, Ltd FT232 USB-Serial (UART) IC

The wireless dongle is obvious. If we have any questions about it, Internet searches on the chipset ("Ralink RT2571W") or vendor and product id ("07d1:3c07") are often more helpful than searches on the product name: Often several different manufacturers will base their products on the same underlying electronics, and linux may treat them all the same.

Many types of wireless dongle require binary firmware to be loaded on them. Fortunately some of the more common firmwares are already available in the raspbian distribution (in the /lib/firmware directory) and others are available in packages which can be downloaded using normal dpkg tools. If your device doesn't give you a wireless interface, and error messages in your system log indicate a missing firmware file, check if there is a package available before finding instructions on the web for hacking the firmware out of Windows drivers.

## Software Configuration

If you're ultimately going to use your pi on a boat without an Internet connection, set it up beforehand somewhere where you do have an Internet connection.

### Installing Software

Our first task is to install all the additional software we're going to need. If your installation procedure involved a software update you don't need this next step, otherwise update your apt cache with "apt-get update" then update you software with "apt-cache upgrade".

Next install hostapd...

apt-get install hostapd

...and dhcpd...

apt-get install isc-dhcp-server

Now follow the instructions for obtaining and installing kplex [here](http://www.stripydog.com/kplex/kplex.html). Don't worry about configuring it yet.

### Configuring hostapd

My /etc/hostapd/hostapd.conf looks like this:

interface=wlan0

driver=nl80211

ssid=NMEA0183

channel=1

country\_code=UK

ieee80211d=1

hw\_mode=g

auth\_algs=1

wpa=2

wpa\_passphrase=changeme!

wpa\_key\_mgmt=WPA-PSK

wpa\_pairwise=TKIP

rsn\_pairwise=CCMP

Do take some time to read what these settings mean. I've stripped out the comments for brevity but the original prototype (with all comments present) is [here](http://hostap.epitest.fi/gitweb/gitweb.cgi?p=hostap.git;a=blob_plain;f=hostapd/hostapd.conf). You will minimally have to change the SSID setting to some string of 32 characters or less (31 or less in some implementations) which you like, and the passphrase to some string between 8 and 63 characters long. Change the country code according to your location, and if your wifi dongle doesn't use the mac80211 driver, you may need to change the "driver" line.

To check if your device uses mac80211, type

lsmod | grep mac80211

If there is no output, you will need to work out which driver to use, but if you see some output with "mac80211" somewhere in the left hand column, you're good to go. Here's what I see from the above command on my pi:

mac80211 273413 2 rt2x00lib,rt2x00usb

cfg80211 184163 2 mac80211,rt2x00lib

In some distributions you may need to tell startup scripts where to find the configuration file. This can be done by setting the DAEMON\_CONF variable, either in /etc/default/hostapd or the startup script itself. If setting it in the startup script simply append "/etc/hostapd/hostapd.conf" to "DAEMON\_CONF=":

DAEMON\_CONF=/etc/hostapd/hostapd.conf

### dhcpd

Here is a minimal /etc/dhcp/dhcpd.conf for a closed network (i.e. no gateway to the outside world) with no DNS.

ddns-update-style none;

default-lease-time 600;

max-lease-time 7200;

authoritative;

log-facility local7;

subnet 192.168.129.0 netmask 255.255.255.0 {

range 192.168.129.129 192.168.129.191;

}

You can use any of the private network address spaces specified in [RFC 1918](http://tools.ietf.org/html/rfc1918). Here we are using 192.168.129.0/24 and allowing our dhcp server to allocate from a pool of 62 address in that range. Later we'll see how to modify this to enable us to link up with a connection to the outside world.

### kplex

You can do a lot with kplex but for now we are going to do something very simple: Plug in a serial to usb adaptor with the serial end attached to a 38400 bps (such as would be used for a connection carrying AIS and possibly other data) NMEA-0183 feed and create a tcp server which will allow us to connect to it from a wireless application. For the details of what kplex can do (including broadcasting data over UDP) the instructions are [here](http://www.stripydog.com/kplex/configuration.html) but for now here's our basic /etc/kplex.conf, which essentially just requires uncommenting the configuration in the example that was installed with the .deb.

[serial]

filename=/dev/ttyUSB0

direction=both

baud=38400

[tcp]

mode=server

port=10110

direction=both

If your NMEA feed is 4800 baud, change the "baud=" directive accordingly.

### Iniialisation Script

We could start any of the software we've just configured automatically at boot time independently of the other services which start. However we may want to control the "access point" functionality as a unit: hostapd, dhcpd and (when we add it later) BIND. To achieve this, let's creat a trivial script and call it /usr/local/bin/apctl. We'll develop this script further later.

#!/bin/bash

fname=${0##\*/}

usage() {

echo "Usage: $fname start|stop|restart" >&2

exit 1

}

do\_start() {

/etc/init.d/hostapd start

/etc/init.d/isc-dhcp-server start

/etc/init.d/kplex start

}

do\_stop() {

/etc/init.d/kplex stop

/etc/init.d/isc-dhcp-server stop

/etc/init.d/hostapd stop

}

if (( $# == 1)) ; then

cmd=$1

else

if [[ $fname == start\* ]]; then

cmd=start

elif [[ $fname == stop\* ]]; then

cmd=stop

else

usage

fi

fi

case $cmd in

start)

do\_start

;;

stop)

do\_stop

;;

restart)

do\_start

do\_stop

;;

\*)

usage

esac

exit $?

Make it executable:

chmod 755 /usr/local/bin/apctl

In /usr/local/bin create two links to this file:

cd /usr/local/bin

sudo ln apctl startap

sudo ln apctl stopap

Now let's configure /etc/network/interfaces:

auto lo

iface lo inet loopback

allow-hotplug wlan0

iface wlan0 inet static

address 192.168.129.1

netmask 255.255.255.0

post-up /usr/local/bin/startap

pre-down /usr/local/bin/stopap

Here we've assigned the address 192.168.129.1 to our wireless interface. When logging in to the pi over ssh from another computer, this is the address we can go to. More importantly, this is the address we will use as the server for our wireless NMEA-0183 apps to connect to.

### Firewall Rules

For our first cut, isolated from the Internet, we won't install any firewall rules. Out wireless network should be protected by WPA2 with only clients into which you have entered the passphrase should be able to connect. Ignoring firewall rules avoids us having to discuss variations in rules to accommodate different configurations of kplex. We will revist this when we discuss connecting the network to the Internet.

### Disable Unnecessary Services

There are two good reasons to disable services you aren't using.

1. Security: Any running program is a potential risk. Possibly a very small risk, but if you don't need it, why have it?
2. Performance: Any running program you don't need is a waste of resources

### Misc Configuration

If you're not using IPv6, you might as well disable it. If you have a file /etc/modprobe.d/ipv6.conf, ensure the following line is in it:

alias net-pf-10 off

If you don't, or there isn't, make a file of that name containing the line above. Later we'll discuss making proper use of IPv6 on your boat network.

## Firing it up

Everything should start cleanly on boot. You should be able to:

* See the SSID you specified in hostapd.conf in your list of available networks on another wireless device (e.g. ipad)
* Connect to the access point using the passphrase you specified and obtain an IP address
* Connect an application which uses NMEA-0183-over TCP by telling it to use the server at:
* Address: 192.168.129.1
* Port: 10110

## Next Steps

If there is no link associated with the list items below, that page hasn't been written yet and is "Coming Soon..."

* Using the Pi's "Serial Port" for an NMEA-0183 connection
* Adding an Internet connection
* IPv6 connectivity

# Example : Raspberry Pi Marine Traffic server

## Objective

This page describes how to use a [Raspberry Pi](http://en.wikipedia.org/wiki/Raspberry_Pi) and [kplex](http://www.stripydog.com/kplex/kplex/) to send AIS data to [Marine Traffic](http://www.marinetraffic.com/ais/addyourarea.aspx). The instructions here are actually valid for any POSIX-like system. At the time of writing pre-compiled kplex debian packages are only available for ARM (e.g. raspberry pi) and x86\_64 although the source should compile on other versions of Linux, FreeBSD and MacOS X. See the [installation instructions](http://www.stripydog.com/kplex/kplex.html) for full details.

## Pre-requisites

You will need:

* AIS receiving equipment (See the [Marine Traffic instructions](http://www.marinetraffic.com/ais/addyourarea.aspx?level1=150#3) for further details.
* Raspberry Pi installed with the latest Raspbian OS (or other computer running a POSIX-like OS), connected to the Internet
* [kplex](http://www.stripydog.com/kplex/index.html), installed according to the [installation instructions](http://www.stripydog.com/kplex/kplex.html).
* A port number which Marine Traffic will provide you with when you register with them to cover your area

## Putting it together

A more complete tutorial on creation of an NMEA server is given in [this tutorial](http://www.stripydog.com/kplex/examples/nmeaserver.html) which includes information on the options for connecting your AIS receiver to the pi. Briefly though, if the AIS receiver has USB output, plug this into the pi's USB port. If you have a 9 pin serial output, connect that to a serial to USB converter and plug that into the pi's USB port. If you have bare wires only, refer to the tutorial. If you only have one USB serial device plugged into the pi, this will normally be accessible through the logical device "/dev/ttyUSB0".

The connection to marine traffic is outbound from your computer, so you don't need to assign a fixed IP address to the pi. The default Raspbian installation will instruct the pi to obtain an address from a DHCP server.

## Device Permissions

For simplicity, this tutorial explains how to run kplex as root. This is not the optimal configuration in an environment where security is a concern. If running kplex as a non-root user you will need to edit the kplex start script and ensure that the user kplex runs as has access to the AIS serial device as explained [here](http://www.stripydog.com/kplex/examples/nmeaserver.html#user).

## kplex configuration file

All we need to do is customise kplex's configuration file and instruct the OS to start kplex at boot time. If we have our AIS receiver connected at 38400 baud on /dev/ttyUSB0 and we have been assigned port 5321 by marine traffic, the minimal configuration for /etc/kplex.conf looks like this:

[serial]

filename=/dev/ttyUSB0

baud=38400

direction=in

[tcp]

address=5.9.207.224

port=5321

persist=yes

direction=out

Change the "port=" line according to the port which Marine Traffic Assigns you.

Note that this establishes a TCP connection to Marine Traffic rather than a UDP connection as described in tutorials for other software. Marine Traffic supports both types of connection. TCP gives reliable data transmission over a wide area network unlike UDP, but at the expense of a greater overhead. The "persist=yes" directive tells kplex to attempt to re-establish a lost connection, for example if the marine traffic server reboots. If using kplex 1.1 (beta) or later, add the following to the [tcp] stanza to ensure that connections lost because of lost NAT mappings between you and marine traffic are restarted.

keepalive=yes

### Update: Feb 2016

kplex now supports UDP unicast. To configure your connection to Marine Traffic as unicast UDP replace the [tcp] stanza above with:

[udp]

address=5.9.207.224

port=5321

direction=out

coalesce=yes

"coalesce=yes" is optional. If specified kplex attempts to send all parts of a multipart AIS message in a single UDP packet in order to reduce data transmitted.

## Optimisation

kplex supports rate limiting for sentence types but does not support AIS downsampling, in other words it will not retrict the AIS information supplied from particular vessels. If your AIS receiver outputs sentences other than AIVDM/AIVDO you can tell kplex only to send AIVDM (and AIVDO if relevant) to marine traffic. To just send AIVDM sentences to marine traffic add the following to the "[tcp]" section of kplex.conf:

ofilter=+AIVDM:-all

To send only AIVDM and AIVDO:

ofilter=+AIVDM:+AIVDO:-all

## Connecting to other services

kplex can send your data to multiple services and also make it available on your local network. Full configuration details are available in the [kplex configuration guide](http://www.stripydog.com/kplex/configuration.html). To send data to a service similar to marine traffic, simply replicate the section under "[tcp]" in the example above at the end of the file but using the address and port of the other service.

To create a tcp server which is accessible on your local area network and makes AIS data available to client applications such as [OpenCPN](http://opencpn.org/ocpn/), add the following section to the configuration file:

[tcp]

mode=server

port=10110

This makes your data available on tcp port 10110 on all network interfaces of the pi. See the discussion on the address to use to contact the pi in the [NMEA server tutorial](http://www.stripydog.com/kplex/examples/nmeaserver.html#connect) if you didn't assign a fixed IP address to the pi.

If your machine is directly connected to the Internet without a firewall be aware that there is no password or other access control on the tcp server which kplex creates.

To distribute your data via IPv4 UDP broadcast on port 10110, you can append the following declaration to kplex.conf:

[bcast]

device=eth0

port=10110

direction=out

If your pi is connected to the network via a wireless dongle, replace "eth0" with "wlan0" or other name of the device you are using.

Putting it all together, assume we want to send our data to marine traffic as above, sending only AIVDM sentences and also creating a tcp server which we can connect client applications to on our local area network:

[serial]

filename=/dev/ttyUSB0

baud=38400

direction=in

[tcp]

address=5.9.207.224

port=5321

persist=yes

direction=out

ofilter=+AIVDM:-all

[tcp]

mode=server

port=10110

## Enabling kplex to start at boot

As discussed in more detail in the [NMEA server tutorial](http://www.stripydog.com/kplex/examples/nmeaserver.html#boot), to enable kplex to start at boot, run the following command:

sudo update-rc.d kplex defaults

To start kplex without rebooting you can then run:

sudo serice kplex start

bottom border

# GPSD Installation Instructions

<http://catb.org/gpsd/installation.html>

Eric S. Raymond <esr@thyrsus.com>

Here are the steps for installing GPSD and verifying its performance. They assume you have GPSD available as an installable binary package,

Instructions for building GPSD from source (including cross-building) are in the file "build.txt" in the source distribution.

Most of these installation instructions are generic to Linux (inc There are some special notes on installation for OS X and the Raspberry Pi near the end of this file.

## Check that your GPS is live and you can get data from it

Start by making sure you can get data from your GPS, otherwise the later steps will be very frustrating. In this command

stty -F /dev/ttyXXX ispeed 4800 && cat </dev/ttyXXX

replace ttyXXX with the filename of the port. This will probably be either /dev/ttyUSB0 or /dev/ttyS0. If you are on a \*BSD Unix or MacOS X, replace -F with -f.

When you run this command, you should see text lines beginning with $ come to stdout (possibly after a short initial burst of binary garbage). If you don’t see this, you may have OS-level problems with your serial support, but more likely have the wrong device. Look again.

If you have trouble with the preceding step, check your cabling first. Verify that the device is connected and that its power LED (if it has one) is lit.

If you seem to have some sort of serial-device problem, check that your kernel properly supports the device you are using. For GPSes using an RS-232 port (which is no longer common) you will need serial-port support compiled into your kernel. Various USB-to-serial adapter chips found in GPSes require specific drivers.

Under a stock Linux kernel these will all be loaded on demand when the USB system sees the appropriate vendor/product ID combinations. See build.txt for instructions relating to custom kernels.

## Check that your system configuration will allow GPSD to work

Ensure that device permissions will enable gpsd to read from and write to GPS devices even after it drops root privileges. If you are running Fedora Core, Ubuntu, or stock Debian you can skip this step, as the stock configuration has the right properties.

gpsd requires two things: (1) that GPS devices have group read and write enabled, and (2) all of them have the same group ID as a prototypical device, typically /dev/ttyS0 under Linux or /dev/tty00 under \*BSD. It does not actually matter what the owning group is, as gpsd will look this up on startup. Alternatively, (3), you can set a fallback group with the gpsd-group option in case the prototype is not found: this should be the group that has write access to serial devices. On Debian and derivatives including Ubuntu this is "dialout"; on Gentoo/Fedora/openSuse it is "uucp".

Before dropping privileges, gpsd will ensure that it has access to devices given to it on the command line by forcing their group read and write permissions on.

On a Linux with udev, check the files in /etc/udev/permissions.d to ensure that /dev/tty\* devices are all created with the same group and with 0660 permissions.

When gpsd drops privileges, its default is to set uid to nobody and group to the owning group of the prototype device (the configure option gpsd-user=foo will cause gpsd to change to foo instead).

If your system has the Linux hotplug facility installed you can skip the permission-setting part; the hotplug scripts will force the permissions for you. You still have to make sure all the tty devices are in the same group.

## Check your installation prerequisites

A minimum build of GPSD can run pretty close to the metal; all it absolutely needs is the C runtime support. The test clients and various additional features have additional prerequisites:

|  |  |
| --- | --- |
| pthreads library | support for PPS timekeeping on serial GPSes |
| DBUS | gpsd will issue DBUS notifications |
| ncurses | a test client and the GPS monitor depend on this |
| libtinfo5 | shared low-level terminfo library (see below) |
| libusb-1.0.x or later | better USB device discovery |
| Qt + qmake | libQgpsmm depends on this |
| python2.x(x>=6) or 3.y(y>=2) | required for various clients and utilities |
| python-GI bindings | the test clients xgps and xgpsspeed, need this |
| pyserial | ubxtool and zerk need this |

Some ncurses packages contain the terminfo library; some break it out separately as libtinfo5 or libtinfo.

## Installing gpsd

### Install your distributions package(s)

Up-to-date gpsd packages are generally available for Linux distributions including Debian and derivatives (including Ubuntu and Mint), Fedora and derivatives (including CentOS), openSUSE, PCLinuxOS, Mageia, Gentoo, and Slackware. In the embedded space, CeroWRT and Yocto carry GPSD. The GPSD package in the FreeBSD ports tree is also reliably up to date. Even if your distribution is not on this list, it is quite likely GPSD has already been packaged for it.

Whatever distribution you are running, the name of the core GPSD package containing the service daemon is almost certainly "gpsd". However, many distributions break up GPSD into separate installable packages for the core daemon and clients; you should search your repository index for anything with gpsd as a prefix.

### Install from source code

Directions for installing from source are in the file build.txt found in the source distribution.

## How to test the software

1. Start gpsd. You’ll need to give it as an argument a path to a serial or USB port with a GPS attached to it. Your test command should look something like this:

gpsd -D 5 -N -n /dev/ttyUSB0

1. Once gpsd is running, telnet to port 2947. You should see a greeting line that’s a JSON object describing GPSD’s version. Now plug in your GPS (or AIS receiver, or RTCM2 receiver).
2. Type ?WATCH={"enable":true,"json":true}; to start raw and watcher modes. You should see lines beginning with { that are JSON objects representing reports from your GPS; these are reports in GPSD protocol.
3. Start the xgps or cgps client. Calling it with no arguments should do the right thing. You should see a display panel with position/velocity-time information, and a satellite display. The displays won’t look very interesting until the GPS acquires satellite lock.
4. Have patience. If you are cold-starting a new GPS, it may take 15-20 minutes after it gets a skyview for it to download an ephemeris and begin delivering fixes.
5. A FAQ and troubleshooting instructions can be found at the GPSD project site.

## Once you have verified correct operation

1. If you installed from a .deb under Debian or a Debian-derived system, you may need to ‘dpkg-reconfigure -plow gpsd’ to enable the hotplug magic ("Start gpsd automatically").
2. Check out the list of supported hardware at the Hardware page on the GPSD project’s website. If your GPS isn’t on the list, please send us information to add a new line to the table. Directions are included on that page. We can also use updates of the latest version number known to work with hardware already supported.
3. GPSD includes gpsd.php, a PHP script, that you can use to generate a PHP status page for your GPS if you wish. (It may not be in the core package.) It should be manually copied to your HTTP document directory. The first time it’s invoked, it will generate a file called gpsd\_config.inc in that directory containing configuration information; edit to taste.
4. There are other non-essential scripts that may be useful; these are in the contrib/ directory of the source. They may not be available in the packages available from distributions.

For special instructions related to using GPSD for time service, see the GPSD Time Service HOWTO in the distribution or on the web.

## Special Notes for OS X Installation

gpsd will build, install and run on OS X. The easiest way to do so is to first install the MacPorts package. Follow their install procedure at: <http://www.macports.org/install.php>

Then use their port command to install scons, and optionally git if you want to access the development source.

# port install scons

# port install git

While running gpsd, or scons check, you may run out of shared memory segments. If so, you will see this error message:

gpsd:ERROR: shmat failed: Too many open files

By default OS X allows a very small number of shared segments. You can check your allowed maximum number of shared segments, then increase the maximum number, with these commands:

# sysctl kern.sysv.shmseg=8

kern.sysv.shmseg: 32 -> 8

# sysctl -a | fgrep shmseg

kern.sysv.shmseg: 8

# sysctl kern.sysv.shmseg=16

kern.sysv.shmseg: 8 -> 16

# sysctl -a | fgrep shmseg

kern.sysv.shmseg: 16

If you are using a USB based GPS you will likely need the Prolific PL2303 driver. You can find it here: <http://www.prolific.com.tw/US/ShowProduct.aspx?p_id=229&pcid=41>

## Special Notes for Raspberry Pi Installation

gpsd will build, install and run on the Raspberry Pi (RasPi) and Pi 2 using Debian jessie. Other distributions based on Debian (raspbian, etc) will work fine as well. The gpsd package in Debian Wheezy is known to be flaky, be sure to update to a new version of gpsd from source.

### Raspbian

Before compiling gpsd from source, you will need to update your system as root. Switching to the latest raspbian distribution (jessie) is strongly recommended.

# apt-get update

# apt-get dist-upgrade

# rpi-update

# reboot

# apt-get install scons libncurses5-dev python-dev pps-tools

# apt-get install git-core

Git-core is only required to build from a git repository. pps-tools is for testing PPS inputs.

The rest of the installation is just as for any other source based install, as noted in the file **build.txt** .

### Other Debian derivatives (including stock)

#### Jessie

# apt-get install scons libncurses5-dev python-dev pps-tools

# apt-get install git-core

Git-core is only required to build from a git repository. pps-tools is for testing PPS inputs.

The rest of the installation is just as for any other source based install, as noted in the file **build.txt** .

#### Wheezy

Wheezy, being older, requires updating the tools for compiling and testing gpsd:

You need scons at 2.3.0 or higher to build. If your scons is less than 2.3.0 you will need to get a newer scons from wheezy-backport. Partial instructions are detailed here: <http://backports.debian.org/Instructions/>

Basically you need to add this line to /etc/apt/sources.list:

deb http://http.debian.net/debian wheezy-backports main

Then do another update:

apt-get update

Which may lead you to this error if you lack a full set of debian keys:

W: GPG error: http://http.debian.net wheezy-backports Release: The following signatures couldn't be verified because the public key is not available: NO\_PUBKEY 8B48AD6246925553

Partial but detailed instructions to fix that are here:

https://wiki.debian.org/SecureApt

Use either of the following code blocks. The first is more robust:

apt-get install debian-archive-keyring

gpg --keyserver pgpkeys.mit.edu --recv 8B48AD6246925553

gpg -a --export 46925553 | apt-key add -

apt-get update

You can now install scons from the wheezy-backports repository:

apt-get -t wheezy-backports install scons

and other tools:

# apt-get install scons libncurses5-dev python-dev pps-tools

# apt-get install git-core

Git-core is only required to build from a git repository. pps-tools is for testing PPS inputs.

The rest of the installation is just as for any other source based install, as noted in the file **build.txt** .

### Other Raspberry Pi tips

Any USB connected GPS that is known to work with gpsd will work fine on the RasPi. No special instructions apply.

A very popular option is to install the AdaFruit Ultimate GPS HAT. With this GPS you also get a good 1PPS signal. This works as any other GPS with gpsd, but there are two things to note. The GPS takes over the serial console: /dev/ttyAMA0. The PPS signal will be on GPIO Pin #4.

Only three specific changes need to be made to make the HAT work. First in the file /boot/cmdline.txt, remove this part "console=ttyAMA0,115200 kgdboc=ttyAMA0,115200)". That frees the serial port from console use so the GPS can use it.

Second you need to tell the boot process to load the pps\_gpio module and attach /dev/pps0 to GPIO pin 4. Do that by adding this line to the bottom of /boot/config.txt: dtoverlay=pps-gpio,gpiopin=4

Reboot so those changes take effect.

Run gpsd like this:

~ # gpsd -D 5 -N -n /dev/ttyAMA0 /dev/pps0

If you are on the RasPi with gpsd version 3.17, or above, /dev/pps0 can be autodetected. and used for PPS if available.

gpsd 3.17 and up only:

~ # gpsd -D 5 -N -n /dev/ttyAMA0

You can verify gpsd is using the PPS by running ntpshmmon:

~ # ntpshmmon

# Name Seen@ Clock Real L Prec

sample NTP0 1461619703.641899335 1461619703.445224418 1461619703.000000000 0 -1

sample NTP2 1461619703.642203397 1461619702.999262204 1461619703.000000000 0 -20

sample NTP0 1461619704.142097363 1461619703.445224418 1461619703.000000000 0 -1

sample NTP2 1461619704.142204134 1461619703.999258157 1461619704.000000000 0 -20

If you do not see NTP2 then you misconfigured the pps\_gpio driver.

The serial time is provided to ntpd on NTP0, the PPS time is on NTP2, not on NTP1 like described earlier. So your ntp.conf will need to be adjusted from:

# GPS PPS reference (NTP1)

server 127.127.28.1 prefer

fudge 127.127.28.1 refid PPS

To:

# GPS PPS reference (NTP2)

server 127.127.28.2 prefer

fudge 127.127.28.2 refid PPS

Now proceed as for any other operating system to use gpsd.

Warning, the pps\_gpio driver in all linux kernels up to the current 3.19 only reports one edge. Be sure to validate that your PPS signal is not offset by the pulse width.

Detailed instructions are available from their web site: <https://learn.adafruit.com/adafruit-ultimate-gps-hat-for-raspberry-pi/>

You will need to dig deeper to make the PPS work, here is a good reference: <http://www.satsignal.eu/ntp/Raspberry-Pi-NTP.html>

Last updated 2018-10-22 07:17:14 EDT