MWPTools

Tools for MultiWii NAV on free OS

Overview

The MultiWii flight controller is currently (early 2014) undergoing some exciting development; namely the introduction of way point navigation. Sadly, this impressive development has somewhat transformed the previously cross platform (albeit ugly Java) MultiWii experience, into effectively, a proprietary (Microsoft) operating system experience.

This project attempts to address this by providing tools for MultiWii-NAV mission planning on free operating systems. The principal application, **mwp** is a mission planner. Whilst it is unlikely that **mwp** can approach the functionality of WinGui[1], the aim is to at least enable users on free platforms to:

- Plan a mission using a geographic display;
- Edit missions, way point and actions;
- Upload the mission to a capable MultiWii-NAV flight controller;
- Download a mission from a capable MultiWii-NAV flight controller;
- Load and save missions to a file, in a format compatible with WinGui;
- Track the mission in real time where suitable telemetry devices are available;
- Mission logging and log conversion to at least KML and GPX.

Other tools in the **mwptools** suite include:

- **pidedit** : A simple PID editor;
- **mspsim** : A simulator for MSP (MultiWii Serial Protocol). **mspsim** provides enough of the MSP to facilitate the development of applications such as **mwp** and **pidedit**.

Caveat

There is not, to the best of the author's knowledge, any protocol documentation for the changes to MSP to support MultiWii-NAV; the applications in this suite have been developed from knowledge gleaned from postings in the MultiWii forum, GPS NAV thread <http://www.multiwii.com/forum/viewtopic.php?f=8&t=3989>, and by reverse engineering of WinGui source code. Neither is an entirely satisfactory basis for development, compared to a proper specification, but it's the best available to date.

In addition to the lack of documentation, the author has yet to fly a GPS capable multi-rotor (However, I do have a GPS and GPS capable FC for bench testing). **NO WARRANTY. You have been warned**.

mwp – "A mission planner for the rest of us"

mwp is the mission planner component of **mwptools**. It provides a graphic user interface for creating, editing and managing MultiWii NAV missions.

1 WinGui is an open source MultiWii management program on Microsoft proprietary operating systems by András Schäffer (EOSBandi). It may be downloaded from https://code.google.com/p/mw-wingui/.

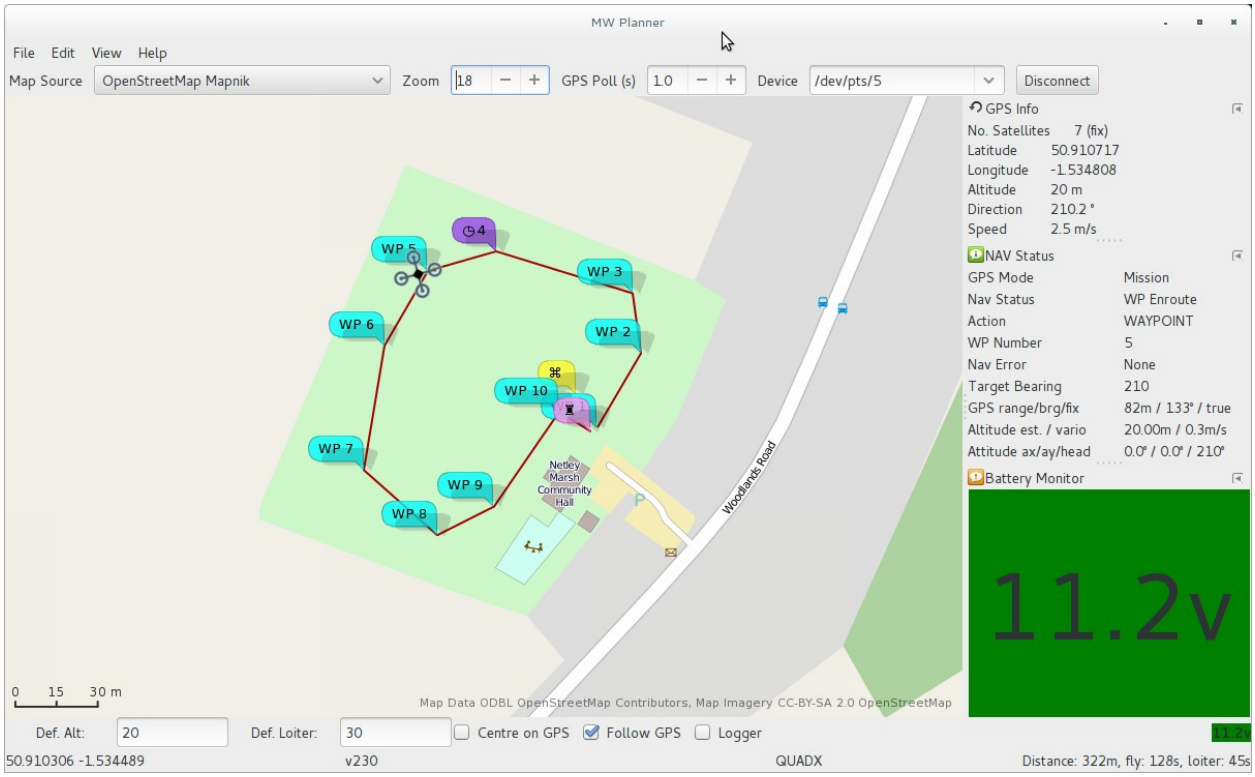


Illustration 1: **mwp** window showing a mission and mission tracking

Illustration 1 Shows the **mwp** main window and the main user interface elements.

- A menu bar. The menu options are described below;
- A setting bar, allowing the selection of the map source, zoom level, GPS update and the serial device used to connect to the flight controller. In this example, a pseudo-terminal is used with **mspsim** (the **mwptools** MSP simulator) providing the GPS and navigation status;
- The main body of the application window it divided into a number of panes:
  - The map pane shows the map view and the current mission. If GPS following is enabled, the aerial vehicle is also shown;
  - A dock area. This can display a variety of information, include the mission (in tabular form), the current NAV status, the current GPS data and the battery status. In the screen shot, the mission tote is hidden, but can be enabled from the View menu. The dock is further described in the section "Using the Dock". Note that many of the screen shots following were taken before the dock was implemented;
- A status bar shows the current map centre position, the MultiWii version, the MultiWii vehicle type and mission statistics (distance, flight time and loiter times). Jumps are taken into account when calculating these values, unless the jumps have infinite repeats.

Existing missions can be loaded from the File / Open menu item. A standard file chooser, with filters of 'Mission' (*.xml, *.mission) will load existing mission files, including those created with WinGui. Where the mission was previously saved by **mwp**, the mission file will include map centre and zoom information, so loading a mission will zoom to the mission location.

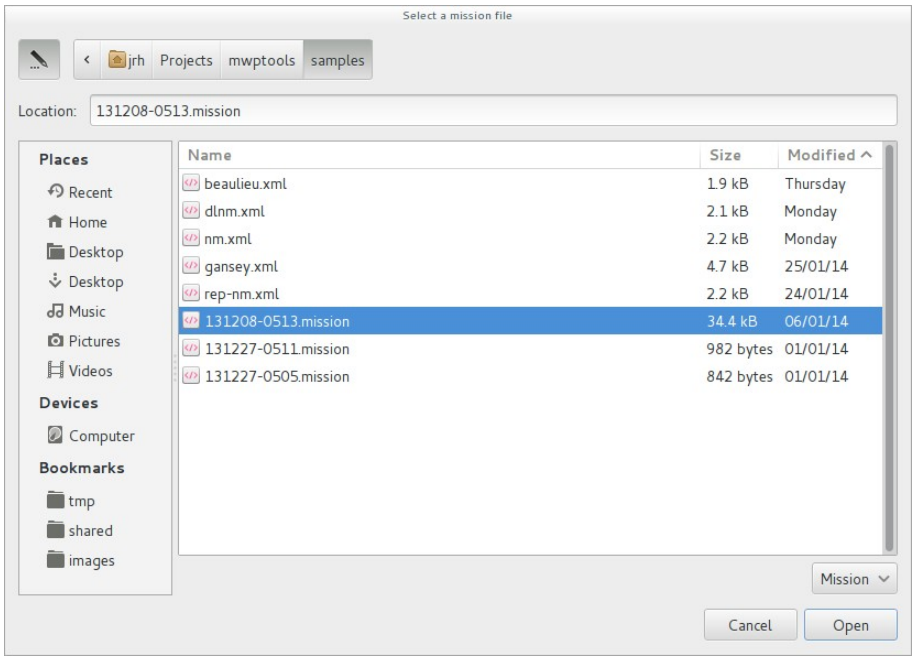


Illustration 2: Opening a mission file (EOSBandi / WinGui sample file)

Opening the selected file loads the mission and displays the mission data on the map, and in the tote, replacing any previous mission data (note this example WinGui mission would not load on a real FC, due to the excessive number of way points):

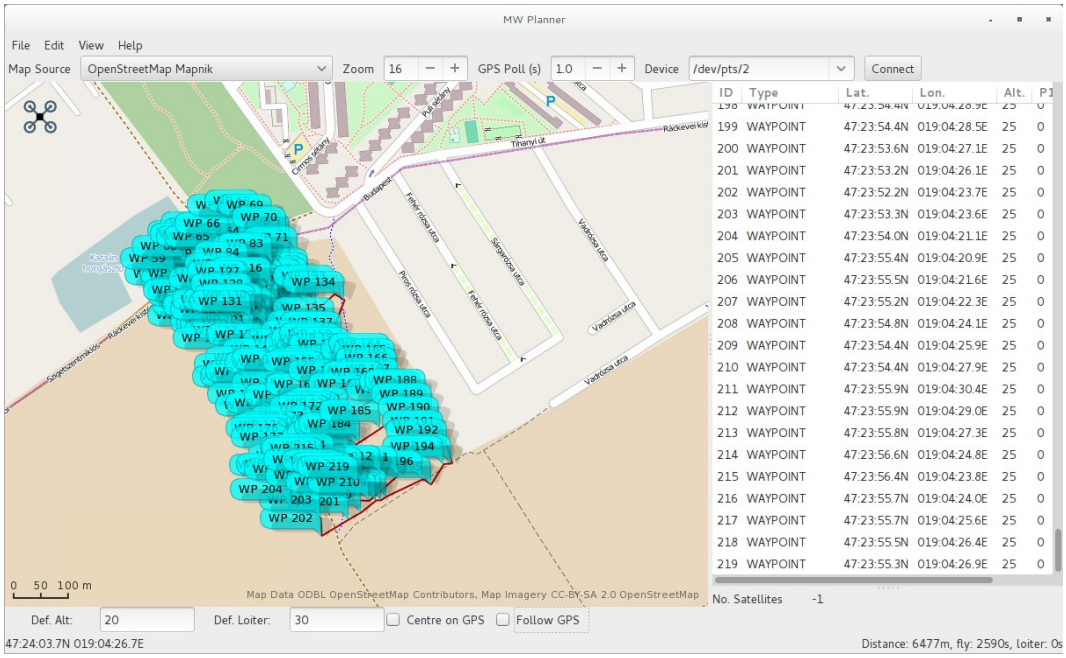


Illustration 3: Import of EOSBandi / WinGui sample file (don't fly this one)

New way points and actions may be created in two ways:

1. Right mouse click on the map;
2. From the right mouse button pop-up menu in the mission tote.

- A defaults and settings bar. Here the default altitude and loiter times used for way point definition are shown / edited. There are also display options to follow the GPS, and centre on the GPS. The current battery voltage is also shown, colour-coded according to the battery status

setting from the FC;

Opening Missions

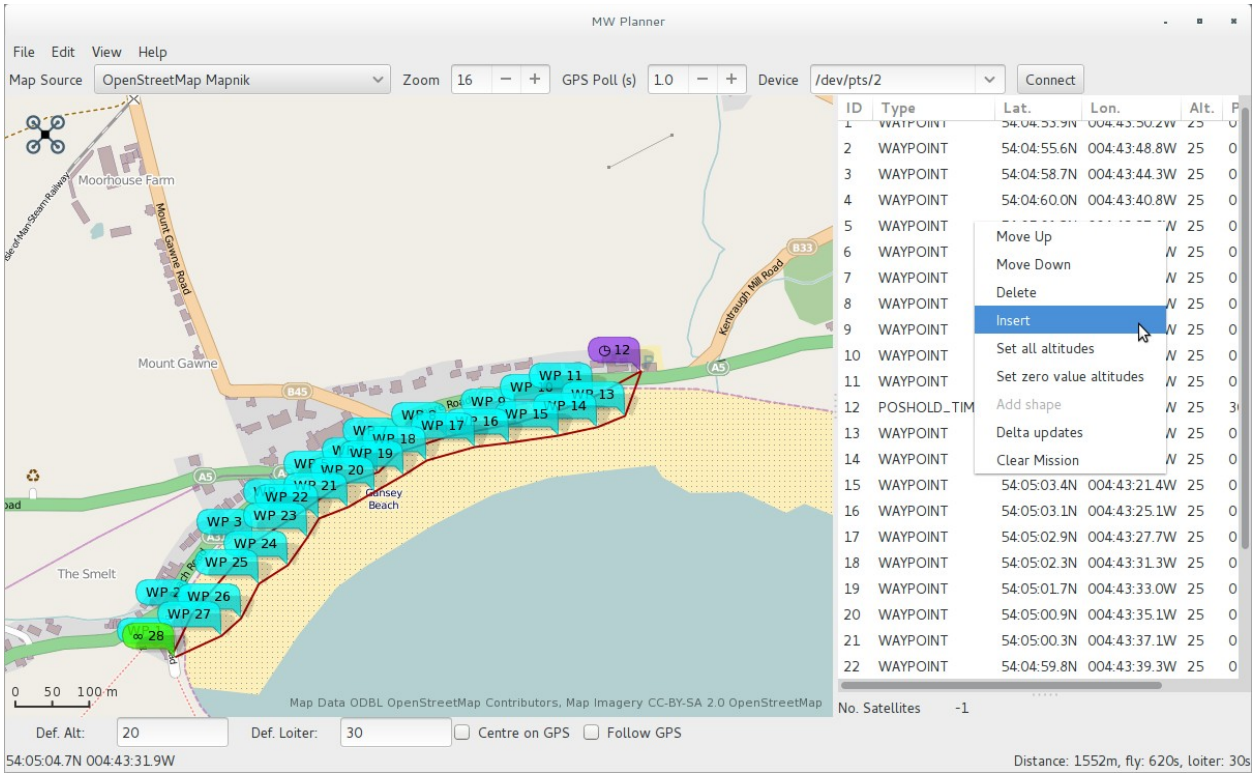


Illustration 4: Mission pop-up menu (let's go for a [quick] pint at "The Shore" hotel)

In Illustration 4, the mission pop-up menu has been invoked, and the “Insert” action selected.

Creating and editing missions

Way points and actions may be created for new missions, or modified in existing (loaded) missions.

Mission points are created either by a right click on the map, or by the mission pop-up menu, “Insert” action (see Illustration 4). Depending on how the point is created, it will be displayed in a slightly different way. This distinction is somewhat artificial, but seemed logical at the time --- it may be changed in the future. This is shown in Illustration 5, and explained further below.

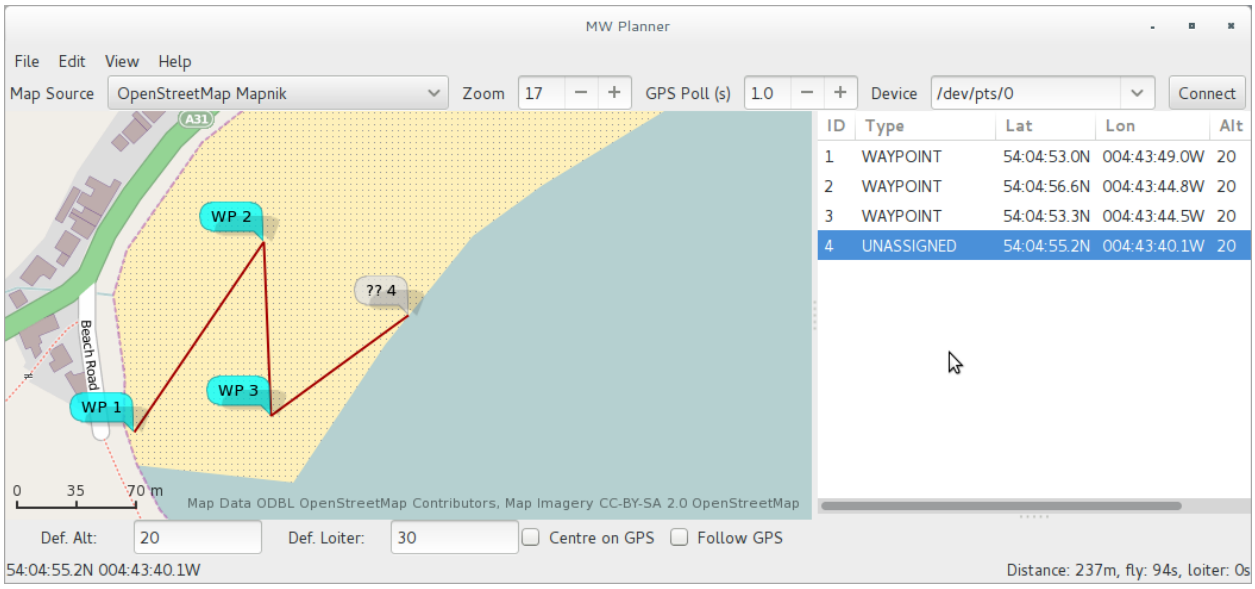


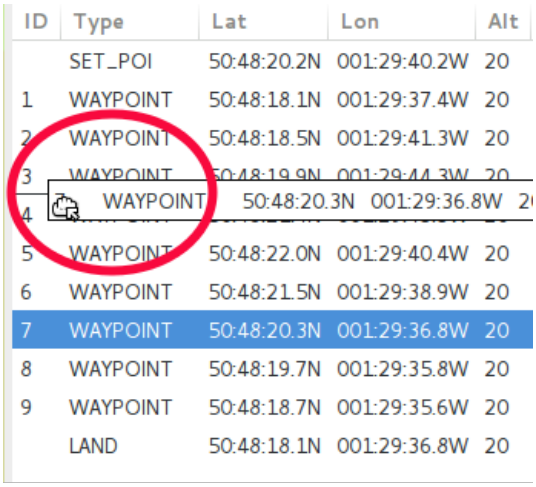
Illustration 5: Entering way points in **mwp**

In Illustration 5, way points 1,2, and 3 have been entered by right click on the map (as this is the most common way point type, this is the default), way point 4 (??4) has been entered from the mission pop-up menu, and thus has type of UNASSIGNED. UNASSIGNED way points are *not* saved or uploaded as part of a mission; they must be converted to a 'real' mission way point or action (see Type below).

Editing way points

Way points are edited in the tote. When a row is selected, the tote column headers will change to indicate the data fields appropriate to the point type (in particular the "parameters" P1,P2,P3 those interpretation is dependent on the point type.

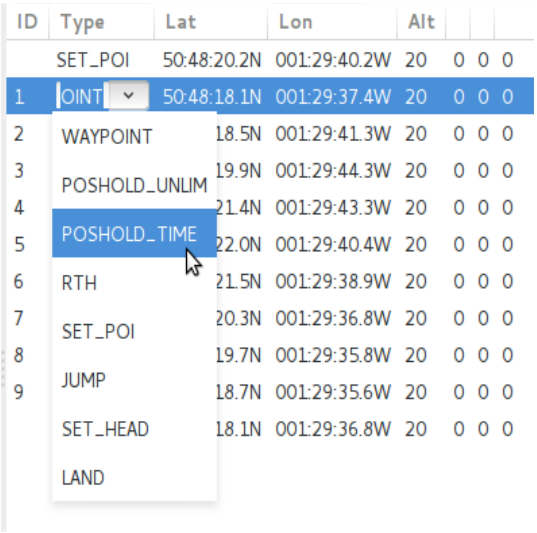
- Position. The position of a way point may be changed by dragging the way point icon on the map. If the display mode is decimal degrees (see Preferences and Settings), then the position may also be edited in the tote. Where the position is DMS.s, then the position cannot currently be edited in the tote;
- Order. The order of way points may be changed by either:
  - Using the "Move Up" and "Move Down" entries from the mission pop-up menu; or
  - Dragging the tote item to the desired position. In order to drag, the entry must be 'grabbed' on the ID column (Illustration 6). In that screen-shot (below), way point 7 is being dropped between way points 3 and 4.
  - At the end of the drop, the list and markers on the map will be re-ordered.



ID	Type	Lat	Lon	Alt
	SET_POI	50:48:20.2N	001:29:40.2W	20
1	WAYPOINT	50:48:18.1N	001:29:37.4W	20
2	WAYPOINT	50:48:18.5N	001:29:41.3W	20
3	WAYPOINT	50:48:19.9N	001:29:44.3W	20
4	WAYPOINT	50:48:20.3N	001:29:36.8W	2(
5	WAYPOINT	50:48:22.0N	001:29:40.4W	20
6	WAYPOINT	50:48:21.5N	001:29:38.9W	20
7	WAYPOINT	50:48:20.3N	001:29:36.8W	20
8	WAYPOINT	50:48:19.7N	001:29:35.8W	20
9	WAYPOINT	50:48:18.7N	001:29:35.6W	20
	LAND	50:48:18.1N	001:29:36.8W	20

Illustration 6: Dragging a way point to re-order.
Note the 'grab' by the ID column

- Type. The way point type may be selected from a drop down menu embedded in the “Type“ column of the tote:



ID	Type	Lat	Lon	Alt			
	SET_POI	50:48:20.2N	001:29:40.2W	20	0	0	0
1	OINT	50:48:18.1N	001:29:37.4W	20	0	0	0
2		18.5N	001:29:41.3W	20	0	0	0
3		19.9N	001:29:44.3W	20	0	0	0
4		21.4N	001:29:43.3W	20	0	0	0
5		22.0N	001:29:40.4W	20	0	0	0
6		21.5N	001:29:38.9W	20	0	0	0
7		20.3N	001:29:36.8W	20	0	0	0
8		19.7N	001:29:35.8W	20	0	0	0
9		18.7N	001:29:35.6W	20	0	0	0
		18.1N	001:29:36.8W	20	0	0	0

WAYPOINT
POSHOLD_UNLIM
POSHOLD_TIME
RTH
SET_POI
JUMP
SET_HEAD
LAND

Illustration 7: Changing a way point type

Once the type has been changed, default parameters for that way point type or action will be set. The defaults are discussed in the section “Preferences and Settings”, and may be edited in the tote.

- Altitude. New points are created with the default altitude (from the “defaults and settings bar“, and as pre-defined in preferences). Some basic validation is performed;
- Parameters P1, P2 and P3. The parameters P1,P2 and P3 are integer values that have a meaning specific to the way-point type or action. For example, for action type of JUMP, P1 is the point to which to jump, and P2 is the number of repeats. The author's understanding of the parameters associated with each way point type or action is given in the reverse engineered protocol documentation (e&oe);
- Multiple points may be added using the pop-up “Add shapes” option (see below).

Other mission popup menu actions

- Delete. The delete action will delete the selected (highlighted) way point. If no way point is

selected, this option has no affect;

- Set all altitudes. Sets all way point (but not LAND) altitude values to the current “Def Alt” value;
- Set zero value altitudes. As for “Set all altitudes”, but only affects points with an altitude of zero;
- Add shape. Where the mission consists of *exactly one* SET POI point, this option will display a dialogue to enter the number of points in a shape, the radial distance (from the SET POI to a point), an offset angle and the direction of rotation:

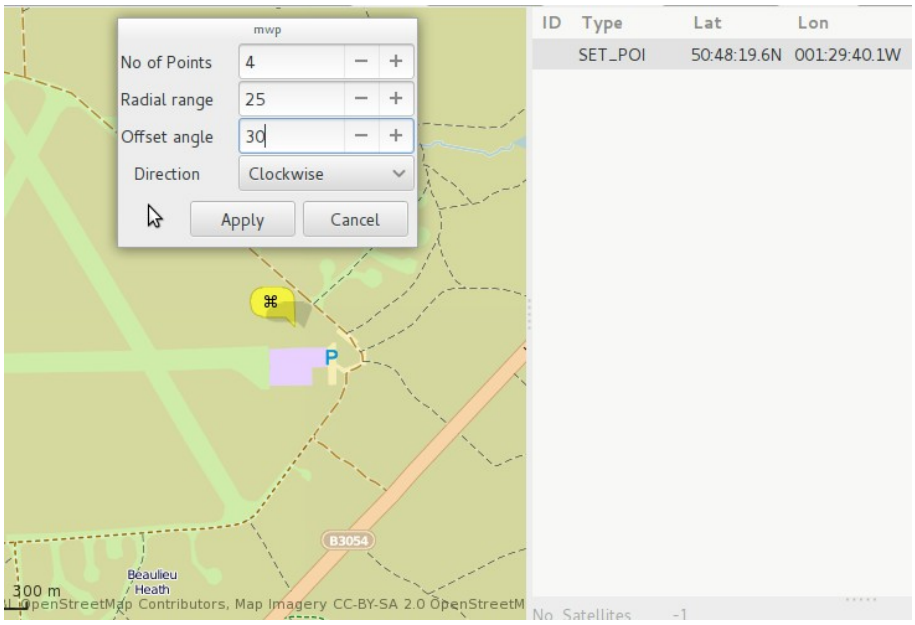


Illustration 8: Defining a shape around a single POI

Apply, with the result:

Illustration 9: Inserted shape points

Noting that points 1 and 5 (start and end) are coincident, If you didn't expect to see this, consider what happens with no offset:

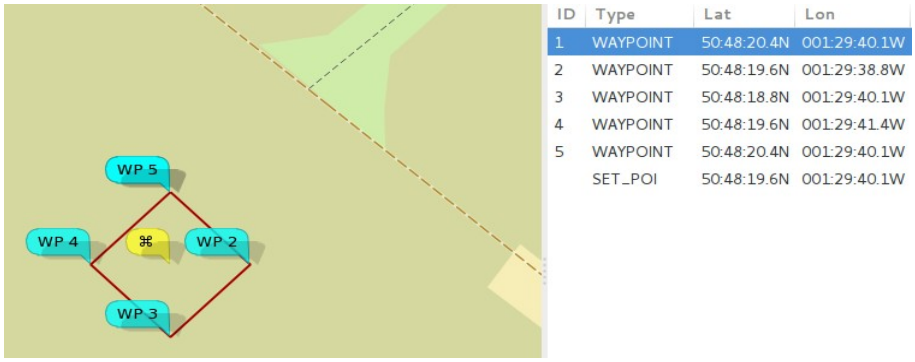


Illustration 10: Square with no offset

Here we start with the first (and last) points immediately North of the POI; thus, in Illustration 9 we have the same shape rotated by 30°, as specified. If you wanted the lines to be horizontal / vertical, specify an offset of 45° for a square.

- Delta updates. This option allows you to make bulk (delta, positive or negative) changes to all positions and / or altitudes:

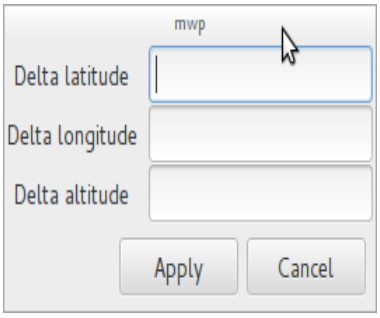


Illustration 11: Delta changes dialogue

- Clear Mission. The Clear Mission option clears the mission. There is no confirmation and no reversion, so be sure you want to do this.

Preferences and Settings

Preferences and default settings are stored using GTK+3 gsettings / dconf (on Linux / other free Unix like platforms, and whatever OS mechanism this maps onto for other operating systems). The gsettings schema is org.mwptools.planner. The following settings are stored.

Key	Usage	Default	Prefs
device-names	A list of serial devices that will be presented in the Connect combo box.	'/dev/ttyACM0', '/dev/ttyUSB42', '/dev/rfcomm42'	Yes
default-latitude	The default latitude for the map	50.909528	Yes
default-longitude	The default longitude for the map	-1.532936	Yes
default-loiter	The default loiter time for POSWAIT_TIME way points	30 (seconds)	Yes
default-altitude	The default altitude for way points	20 (metres)	Yes
default-nav-speed	The default navigation speed (used for timings only, does not affect FC settings)	2.5 (m/s)	Yes

Key	Usage	Default	Prefs
default-map	The default map to be used. If not defined, the default from the underlying toolkit (libchamplain, OSM Mapnik) will be used.	(none)	Yes
default-zoom	Default zoom level	15	Yes
display-dms	Display positions as degree:minute:seconds (vice decimal degrees).	False	Yes
compat-version	The WinGui version stored in mission files. Note that no validation is done dependent on the compat-version value, as any WinGui version specific variations are not known.	2.3-pre8	No
show-scary-warning	Defines whether a scary warning concerning the dangers of flying an undocumented protocol is displayed when you upload way points to the flight controller.	True	No
map-sources	User defined map sources. You can add additional map sources beyond the default sources in libchamplain. The value is the name of file defining such sources. This is further explained in the Map Sources section of this document.	(none)	No

Table 1: Settings used by mwp

The preferences dialogue is invoked from the menu Edit/Preferences item, and appears as Illustration 12:

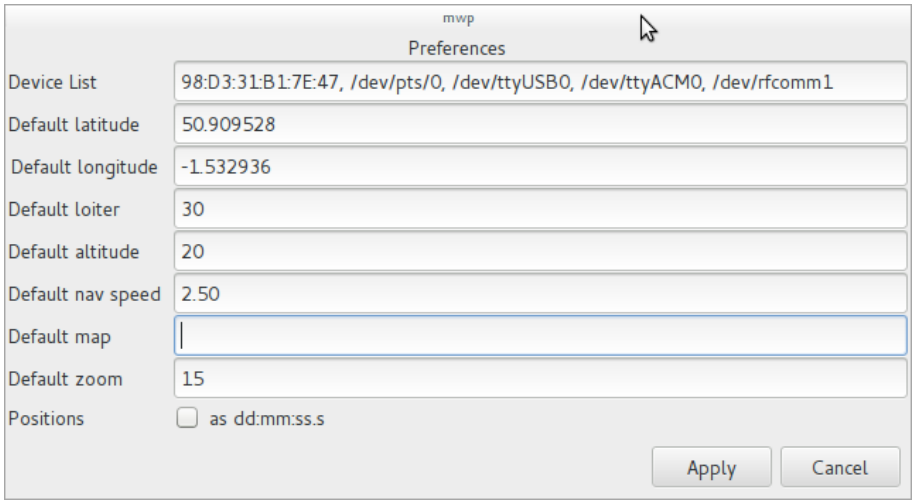


Illustration 12: **mwp** preferences dialogue

Note that preferences in general do not take effect until the application is restarted (other than position format, which will be used the next time positions are updated). This may be rationalised in a future version.

For Linux, if you wish to use a bluetooth serial device, the you can enter the device name as the bluetooth address; **mwp** will then use a rfcomm socket, rather than you having to create a `/dev/rfcomm` port (but you can create and use a `/dev/rfcommX` port if you so wish). On OS other than Linux, a device name is necessary to use bluetooth. In the screen-shot, the first device is a HC-06 BT serial device, which is also

mapped to `/dev/rcomm1` (using an address / BT socket will result in slightly faster device opening compared to a bound serial device).

Other menu options

- File / Save. Saves the current mission to an XML fiile.
- File / Upload. Uploads the current mission to the flight controller (where connected and the FC supports sufficient capability). Once uploaded, the mission is then downloaded and compared to the source mission. You are warned if there is a mismatch, and the state of the FC is undefined if a mismatch occurs;
- File / Download. Downloads the mission from the flight controller, replacing (without warning, you selected to do this), the mission in the application;
- File / Quit. Quits the application (without warning .... you get the modus operandi here);
- View / Nav Config. Where connected to a suitably capable flight controller, displays a nav config window (static data from MultiWii config.h);

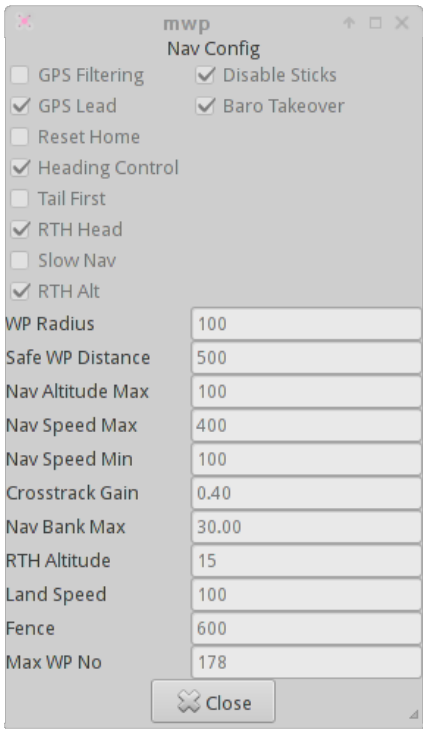


Illustration 13: Nav Config dialogue (static information)

- View / Mission Tote. Forces display of the mission tote in a dockable window;
- View / Nav Status. Where connected to a suitably capable flight controller, displays a nav status dock window;
- View / GPS Status. Forces display of the GPS status a dockable window;
- View / Voltage in dock. Forces display of the battery voltage (where monitored) in a dockable window;

- Help / About. Displays a somewhat uninformative 'about' dialogue (however, my wife likes pink, and I like MW quad-copters).

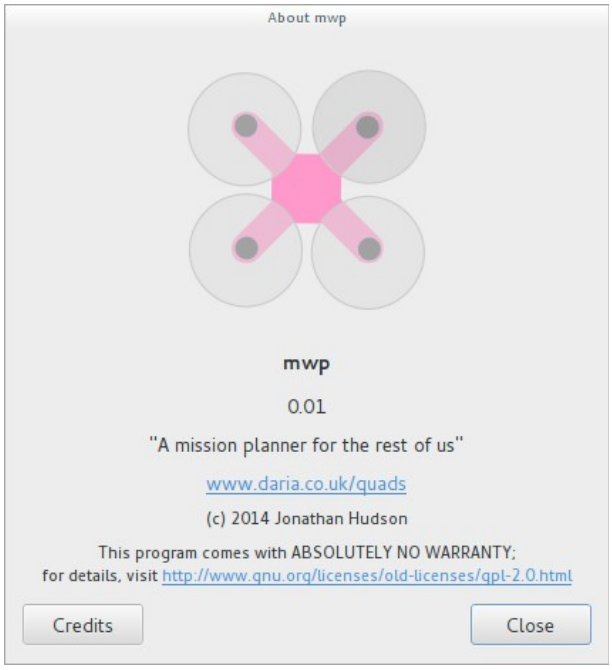


Illustration 14: Uninformative "About" dialogue

Logging

The screen-shots so far have been taken prior to the addition of the logging capability. As of v0.01, **mwp** supports logging. Logging is started / stopped by the Logger checkbox:

Illustration 15: Logging option

When logging is invoked, a log file is created in the current working directory:

```
mwp_YYYY-MM-DD_HHmmSS.log
```

where YYYY is the year, MM month, DD day, HH hour, mm minutes SS seconds in the local time zone.

The file format is “streamed json”, i.e. a json document for each logged event, for example (in this example, the vehicle is sitting on a window sill in my house, at the time there was no voltage monitoring on the vehicle, nor any mission loaded):

```
{"type":"status","utime":1391973539,"gps_mode":0,"nav_mode":0,"action":0,
 "wp_number":0,"nav_error":10,"target_bearing":0}
{"type":"comp_gps","utime":1391973539,"bearing":0,"range":0,"update":1}
{"type":"altitude","utime":1391973539,"estalt":1.03,"vario":0}
{"type":"attitude","utime":1391973539,"angx":-0.4,"angy":-1.3,"heading":273}
{"type":"analog","utime":1391973539,"voltage":0,"power":0,"rssi":0,"amps":0}
{"type":"raw_gps","utime":1391973539,"lat":50.9150417,"lon":-1.5304901,
 "cse":21.1,"spd":0.13,"alt":21,"fix":1,"numsat":7}
```

For each entry, there is a type field describing the data that follows, and “Unix” timestamp (utime, time_t), followed by type specific data.

The script (in the samples) **mwp-log2gpx.rb** is a Ruby language script that converts the log to a GPX file. **mwp-log2kml.rb** converts the log file to KML.

Using the Dock

mwp uses the GNOME Docking Library (gdl) to provide a dock capability. Items in the dock may be hidden, iconified or torn off into a separate window (that may then be returned to the dock). The section explains how use gdl in **mwp**.

When **mwp** is started, the dock contains the mission tote and two iconified items (the GPS Status and Nav Status windows [highlighted in red]).

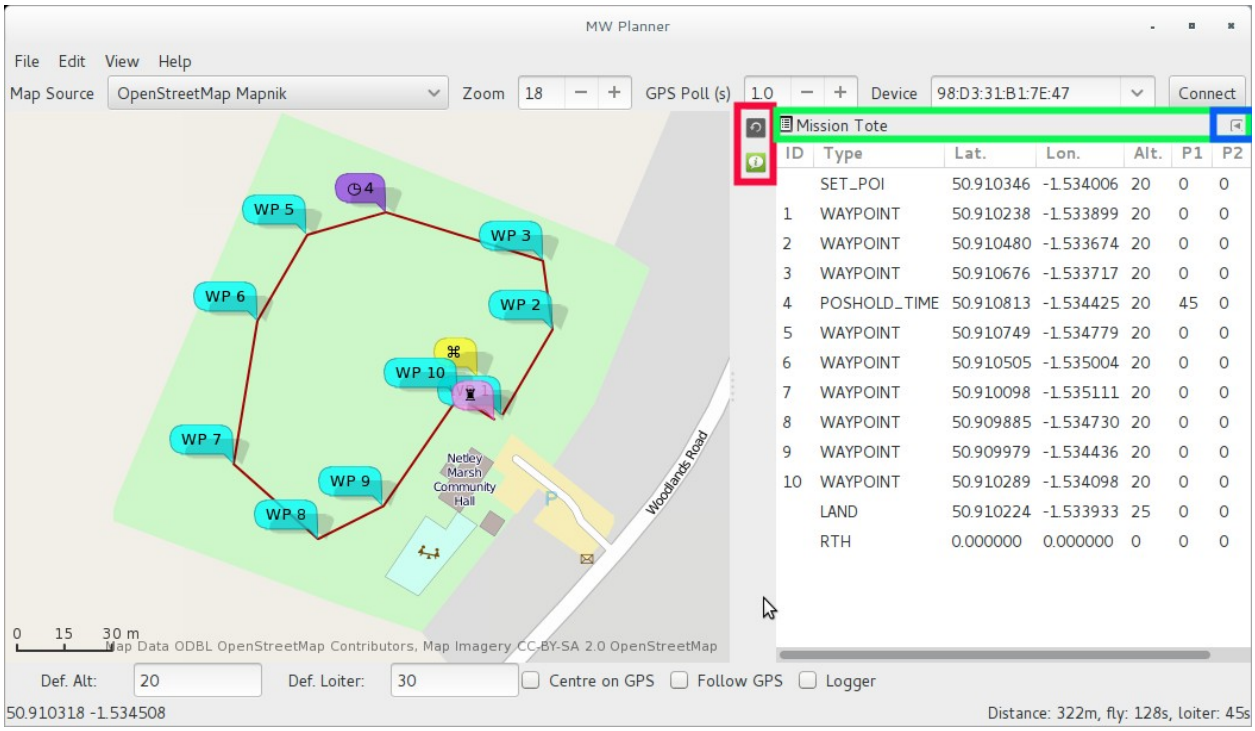


Illustration 16: Inital Dock display

In Illustration 16, three areas of the dock are highlighted:

- In red, the dock icons. Clicking on these will restore the window (either to the dock, or as a separate window);
- In green, the dock tab bar. Where multiple items are in the dock, the tab icon may be dragged to reposition the docked window. In also has a pop-up menu, that allows the item to be completely hidden (but recoverable from the View menu), and;
- In blue, a iconify widget that will add the item to dock icon bar (the bit in red).

If the item tab bar icon (left-most in the green area) is dragged from the dock, the item will appear as a separate window.

By default, the battery monitor does not appear in the dock (but may be added from the View menu). In Illustration 17, the battery monitor has been added to the dock, and the mission tote has been moved to the dock icon bar.

The battery monitor window will resize the text to provide as visible as possible a voltage window. As for the all the dock windows, this window may be dragged on the desktop (and resized). In figure Illustration 18, the battery monitor window has been dragged from the dock and may be resized as an ordinary window. The detached window may be added to the dock by dragging the window's "tab bar" back into the dock, or added back to the dock icon bar using the iconify button (the left facing arrow to the right of

the window's “tab bar”. If the detached window is closed, then it becomes hidden, and may be reattached to the dock (as an iconifed dock item) from the View menu.

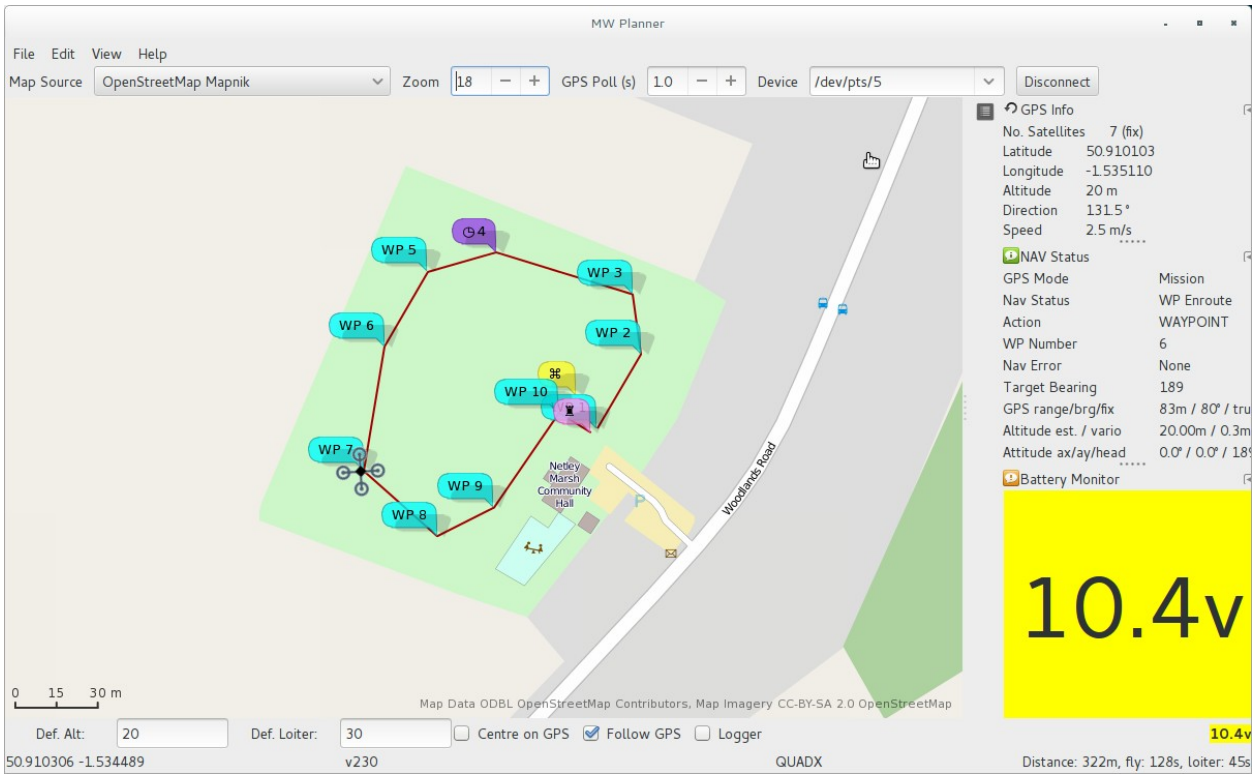


Illustration 17: Dock with battery monitor

Illustration 18: Battery window detached from the dock

Map Sources

As may have been mentioned in passing, **mwp** uses libchamplain for the map display. libchamplain, by default provides a set of free, unencumbered, open source map resources. The libchamplain developers take the terms and conditions of commercial mapping sources very (some might say, over) seriously and no commercial sources (Google, Bing etc.) are, by default, included, for fear of the user violating their somewhat ambiguous terms of service.

There are, however, other map sources that can be legitimately added to libchamplain (and hence **mwp**). A mechanism is therefore provided to add such sources. By default, this will not permit you to abuse the terms of service of commercial providers, as only 'fixed URI' is ever supported. There is a 'quadkeys' patch for libchamplain floating around that enables Bing maps to be supported, apply at your own risk; this will not support other sources that require rewritten URIs based on scale (such as Ovi, Yahoo, Yandex etc).

To use an additional map source in **mwp**:

- Define the "map-sources" key in the settings, this is the name of a JSON file in `~/.config/mwp` defining the map sources (or rather XDG_CONFIG_HOME on Unix like systems, and perhaps something to do with CSIDL_LOCAL_APPDATA on Microsoft OS).

A JSON file, `samples/sources.json` provides an example:

```
{
  "sources": [
    {
      "id": "historic-gb",
      "name": "GB Historic",
      "license": "(c) National Library of Scotland",
      "license_uri": "http://maps.nls.uk/projects/api/index.html",
      "min_zoom": 5,
      "max_zoom": 16,
      "tile_size": 256,
      "projection": "MERCATOR",
      "uri_format": "http://geo.nls.uk/mapdata2/os/seventh/#Z#/#X#/#TMSY#.png"
    },
    {
      "id": "localcache",
      "name": "Private cache",
      "license": "(c) OSM",
      "license_uri": "http://localhost/",
      "min_zoom": 0,
      "max_zoom": 17,
      "tile_size": 256,
      "projection": "MERCATOR",
      "uri_format":
"http://localhost/mapcache/tms/1.0.0/nsites@GoogleMapsCompatible/#Z#/#X#/#TMSY#.png"
    }
  ]
}
```

The latter only works (probably) chez-author. See the libchamplain documentation for map sources definitions (this documentation really exists).

With the quadkeys patch, you can add the Bing source (just bing for it ...).

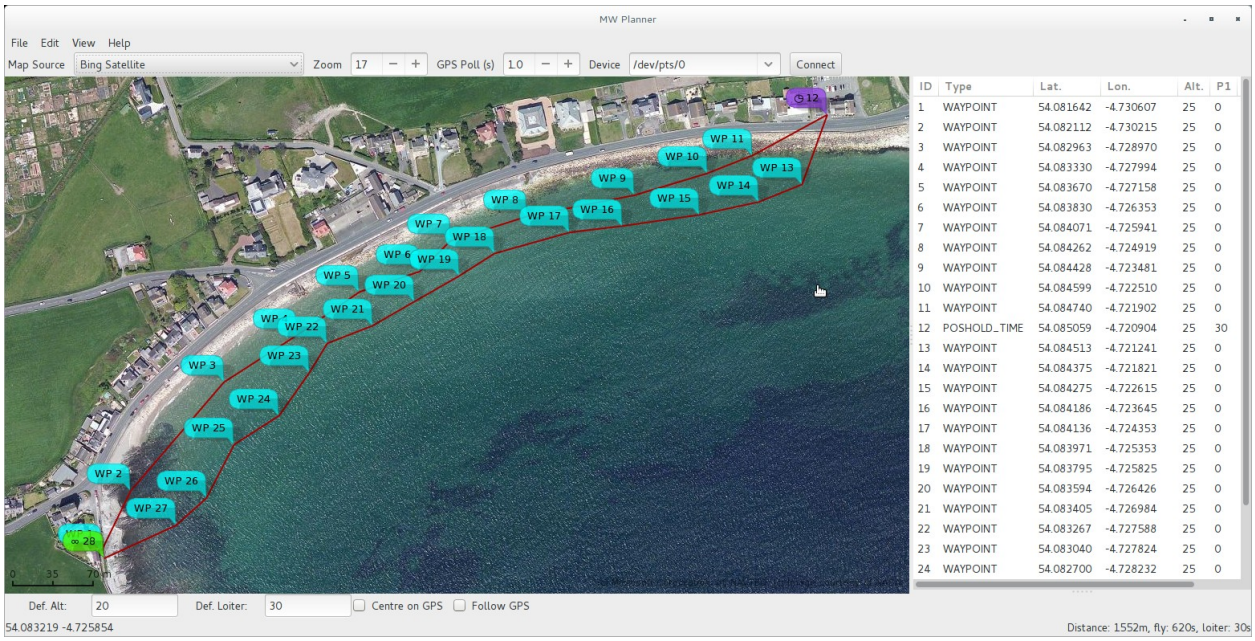


Illustration 19: Don't fly this when the tide is in (Bing maps)

Flash backs to the 1950s work too (the map data, not the Xfce UI, Arch Linux on a Samsung ARM Chromebook):

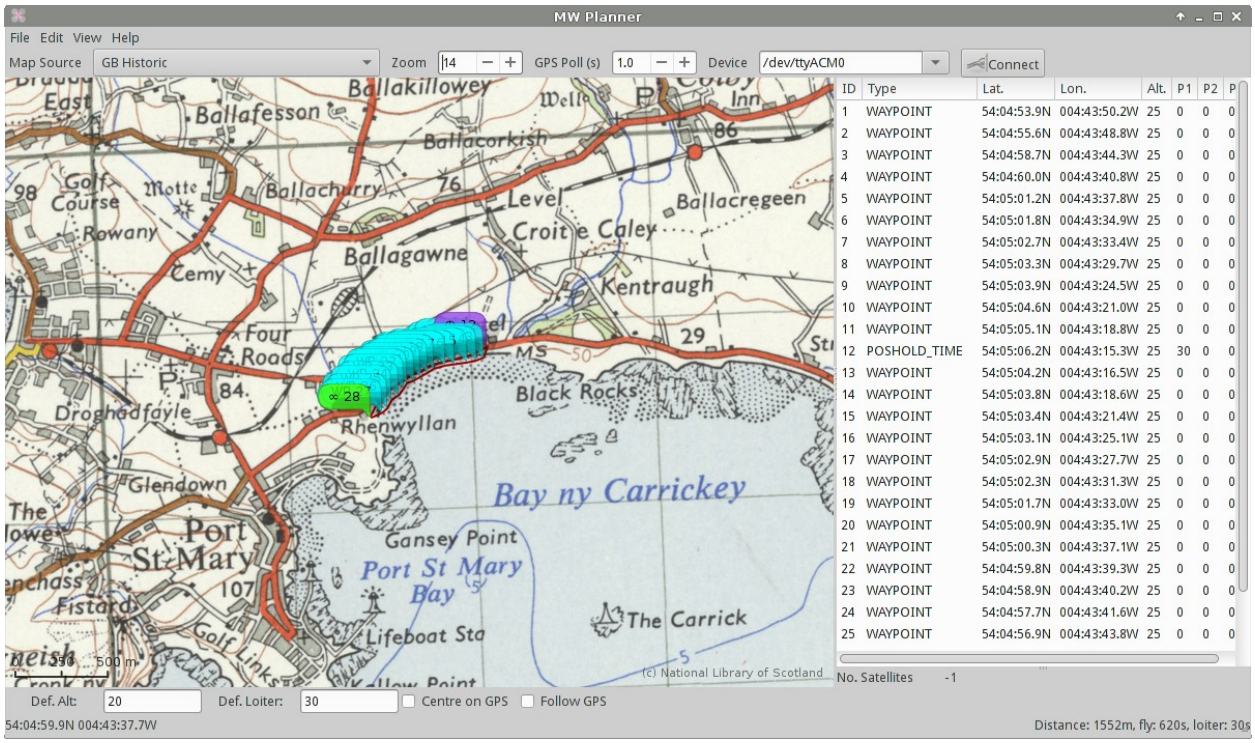


Illustration 20: "You can take the boy out of the island, but you can't take the island out of the boy"

Installation

The **mwptools** applications have a number of installation targets:

- make install-system: Requires root. Installs under `/usr`;
- make install-local: Installs under $HOME, XDG directories. Applications in `$HOME/bin`. Directories

are created as needed.

Default for make install is install-local.

Note that for the .desktop files to be found for install-local, it is necessary that `$HOME/bin` is in the path when the desktop environment is started; note also that `~/.profile` is sourced by the desktop environment. It is probably also necessary for XDG_DATA_DIRS to contain `$HOME/.local/share`.

For non-free / non-Unix like OS, you will have to work out your own installation, as the author is not familiar with such systems.

Building

In order to build **mwp** (and **mwptools**) on any platform a number of infrastructure open source components are required:

- vala (0.20 or later recommended, may work with earlier), and gcc (or clang);
- GTK+-3.0 (and normal dependencies, 3.8 or later tested, 3.10+ preferred);
- libchamplain (0.12 or later);
- libchamplain dependencies (Clutter mainly);
- gdl (GNOME Docking library);
- libbluetooth (Linux only).

For free OS, these dependencies should be easily satisfied; for propriety platforms, Clutter, gdl and libchampain may (or not, I don't know) be available. The code to open / close the serial port is C (rather than vala), specifically to work on Microsoft Windows, however the author has no means to check that the conditional code even compiles on Microsoft OS, let alone runs. For Ubuntu, you will the "-dev" versions of the libraries.

Specifically on Ubuntu, the version of libchamplain (as of Ubuntu 13.10) does not support bounding boxes, you should use the mwpu target, rather than **mwp**.

mwptools is developed and tested on Arch Linux (ARM, x86_64, ia32); it is also tested on Ubuntu.

In particular, for Ubuntu, and platforms using versions of libchamplain c. 0.12.3, rotation of the platfom is disabled, as this causes the platform to be incorrectly geo-located on the map. Libchamplain 0.12.6 (at least) have no such issues.

Protocol Documentation

In order to attempt to understand the new NAV protocol, a reverse engineered understanding is published as https://docs.google.com/document/d/16ZfS_qwc-rJeA7N5Tx0DA6wtgxl6HdGgaz-jE3lHBWs/edit?usp=sharing. I hope this is useful. Corrections appreciated.

pidedit

pidedit is a PID editor. Unlike **mwp**, this is based on well documented MSP protocols and tested / flown against real MultiWii FCs.

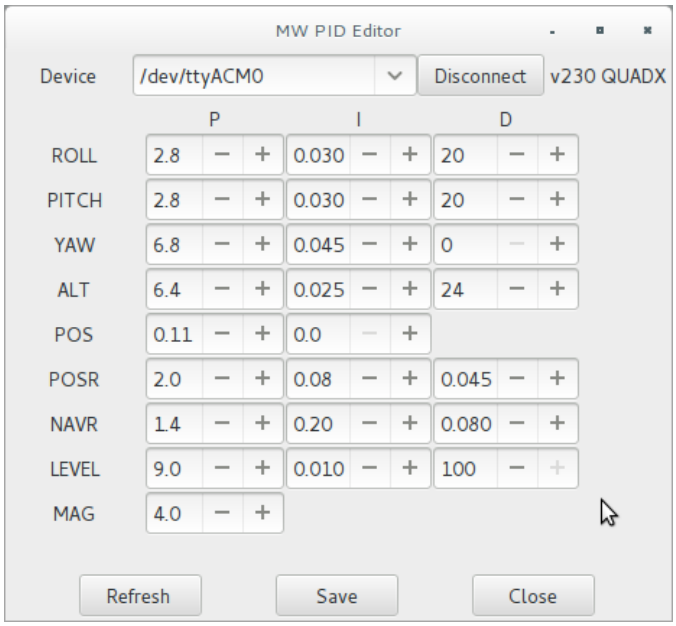


Illustration 21: **pidedit** - trivial PID editor

The application should be somewhat self-explanatory:

- Each PID parameter can be edited;
- Refresh: Refreshes the user interface from the flight controller;
- Save: Saves the values to the FC EEPROM;
- Close: Quits the application.

As for **mwp**, there is a gsettings entry for **pidedit**, the schema is “org.mwptools.pidedit”, and the sole item is "device-names”, a comma separated list, as for **mwp**.

mspsim – MSP simulator

mspsim is a simple simulator for the MSP protocol (including the undocumented NAV extensions). Its sole purpose has been to allow the author to develop **mwp** without a NAV capable MultiWii platform. Unlike the other tools in the **mwptools** suite, it is unashamedly POSIX dependent, using POSIX pseudo-terminals for communications (it would however be simple to use e.g.com2com on Microsoft OS).

The concept is simple. On start-up **mspsim** allocates a master pseudo-terminal (pty) for its use and displays the slave pty device name that can be used in the application under test (e.g. **mwp** or **pidedit**).

In Illustration 22, **mspsim** is being used to test **mwp**. **mspsim** is invoked with the same mission file as is loaded into **mwp**. The slave pty advertised by **mspsim** is used as the serial device by **mwp. mspsim** responds to MSP commands issued by **mwp**, so the simulated FC configuration, status, GPS location and nav status is played into **mwp**. When the “Play” button is **mspsim** is activated, **mspsim** “flys” the mission, providing simulated GPS and “nav status” to **mwp**.

mspsim is a very simple simulation; it does not honour POSHOLD_TIME loiter time, nor does it honour JUMP; its sole purpose is to simulate the MSP communications. Thus a number of states are hard coded into the simulation (e.g. vehicle type is always QUADX).

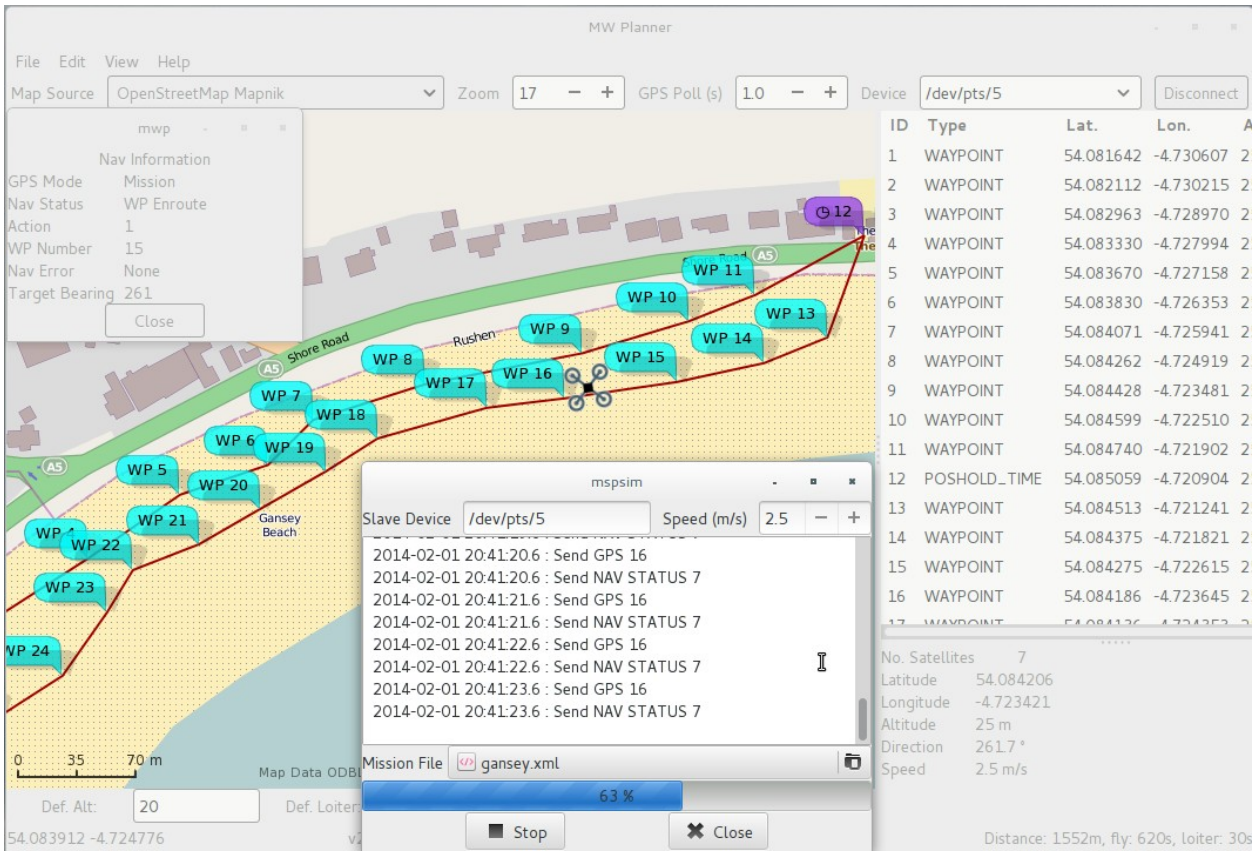


Illustration 22: **mspsim** being used to test **mwp** (pre-dock UI)

The numbers after the command mnemonics in the status list represent the (assumed) message payload size.

The following messages are supported (to some degree) by **mspsim**:

- The Connect / Disconnect buttons connect / disconnect the serial device.;

MSP Messages simulated

Mnemonic	Description, usage
IDENT	Version identity (230).
STATUS	FC status. **mspsim** supplies sensor (0x1f) and cycle time (2345).
MISC	FC settings. **mspsim** supplies minthrottle, maxthrottle, mincommand.
PID	PID settings. **mspsim** supplies values appropriate to an extremely lightweight, somewhat over-powered 250mm MTM QUADX.
ALTITUDE	Returns appropriate data from the simulated mission.
EEPROM_WRITE	Recognised, no response necessary.
SET_PID	Recognised, response sent. **mspsim** does not retain value supplied.
RAW_GPS	**mspsim** supplies values based on the mission file supplied (or 1 satellite, no fix if no file).
SET_HEAD	Recognised, no response necessary.
WP	**mspsim** supplies any values previously received via SET_WP.
SET_WP	Values are stored, and will be returned via a WP query.

Mnemonic	Description, usage
NAV_STATUS	**mspsim** supplies values based on the mission file supplied and the state of "Play" through the mission file.
NAV_CONFIG	Recognised, with a default filled response.
RADIO	Recognised, with a zero filled response (a fault, but dependent on availability of protocol documentation).
ATTITUDE	Recognised, response based on simulated mission.
COMP_GPS	Returns appropriate data from the simulated mission.

It must be reiterated that the purpose of **mspsim** is not to supply a realistic flight simulator, rather it is to provide responses to test the extended MSP protocol.

Mission File extensions

mwp writes a single extension to the WinGui XML mission file format, representing the zoom level and centre coordinates appropriate to the mission, for example:

```
  <mwp zoom="18" cx="-1.5347760915756226" cy="50.910293849231742"></mwp>
```

These extensions should be ignored by WinGui.

Command line options

mwp and **mspsim** may be supplied with the name of a mission file. If one is using **mspsim** to test **mwp**, it is advisable to use the same file.

```
$ mwp --help
Usage:
  mwp [OPTION…]

Help Options:
  -h, --help              Show help options

Application Options:
  -m, --mission           Mission file
  -s, --serial-device     Serial device
  -c, --connect           connect to first device
  --ignore-sizing         ignore minimum size constraint
  --force-rotation        Force vehicle icon rotation on old libchamplain
$ # thusly ...
$ mspsim ../samples/gansey.xml &
$ # see which slave PTY we're offered ... in this case /dev/pts/5
$ mwp -m ../samples/gansey.xml -s /dev/pts/5 -c
```

Platform Caveats

The communications processing uses packed structures, and are read and written using the appropriate data size (int16, int32). Due to the way the MSP structures are defined, this means that the programs are performing 'unaligned' read and write. On CPUs such as ia32, x86_64, ARM and PPC this causes no problem. In the unlikely event of trying to run these tools on a strictly aligned architecture such as SPARC, it may not work (and the application will crash with SIGBUS). Patches accepted.

Other Configuration Items

It is not currently my intention to address all the elements of MultiWii configuration. The cross-platform tools MultiWiiConf or Baseflight may be used for configuration of items not addressed by **mwptools**.

Author, copyright and licence

mwptools is written by Jonathan Hudson <jh+mwptools@daria.co.uk> (aka stronnag on the MW forum, but the NSA already knew that). It is licensed under the GPL 2.0 (or later, your choice).

This document describes the 0.01 pre-alpha software version.