U4B – Ultimate4 Balloon tracker Operating manual

Firmware version 1.01_002

Important information

This is the operating manual for the U4B and should be read in conjunction with the relevant hardware manual. On the U4B page http://qrp-labs.com/u4b there are other helpful documents and video links.

This operating manual contains a comprehensive description of everything in the firmware. It might be a little overwhelming. However, remember that U4B is both extremely simple, extremely complex, and everywhere in between – whatever you want it to be.

For the simplest possible flight, a basic tracking application program is already installed. The following steps will get your U4B tracker ready and tracking set up:

1. Connect a micro-USB cable between U4B and your PC; run a terminal emulator on the PC (see section 2 of this manual) and connect to U4B

2. Enter your callsign in the U4B configuration screen (see section 3.1 of this manual) and make a note of the channel number. Channel number can also be set to any channel in the range 0 to 599.

3. Set up your flight tracking in the QRP Labs shop account (see section 5 of this manual), which will give you a tracking map page at http://qrp-labs.com/tracking – note that there are tracking website which provide better visualizations.

4. Assemble your system hardware (refer to the hardware manual) – consisting of U4B, GPS antenna (twisted #28 wire), HF (e.g. 20m) antenna, solar cells and small LiPo battery

5. Test thoroughly on the ground, before snapping off the USB connector tab along the line of holes (refer to hardware manual)

6. Remember: SAFETY first (refer to hardware manual), then select your launch site, launch day (good weather!), inflate your balloons (refer to documentation at http://qrp-labs.com/u4b) and fly!

Contents

1.   Introduction

The U4B tracker was developed over a period of 7 years from 2015 to 2022, in collaboration with Dave VE3KCL who has launched 83 test flights from Toronto, Canada. With flight duration from 2 hours to 305 days (10 months, almost 17 laps around planet Earth), they all taught us something and were great fun.

U4B is designed to be an easy to use, lightweight, low-cost module that can be configured as simply as entering your callsign, yet for more advanced owners can be flexibly extended with more sensors and as much complexity as you like. Automated tracking maps and utilities are available on the QRP Labs website, both for simple tracking purposes and downloading your own telemetry. The U4B PCB contains:

- 33.0 x 12.7mm PCB (plus removable protrusion with micro-USB connector)
- Weight: 1.8g (with micro-USB protrusion removed)
- 32-bit ARM microcontroller running 64-bit QDOS (**Q**RP Labs **D**isk **O**perating **S**ystem)
- 128K disk (implemented on EEPROM chip)
- 27mW (approximately) transmitter using Si5351A synthesizer
- TCXO referenced frequency stability
- Band coverage 2200m to 2m
- LM75 temperature sensor
- Status LED
- USB interface for configuration, programming and easy firmware update (just copy the new firmware file into the apparent USB Flash drive).

Simplest possible operation:

Just connect to U4B with a PC terminal emulator, and configure it with your callsign. Register the flight name, details and channel on the QRP Labs (or other) tracking website. Fly!

More flexible and advanced features:

U4B contains a wealth of flexibility and hardware expansion options which you can use to customize your flight:

- 19 GPIO pins – of which 9 can be configured as analog inputs and 8 are easily accessible via PCB edge pads; all 19 can be used as digital input or output control pins
- I2C bus for connecting additional sensors e.g. pressure, humidity
- BASIC programming language with full-screen text editor, compiler and debugger
- 128K Disk storage for your programs and data; BASIC can read/write data files
- Command line utility
- Telemetry over WSPR for relaying your additional sensor data

The U4B radio transmitter can transmit the following modes:

- QRP Labs tracking and telemetry over WSPR
- WSPR (including extended mode and slow 15-minute WSPR)
- JT9 (1, 2, 5, 10, 30 minutes)
- JT65 (modes A, B, C)
- Hellshreiber (standard, DX, and slow multi-tone FSK)
- CW (standard speed, QRSS, FSKCW and DFCW)
- Customized "Glyph" patterns can produce a unique idenfier on QRSS

# 2.	Terminal connection

The U4B tracker contains a Virtual COM Serial port USB device, for accessing QDOS (QRP Labs Disk Operating System) via a terminal emulator program running on your PC. This connection is used for initial configuration, and all BASIC programming, development and testing that you may wish to do, if you are pursuing more advanced flight goals. A common USB-A to USB-micro cable is required. Any OS may be used (Windows, Linux, Mac etc).

## 2.1	Drivers and PC setup

No additional drivers are required for operation with most Linux distributions, Apple Mac, MS Windows 10 or 11.

For older versions of MS Windows, it may be necessary to install a driver for the serial port because this driver is not on your computer already by default. This driver is available from the ST Semiconductor website at https://www.st.com/en/development-tools/stsw-stm32102.html and is applicable to 98SE, 2000, XP, Vista®, 7, and 8.x Operating Systems. There is a description for installation on Windows 7/8 on the QRP Labs QLG2 page http://qrp-labs.com/qlg2 so if in doubt, please check this.

Linux special note

On Linux systems, a particular problem can occur. When the U4B Virtual COM (Serial) connection is detected, the PC thinks that a modem has been connected and starts trying to send it Hayes AT-commands dating back to 1981, implemented on Hayes' 300-baud modem. Yes! 40 years ago…

The Operating System attempting to send AT commands to your U4B will certainly mess everything up. Not least because when U4B receives a carriage return character, it will enable Terminal Applications mode; this will send all sorts of characters back to the PC (as U4B thinks it is now talking to a terminal emulator). Disaster.

To fix this you need to issue the following commands to disable ModemManager:

```
sudo systemctl stop ModemManager
sudo systemctl disable ModemManager
sudo systemctl mask ModemManager
```

This will permanently stop ModemManager. If for some reason, you actually DO need ModemManager operational, for some other reason… well there IS a way to stop it just for U4B… but Google will be your elmer on this!

## 2.2	PC terminal emulator

I use Linux (XUbuntu 18.04) and I'm using the PuTTY terminal emulator. There are numerous other terminal applications which will work fine. You may have your own favourite. They are all capable of correct operation with U4B.

I start PuTTY using command line "sudo putty" then connect to U4B on /dev/ttyACM0 (or ACM1, ACM2 etc if ACM0 is already in use by another device). On Windows operating system it will be a COM port numbered for example, COM1. It is necessary to know which serial port is being used

by U4B. There is also a guide to identifying the serial port at http://qrp-labs.com/qlg2 (scroll down the page).

Set the size of the terminal window to 80 columns and 24 rows.
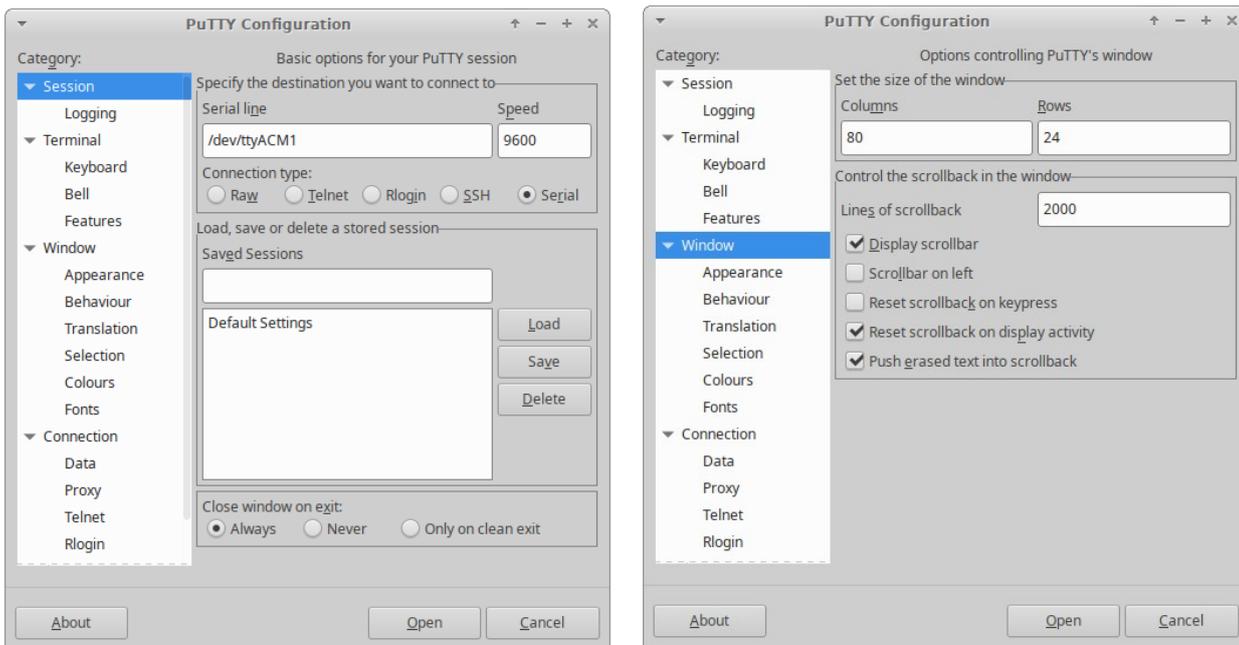
Set the PutTTY configuration as per the above, then click "Open" to start the connection. You will see a blank screen. Now press the Enter key and you should see the U4B applications menu.

For the purposes of this manual, to conserve printer ink for those who choose to print a paper version of the manual, inverse colouring will be used.

Note the firmware version number is shown at the top of the screen. Use the up/down arrow keys then press Enter to select the desired application.
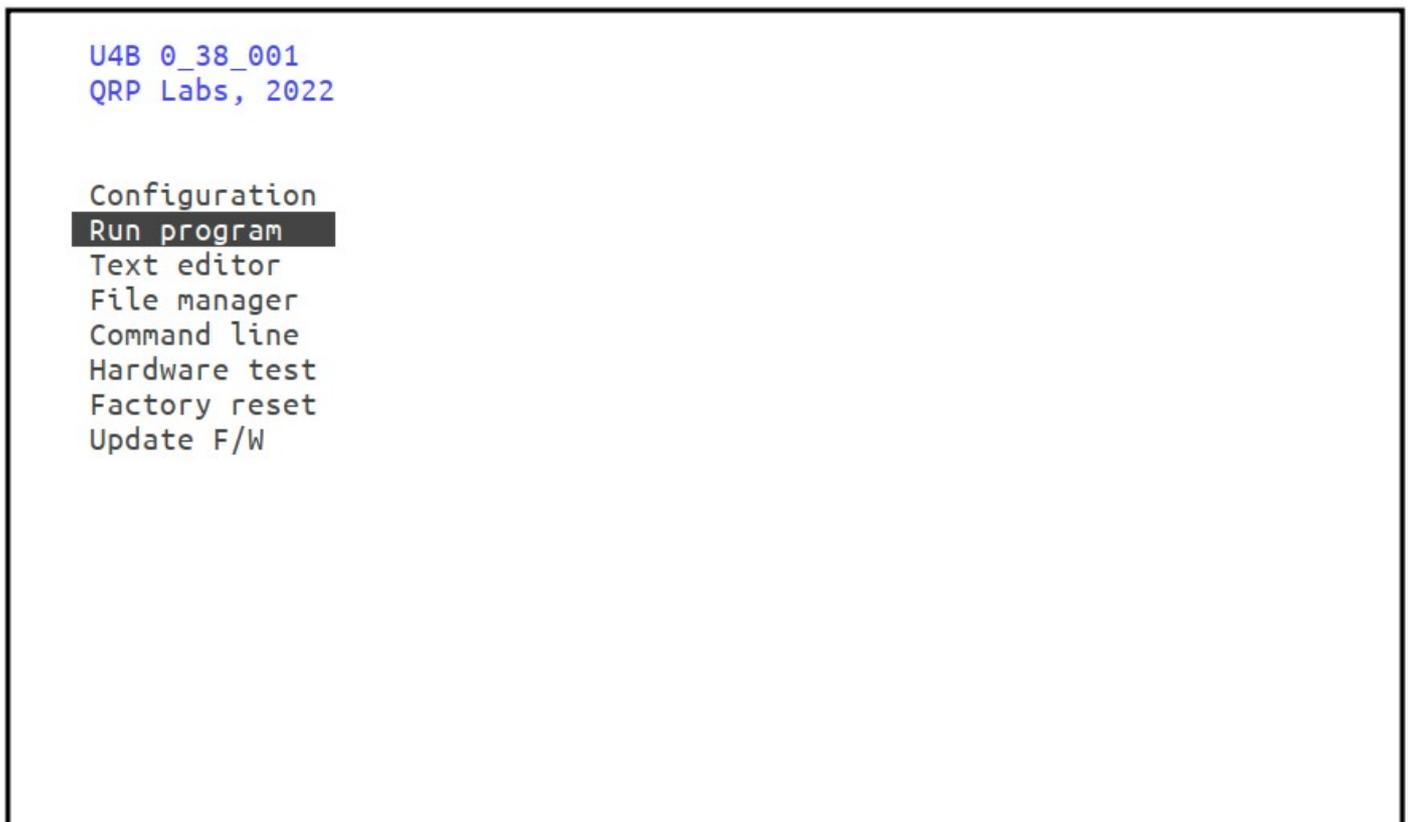
3.   Terminal applications

All terminal applications are activated by using the up/down arrow keys to highlight the desired option, then pressing Enter. To quit an application, use the Control-Q keyboard combination. An entry in the top level menu can also be selected by pressing a numeric key 0 to 7.

3.1   Configuration

```
U4B 0_38_001
QRP Labs, 2022

                          +---Configuration----------------------+
                          |                                       |
 Configuration            |   Callsign          G0UPL            |
 Run program              |   Band              20m               |
 Text editor              |   Channel           536               |
 File manager             |   Autostart program TRACKER           |
 Command line             |                                       |
 Hardware test            |   Frequency         14097180          |
 Factory reset            |   Telemetry format  QX6XXX XXXX XX     |
 Update F/W               |   Start minute      00                |
                          |                                       |
                          |   Use left & right arrow keys to      |
                          |      change Band and Channel.         |
                          |                                       |
                          +------------------------Ctrl-Q = Quit--+
```

The U4B Configuration screen is where the basic configuration of the U4B tracker and the QRP Labs WSPR tracking system is specified.

Callsign: Here you enter the callsign for the flight. This must conform with WSPR protocol callsign rules! Specifically:

- Must be 4-6 characters (longer callsigns cannot be sent using WSPR)
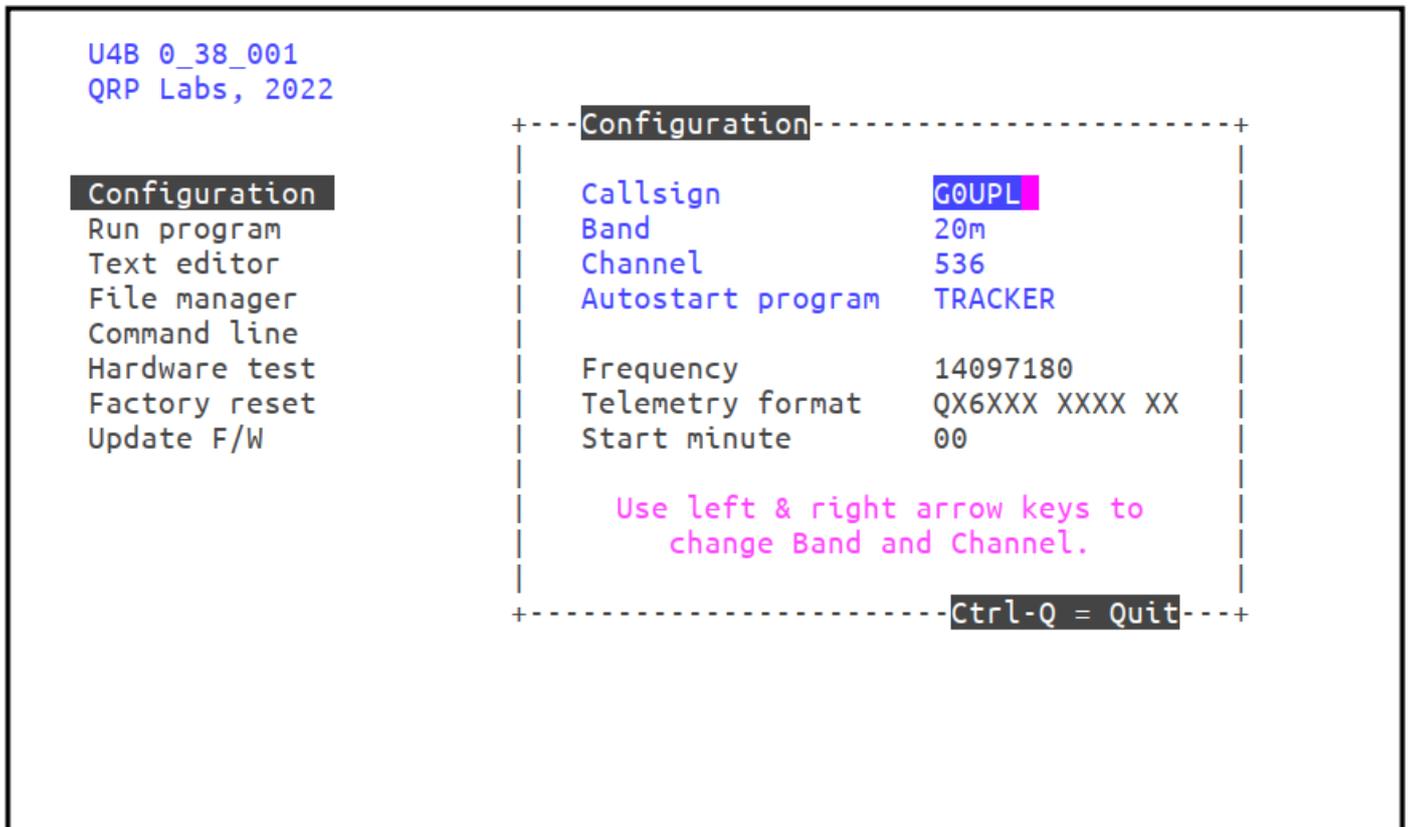- One or two characters which must consist of A-Z or 0-9
- Next character must be a number, 0-9
- Two or three more characters which must be A-Z

Select the callsign field and type your callsign in. There is no need to add blank spaces at the end; you cannot use the arrow keys to fix a mistake, just use the backspace key and re-enter the callsign.

Band: the amateur band for WSPR tracking telemetry. 20m is most popular for balloon tracking and is recommended. With the left and right arrows you can choose the operating band.

Channel: The QRP Labs tracking system uses 600 channels numbered 0 to 599. You can enter any channel. It is strongly recommended to use the Traquito channel reservation system (or other) to choose a channel which is not in use by another user.

Autostart program: The name of the program, which must be in the root directory, to run automatically at power-up when there is no USB connection. When the USB cable to the PC terminal emulator is connected, autostart is disabled. This is the main program which your U4B will run during flight, whenever it powers up with sufficient power from the solar cells.

Frequency: This is an information-only, non-editable field showing the center frequency of your WSPR tracking transmissions. There will be a few Hz variation around this due to temperature fluctuations – a TCXO such as the 25MHz TCXO synthesizer referenced used, is good – but does not completely eliminate drift down to zero.

Telemetry format: An information-only field indicating what format to expect the telemetry transmissions (second WSPR packet) to be in; the first character will be one of 0, 1 or Q and the third character will be numeric (0-9). All other characters of the WSPR callsign, locator and power will be substituted by the QRP Labs telemetry.

Start minute: the minute past the hour, repeating every 10 minutes, at which the first WSPR transmission will be sent. In the example shown, the first (standard) WSPR transmissions will occur at :06 :16 :26 etc past the hour. The telemetry transmissions are sent immediately after the standard WSPR transmission.

3.2 Run program

```
00:00:17   1
00:00:17   2
00:00:17   3
00:00:17   4
00:00:17   5
00:00:17   6
00:00:17   7
00:00:17   8
00:00:17   9
00:00:17   10
00:00:17   End.
```

This application simply runs the default program of the U4B, as defined in the Configuration screen. By default, this program is called "TRACKER". Any output printed from the program is shown on the screen. The Run program application is useful for final testing of your flight program. It can be useful to include debug PRINT statements that will help you to know that everything is operating properly. During flight when there is no host PC terminal emulator connected, the PRINT statements are harmless, they simply do nothing.

Each output line is prefixed by a timestamp in HH:MM:SS format indicating the system time.

In this example, I set up a very simple program called TRACKER that just uses a FOR NEXT loop to iterate a variable from 1 to 10 and prints the variable value at each step:

```
FOR I = 1 TO 10
    PRINT I
NEXT
```

Any errors that occur during the program execution will also be displayed on screen. Finally if and when the program completes, "End." will be shown. Remember that an actual flight program should contain an infinite loop that never completes. Once a program completes, the U4B system will sit and do nothing more until told to do so, or until the power is cycled. At 12km altitude there's nobody with a PC terminal emulator and USB connection to tell it what to do…

3.3 Text editor

The Text editor application is used to enter and edit your programs or other files on the disk.

It is also a full IDE (Integrated Development Environment). U4B programs are written in a simple type of BASIC programming language, which is compiled to check for errors and compact the program so it uses less disk space. The IDE allows compilation of the program, shows errors, and contains a debugger that may be used to step through the program one line at a time.

Note that the editor always uses UPPERCASE characters – you don't need to press Shift or CapsLock on your keyboard, everything will automatically be converted to UPPERCASE for you.

The upper part of the screens shows the text file contents; the lower lines provide information about the number of lines, percentage memory used, and state of the file (modified or not); together with a helpful list of available keyboard operations.

- Ctrl-O: Open a file from disk
- Ctrl-S: Save a file to disk

- Ctrl-A: Save As: saves a copy of the file to a new filename on disk
- Ctrl-C: Compile the BASIC program filename
- Ctrl-D: Switch to debugging mode
- Ctrl-Q: Quit the text editor application.

Control keys supported by the editor:

- Home: Moves the cursor to the start of the current line
- End: Moves the cursor to the end of the current line
- Backspace: deletes the character before the current cursor positione
- Delete: deletes the character at the current cursor position
- Page Up: scroll up one screen-full
- Page Down: scroll down one screen-full
- Arrow up/down/left/right: move the cursor one character in the specified direction

Other keystrokes are interpreted as normal text entered into the editor.

Pressing Ctrl-O opens a file selector screen showing the files on disk:

```
/
Name           Type    Size  Blocks  Start  Compiled
..
TRACKER        TEXT      33       1      1        No












CHOOSE FILE to open
ENT = Enter/Up Dir, or select file
Ctrl-Q = Quit
```

Here we have a single program, TRACKER, which is the default program specified in the Configuration screen. Normally it would, by default, contain a simple example tracking program; here I have replaced it with a very simple program for example purposes.

Use the cursor keys to select the desired file to open, then press Enter. Pressing Enter on the .. line moves the directory listing to the parent directory, if there is one; if there are sub-directories, pressing Enter when a sub-directory is listed, moves the directory listing into the sub-directory. Or you can press Ctrl-Q to quit the File Open operation.

Here's the example program:

```
FOR I = 1 TO 10
    PRINT I
NEXT
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
TRACKER - 3 lines (0%)
Ctrl-O Open, Ctrl-S Save, Ctrl-A Save As, Ctrl-C Compile, Ctrl-D Debug, Ctrl-Q =
```

The full BASIC syntax is described in a different section of this manual.

Pressing Ctrl-S will save the file if there are any changes; if it is a new file which does not yet have a file name, then Ctrl-S is equivalent to pressing Ctrl-A and will show you a similar screen to the above, to choose what directory to use and to enter the new file name.

```
FOR I = 1 TO 10
    PRINT I
NEXT
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
TRACKER - 3 lines (0%)
Compile successful! 16 bytes written
```

Now pressing Ctrl-C compiles the file, and writes it to disk as a Compiled file. If there are any errors, the compilation will fail without writing to disk, and the error will be identified on the bottom line. Note that the size of the compiled file is generally significantly less than when it was a text file.

A compiled file can still be opened for reading and writing in the text editor. But as soon as it is edited, the file when saved, reverts to being a non-compiled text file. There is no concept of a separate "source file" as in more sophisticated programming environments. Each program is just one file, which is either the raw non-compiled text file, or the compiled text file. Put another way – when a compiled file is opened in the text editor, the system has to de-compile it first to translate it back into text.

Once the file is compiled, you can press Ctrl-D to open the Debug screen!

```
FOR I = 1 TO 10                    +---Variables---------------------+
    PRINT I                        |                                 |
NEXT                               |                                 |
~                                  |                                 |
~                                  |                                 |
~                                  |                                 |
~                                  |                                 |
~                                  |                                 |
~                                  |                                 |
~                                  |                                 |
~                                  +---Output------------------------+
~                                  |                                 |
~                                  |                                 |
~                                  |                                 |
~                                  |                                 |
~                                  |                                 |
~                                  |                                 |
~                                  |                                 |
~                                  |                                 |
~                                  |                                 |
~                                  |                                 |
TRACKER - 3 lines (0%)             |                                 |
S = Step, R = Run, Ctrl-Q = Quit   +---------------------------------+
```

The Debug screen has three areas. The first is the normal text file contents screen, occupying the left half of the screen. Any lines of text that are longer than half a screen width, get truncated in this view. There are two additional windows on the right; the top right window is the "Variables" window and displays the 10 most recently last-accessed variables, sorted in order of most recently accessed at the top. The bottom right window shows any program output of the PRINT statement, preceded by the current timestamp. Certain other commands such as GPS and radio transmission commands also display status information on their progress, in the Output window.

You may now run through the program in two different ways:

- Press S to step forward one single line of the program
- Press R to run forward until either the program finishes, or the breakpoint is encountered

The "breakpoint" is the line containing current position of the cursor. By default, the cursor sits in the top left corner of the screen on the first line. But if you wish, you can move it to any subsequent line; when the program control reaches that line, it will pause, awaiting your next request (pressing the S or R keys).

```
FOR I = 1 TO 10          +---Variables---------------------+
     PRINT I             | I   5                           |
NEXT                     |                                 |
~                        |                                 |
~                        |                                 |
~                        |                                 |
~                        |                                 |
~                        |                                 |
~                        |                                 |
~                        |                                 |
~                        +---Output------------------------+
~                        | 00:53:35  1                     |
~                        | 00:53:36  2                     |
~                        | 00:53:37  3                     |
~                        | 00:53:38  4                     |
~                        |                                 |
~                        |                                 |
~                        |                                 |
~                        |                                 |
~                        |                                 |
~                        |                                 |
~                        |                                 |
TRACKER - 3 lines (0%)   |                                 |
S = Step, R = Run, Ctrl-Q = Quit +------------------------+
```

In this example, the S key has been pressed several times. The "Variables" windows hows variable I and its current value, 5. The Output window shows the PRINT statements that have been executed so far.

The program text itself (left panel) is in plain font; there are three possible colours as follows:

- Cursor position – shown here as a purple square, but it is green in PuTTY with a black background. The line containing the cursor is the current program breakpoint, unless the cursor is at the top left of the screen in its default position.
- Inverse video (the "PRINT I" line in this case) – this is the line of code which has just been executed in the last step.
- Coloured blue (or, in normal black background PuTTY, yellow) – this is the program line that will be executed in the next step.

As usual, press Ctrl-Q to quit the Debugger and go back to the editor.

3.4   File manager

```
/
Name           Type   Size  Blocks  Start  Compiled
..
TRACKER        TEXT    23       1      1       Yes
```

```
Ctrl-C = Copy        Ctrl-X = Cut       Ctrl-V = Paste      ENT = Enter/Up Dir
F = New file         D = New directory  R = Rename          DEL = Delete
E = Edit file        V = View file      Ctrl-D = Disk Mgr   Ctrl-Q = Quit
```

As its name suggests, this application is used for managing files on the disk.

The "disk" on U4B is implemented as a FAT12-ish file structure on the 128K I2C EEPROM chip. There are 1024 blocks, each of which is 128 bytes long. A program that is longer than 128K is stored in multiple blocks. Normally these would be consecutive blocks but in case a large enough consecutive memory area is not available for the size of the file being saved, it will be split into smaller pieces that are spread around the disk and later re-assembled.

The following keys can be used in File manager:

F: Create a new file – you will be prompted for the file name
E: Open the file in the text editor for editing
V: Open the file in the text editor for viewing (read-only)
R: Rename a file – you will be prompted for the new file name
D: Create a new directory – you will be prompted for the directory name
DEL: Delete the file or directory. If deleting a directory, the directory must be empty first.
Enter: Move down into the highlighted sub-directory (listed near the top of the file listing) or into the parent directory which is indicated by the .. right at the top of the file listing, if you have highlighted this .. pair of characters.
Ctrl-C: Copy a file into memory
Ctrl-X: Cut a file – this is like copying, then deleting
Ctrl-V: Paste the file that was previously copied or cut; you will be prompted for the file name
Ctrl-Q: Quit the File manager application
Ctrl-D: Open the Disk manager

The top row of the window shows the current directory location. In this example it is just / which means the root directory. The second line shows the column heading; the third row always shows .. which allows the operator to highlight this and press Enter to move to the parent

directory. Following this is the listing of sub-directories, followed finally by any actual files in this directory.

A directory is a line on the list which for which all the columns except Name are empty. A normal file, has the name specified as well as all the other columns.

Directory and file names in the U4B system must be 1 to 12 characters long and contain only letters and numbers.

The columns of the file manager have the following meaning:

Name: the file or directory name

Type: the type of the file; it is normally TEXT (meaning, a BASIC program; though you could write chapters of your life history too, if the spirit moves you). However if a file is created and manipulated from within BASIC, the file type will be DATA.

Size: the number of bytes used by the file

Blocks: the number of 128-byte blocks used by the file

Start: the location of the starting block of the file

Compiled: Whether or not the file is compiled. Only compiled programs are executable by the RUN command, Run program screen, or in the text editor debugger.
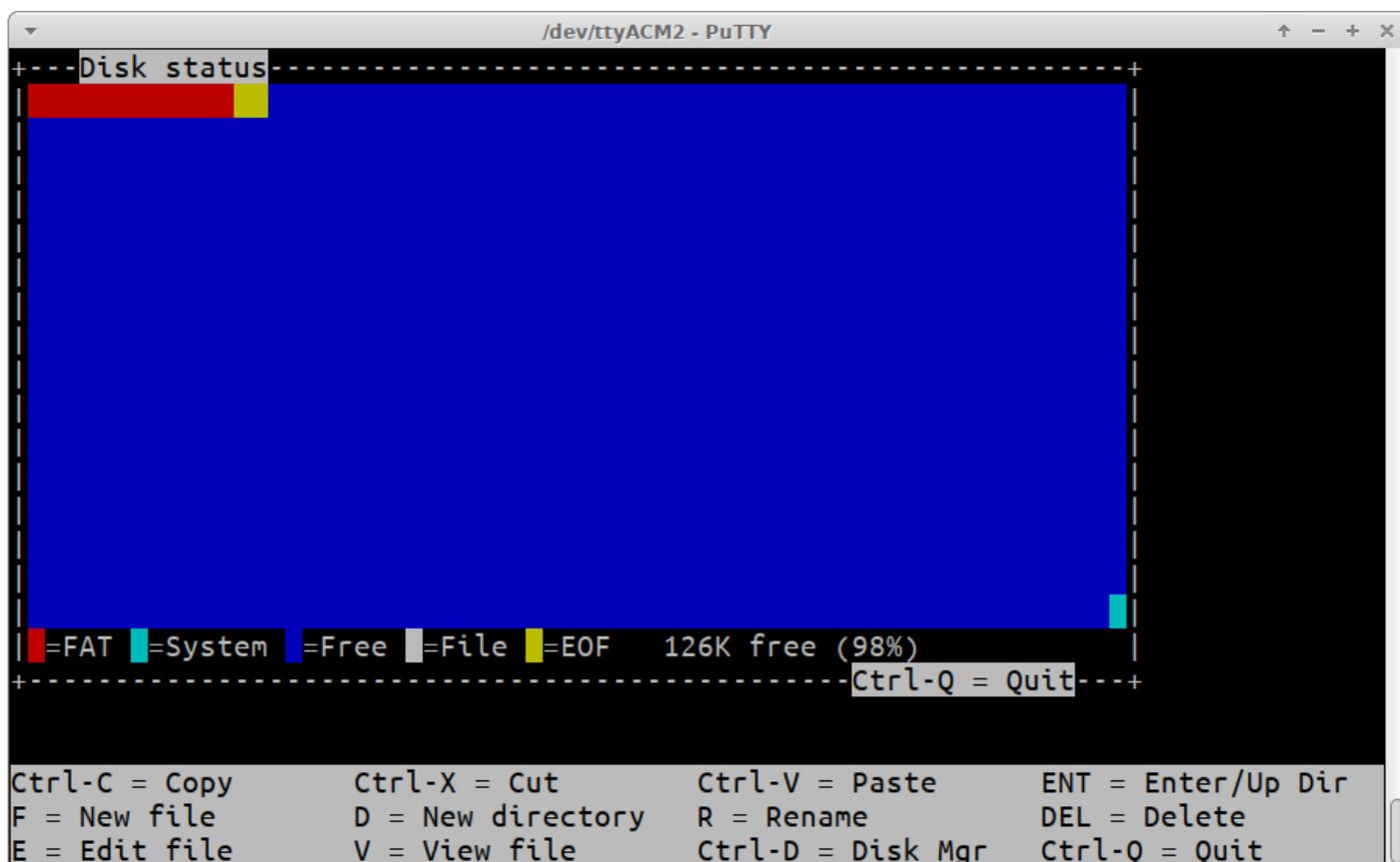
The disk manager Is a pop-up window shown here in its proper colours (not inverted as most of the images in this document are, to save your printer ink).

The window shows all 1024 blocks (128-bytes each) of the 128K disk, and colours them according to their current status, as follows:

Red: The first few blocks on the disk are the File Access Table. They specify the allocation and linkage of the remaining blocks on the disk.

Pale blue: System. There is a single, reserved, system block right at the end of the disk. It contains system variables, including storing information that you edit in the Configuration application such as your Callsign.

Dark blue: Free blocks, not allocated to anything; these are counted and used to display the amount (and percentage of the disk) which is free, at the bottom right of the window. In this example, 126K of the 128K disk is free, which is 98% of the disk.

White: Allocated to a file. Note that directories also look like files. The root (top level) directory is the first block after the FAT area.

Yellow: The last block of a file (EOF = End Of File); this block does not link onward to any further blocks.

Press Ctrl-Q to close the Disk manager window.

3.5 Command line

```
$ DIR
Name            Type    Size  Blocks  Start  Compiled
TEST
TRACKER         TEXT     23      1      1       Yes
$ RUN TRACKER
01:45:36  1
01:45:36  2
01:45:36  3
01:45:36  4
01:45:36  5
01:45:36  6
01:45:36  7
01:45:36  8
01:45:36  9
01:45:36  10
01:45:36  End.
$ TYPE TRACKER
FOR I = 1 TO 10
       PRINT I
NEXT
$
$ PRINT I
10
$ ▮
```

The screenshot shows an example of the Command line application, with some examples of what can be done in the command line. All of the BASIC commands which do not involve program flow, can also be entered directly at the command line. That is to say, all BASIC commands except: END, GOTO, GOSUB, RETURN, IF, ELSE, ENDIF, FOR, NEXT. In this way, you can use the Command line to easily test various aspects of your system; for example, to immediately read a sensor, or control a GPIO output.

In addition to the BASIC statements other than the control-flow-affecting list above, the following commands can be entered directly at the command line:

DIR	List contents of the current directory
EXIT	Quit the Command line application
QUIT	Quit the Command line application
CLS	Clear screen
CD	Change Directory – for example if there is a sub-directory called TEST, CD TEST will make TEST the current directory. The command prompt will change from $ (meaning root directory) to TEST$.
DEL	Delete a specified file or directory. Directories must be empty before they can be deleted.
MKDIR	Create a specified directory. For example MKDIR TEST creates a sub-directory called TEST.
RENAME	Rename a file or directory. For example RENAME TEST TEST1 renames the sub-directory called TEST to TEST1
COPY	Copy a file. For example COPY TRACKER TRACKERBAK makes a copy of the file TRACKER called TRACKERBAK
TYPE	writes the contents of a file to the screen. For example TYPE TRACKER will write the TRACKER file to the screen.
EDIT	opens the specified file in the Text editor. For example EDIT TRACKER opens the file named TRACKER in the Text editor.
RUN	runs the specified file in the Command line screen, printing any output directly to the Command line screen. For example RUN TRACKER runs the TrACKER program.

3.6 Hardware test

```
U4B 1_00_002          +----Hardware test------------------------------+
QRP Labs, 2022        | Date           27-APR-22     GPIO 0    0    1945 |
                      | Time           09:11:56      GPIO 1    1    2063 |
                      | Si5351A TX     OK            GPIO 2    0    1981 |
Configuration         | LM75 temp      310 K         GPIO 3    1    2097 |
Run program           | Battery        4.54 V        GPIO 4    0    1980 |
Text editor           | System freq.   8001318       GPIO 5    1    2105 |
File manager          | TCXO freq.     25000000      GPIO 6    0    1985 |
Command line          |                              GPIO 7    1    2086 |
Hardware test         | GPS Longtitude 02            GPIO 8    0    1959 |
Factory reset         | GPS Latitude   36            GPIO 9    1         |
Update F/W            | GPS Altitude   0.            GPIO 10   1         |
                      | GPS Locator    KM            GPIO 11   1         |
                      | GPS Validity   A             GPIO 12   1         |
                      | GPS Fix        3D            GPIO 13   1         |
                      | GPS Sats Fix   9             GPIO 14   1         |
                      | GPS Tracking   5             GPIO 15   1         |
                      | sat SNR        36            GPIO 16   1         |
                      |                              GPIO 17   1         |
                      | Calibrating system freq...   GPIO 18   1         |
                      |                                                  |
                      +--------------Ctrl-R = Raw GPS-------Ctrl-Q = Quit---+
```

The Hardware test screen provides a comprehensive diagnostic of your hardware.

All 19 IO signal states are shown; both the digital readings of the pin (0 or 1) and the analog level 0 to 4095 for GPIO pins 0 to 8. The screen also tests the Si5351A Synthesizer and shows the LM75 temperature (in Kelvin) and the measured battery voltage. Date, Time and the GPS parameters are all derived from the serial data stream arriving from the GPS module.

The data is updated once per second.

If there are any fails, they are coloured red. A common "failure" is all the GPS parameters being coloured red; this is normally (hopefully) just a matter of waiting longer for a GPS satellite fix. It can take several minutes, depending on the sky view and how well you built a GPS antenna.

Pressing Ctrl-R fills the screen with raw scrolling text directly from the GPS module; pressing any key closes this raw GPS text mode.

After a Factory reset

After a Factory reset, the system and reference oscillator (TCXO) values are the default 8MHz and 25MHz values respectively. Calibration is required. To indicate this, the "Hardware test" item on the main U4B menu will be coloured red.

The hardware test application calibrates these oscillators automatically, and the values are displayed on the hardware test screen. When the oscillators are not yet calibrated, the values ("System freq." and "TCXO freq.") are coloured red.

The hardware test phases are indicated by a yellow status screen at the bottom left of the window: "GPS satellite acquisition" (variable duration), "Calibrating system freq" (about 6 seconds) and "Calibrating TCXO freq" (about 20 seconds).

Oscillator calibration always takes place in the hardware test screen, not just after a factory reset.

3.7   Factory Reset

```
U4B 0_38_001
QRP Labs, 2022


Configuration
Run program
Text editor
File manager
Command line
Hardware test
Factory reset
Update F/W




Factory Reset? Are you sure? If you are sure, press Y
```

Factory reset is a get-out-of-trouble facility which resets your U4B completely to the default factory configuration. All entries of the Configuration screen are returned to their factory default settings. The disk is formatted, deleting all files.

Before doing a factory reset, you are asked if you are sure; press Y to continue, or any other key to quit. After the factory reset, the default TRACKER program is also restored automatically.

3.8   Update F/W (Firmware)

On occasion QRP Labs may make available updated firmware for U4B, in order to deliver bug fixes or functionality enhancements.

U4B contains a firmware update procedure for STM32-series microcontrollers, called QFU (**Q**RP Labs **F**irmware **U**pdate) which provides the following features:

- **Easy** – anyone can do the firmware update

- **No additional hardware required:** only a standard USB A-B cable (or micro-USB cable if you have installed a micro-USB connector)

- **No additional software required:** just the standard file manager application that is already available on any PC

- **No drivers:** no special drivers need to be installed, the existing drivers on any modern PC operating system are used

- **Works on any PC Operating System:** and in the same way: Windows, Linux, Mac

- **Secure:** firmware files are published on the QRP Labs website and are encrypted using 256-bit AES encryption technology

```
U4B 0_38_001
QRP Labs, 2022


Configuration
Run program
Text editor
File manager
Command line
Hardware test
Factory reset
Update F/W



FIRMWARE UPDATE:

Terminal will be closed.
Open U4B as a USB Flash drive on your PC,
then copy in new firmware file.

Press Y to proceed or any other key to cancel.
```

Selecting firmware update, then pressing the Y key proceeds with putting the U4B into bootloader mode. You must then remove the power, and re-apply power; when power is applied in bootloader mode, the U4B onboard LED will flash steadily.

To get back to normal operating mode, again cycle the power; on the next power-up, the U4B will be in normal operating mode.

USB Flash memory stick emulation:

In the firmware update mode, the U4B pretends to be a USB Flash memory stick, having a 4MByte capacity and implementing a FAT16 file system. This virtual "Flash stick" contains two files:

1. the firmware program file of the U4B microcontroller. You may read the file from U4B, or write a new one, just by dragging files in your file manager application. When you copy in a new firmware file, U4B decrypts it and installs it. You will then need to cycle the power to boot up again in normal operating mode.

2. EEPROM (disk image) contents: the entire contents of the U4B disk (not individual files). This is useful for creating a backup, so that during your development work you can get back to a normal state easily. Again, you can read the file from U4B or write a new one to U4B, simply by dragging files in your file manager application.

On entering the Firmware update procedure, a pop-up window should appear on your PC. On my system (Linux XUbuntu 18.04) it looks like this:
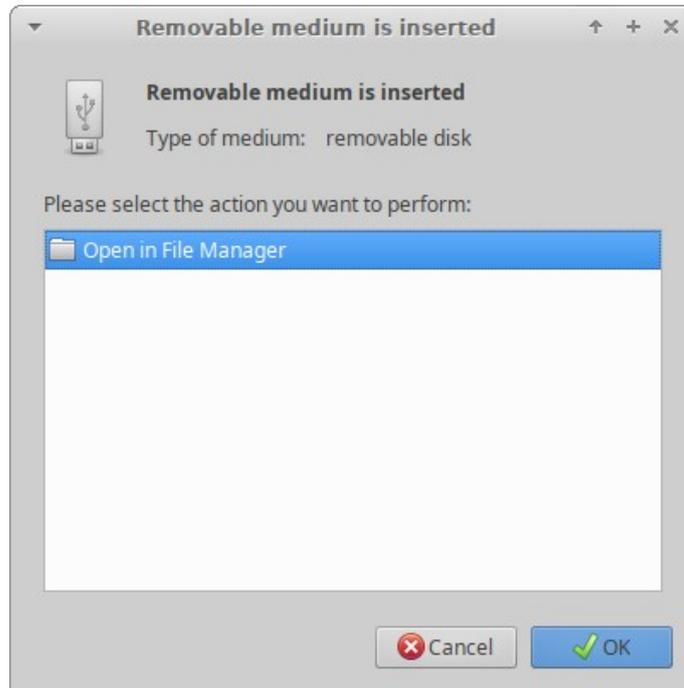
Click the OK button.

The File Manager window will then open, and on my system looks like this:
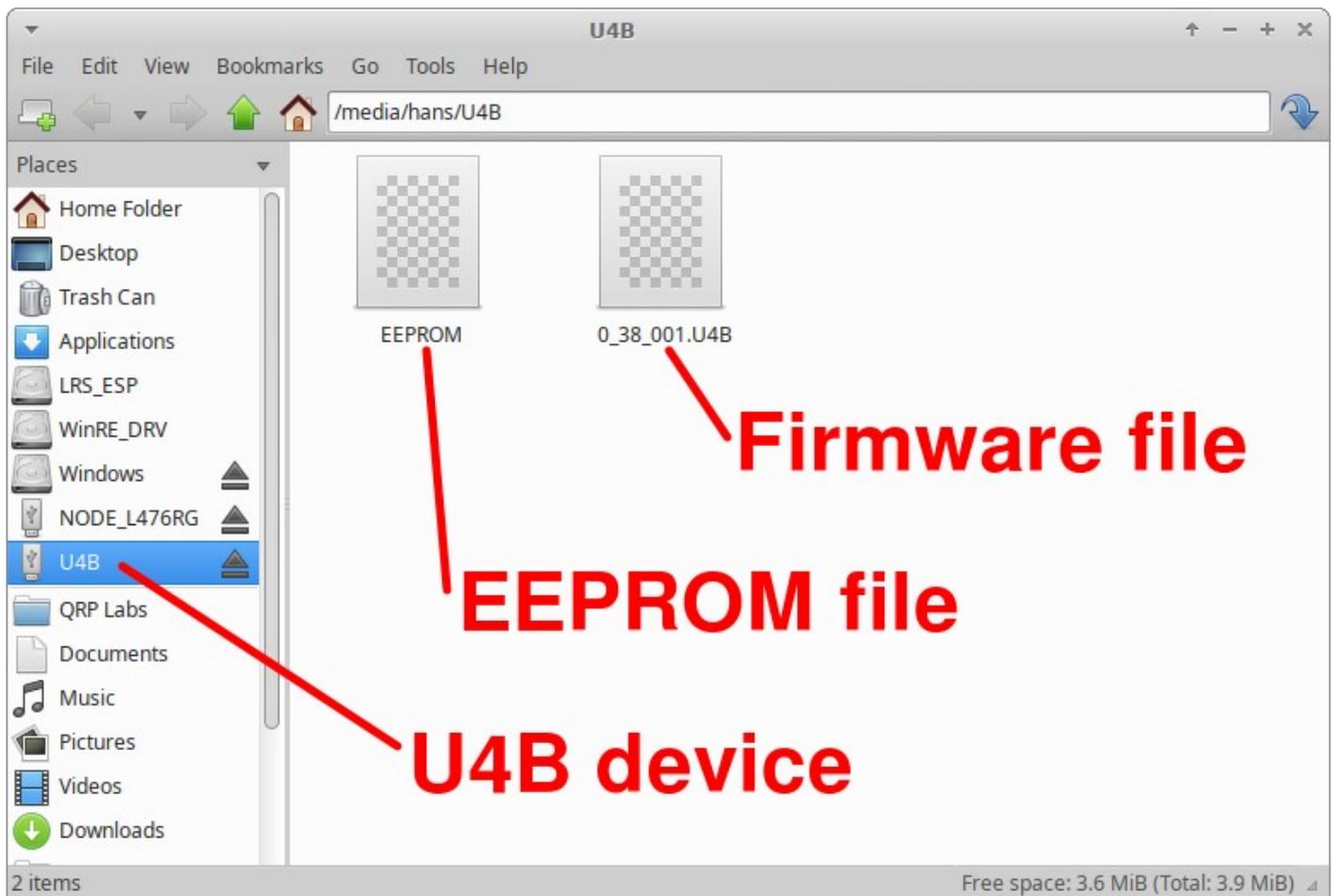
U4B appears as a removable USB Flash device named "U4B", and the folder contains two files. The firmware file shows as a single file which in this example is named 0_38_001.U4B. The EEPROM file is always named EEPROM. You can read and write EEPROM files in order to make and restore backup copies of your configuration etc.

The firmware file name must not be longer than 8 characters, and cannot contain punctuation or spaces; the file extension must be no more than 3 characters. This is because the file system emulation is FAT16 and these are the specifications of the FAT16 format.

You may check the properties of the file and will note that it is a 124K file. U4B firmware images are always a 124K file. The creation date and modification date etc. have not been set, because it was important to minimize the size and complexity of the QFU bootloader, in order to maximize the space available to the application firmware.

You may copy the existing firmware file to another directory of your computer. Crucially, to do the firmware update, all you need to do is copy the new firmware file to this U4B "Flash disk".

Download the new firmware file from the QRP Labs website, unzip it, and simply drag it into the folder where the existing firmware file version is shown. Or copy and paste it, however you wish.

The file on the QRP Labs website is a ZIPPED file, please be sure to unzip it to get the .U4B file before copying it to U4B.

As soon as you copy the new file to the U4B QFU "flash drive", the U4B QFU bootloader erases the current program from its memory and installs the new one.

The U4B firmware is 256-bit AES encrypted and this means:

- The encrypted U4B firmware file will only work on a QRP Labs U4B board, it cannot be installed on any other board, even one containing the same processor.

- No other firmware file will work on the QRP Labs U4B board except an official QRP Labs encrypted U4B firmware file.

The procedure will vary slightly for different Operating systems but in all cases is just a simple matter of copying the new firmware file to the emulated U4B QFU USB Flash drive.

The above firmware update procedure works on ANY modern OS because the QFU bootloader emulates a USB Flash memory stick with the USB Mass Storage Device (MSD) class, for which drivers are already present.

The QFU bootloader implements a USB device stack (Mass Storage Device class), emulated FAT16 file system, Flash erase/write, and 256-AES encryption.

4. U4B BASIC

The U4B implements a simple yet powerful Operating System called QDOS (**Q**RP Labs **D**isk **O**perating **S**ystem). compiled very simple virtual machine as the flight computer. The virtual machine runs compiled BASIC programs that are stored in a simple file system implemented on a 128K EEPROM. All the hardware functions of U4B are accessible from the BASIC commands. There are BASIC commands to transmit WSPR and other digital communication protocols, as well as control I/O pins (read or write), or read sensors attached to the I2C bus.

The simple BASIC programming language is very easy to use and learn, and plenty powerful enough for this application.

Compiled BASIC programs are very compact. You can therefore fit quite large programs in the available EEPROM space!

When the U4B is powered up, one program whose name is specified in the Configuration application (default name "TRACKER") starts executing automatically.

4.1 QDOS BASIC programs

QDOS BASIC programs run in a 64-bit Virtual Machine in the processor of the U4B. The BASIC is simple but yet powerful and flexible enough to permit a lot of experimentation and different flight tracking and data collection scenarios.

BASIC programs are stored on the 128K disk drive which is implemented on a 128K EEPROM chip for persistent storage when the power is off.

BASIC programs may be a maximum of 5,000 characters of text due to limited RAM capacity in the microcontroller chip of the U4B. However, a BASIC program may run a different program using the RUN command. So practically if an enormous BASIC program is required, it may be split up into lots of smaller pieces, each of which is under 5,000 characters of text and can be in a separate file and tested etc separately.

QDOS contains a full screen text editor which communicates with the user via a PC terminal program and the standard USB-micro cable to the U4B. BASIC programs are compiled within the text editor. Only compiled programs can be run. Compiled programs are smaller than their un-compiled text equivalent; therefore even if your program is 5K of text, when compiled it will occupy a considerably smaller space on the disk.

The editor also contains a debugger which may be used to step through the code one line at a time.

Some useful tips follow.

Line numbers: BASIC programs can contain line numbers, but do not have to.

```
10 FOR I = 1 to 10
20   PRINT I
30 NEXT
```

and

```
FOR I = 1 to 10
    PRINT I
NEXT
```

are completely equivalent. However, if your program uses GOTO or GOSUB, these statements require a line number; therefore the target (destination) line for a GOTO or GOSUB actually must have a line number.

Indentation: Indentation (spaces or tabs) can be used to make a program more readable. A tab is equivalent to 4 spaces. Indentation is not a requirement. It's just good programming practice.

```
FOR I = 1 to 10
    PRINT I
NEXT
```

and

```
FOR I = 1 to 10
PRINT I
NEXT
```

are completely equivalent.

LED control: The U4B board has an onboard red LED that can be a useful status indicator during debugging and ground testing. In flight, it can be used as a strobe effect to warn approaching aircraft. Use:

```
OUT 19 1
```

to switch on the LED and

```
OUT 19 0
```

to switch off the LED.

P.S. I'm joking about warning approaching aircraft in flight.

RF output power control: U4B can transmit using the Si5351A chip's Clk0 pin, whereupon the Clk1 pin is grounded. This is called the low power mode and power output is approximately 9mW. It can also drive the Clk0 pin and Clk1 pin in antiphase (180-degree phase shift), which results in an output power of approximately 27mW. By default, the U4B will operate in low power mode. The HP variable controls the transmitter power setting.

```
LET HP = 1
```

sets high power mode. Whereas:

```
LET HP = 0
```

sets the low power mode (default). If the high power mode is set, you will find that the U4B requires higher light levels to be able to operate properly. As the sun comes up in the morning and

the angle of incidence on the solar panels increases, it will take longer to reach the point where there is sufficient power for the High Power mode. Similarly in the evening, the U4B will stop working sooner. If you are really clever, you can measure the battery voltage and use BASIC commands to choose between low and high power modes.

4.2 Variables

There are 26 general purpose variables, they are named A, B, C and through to Z. Each variable is a 64-bit integer.

There are also some system variables, with two-character uppercase variable names as listed below:

Name	Writable	Contents
AT	No	GPS Altitude (meters)
BT	No	Battery voltage (millivolts)
C0	Yes	Counter #0 value (16-bit)
C1	Yes	Counter #1 value (16-bit)
CH	Yes	Telemetry channel (0 to 599)
CL	No	GPS calibration completed successfully: 1, otherwise 0 if not
FS	Yes	System frequency (Hz), nominally 8 MHz
FQ	No	Transmitter frequency (Hz)
FR	Yes	PLL Reference frequency (Hz), nominally 25 MHz
GL	No	GPS 3D satellite lock acquired: 1, otherwise 0 if not
GS	No	GPS Speed (knots)
GV	No	GPS Validity bit: 1 for "A" flag, 0 for "V" flag
HP	Yes	Transmitter High Power mode: 1 = High power, 0 = Low power
SC	No	Number of satellites tracked
SD	No	Realtime clock Date has been set (by GPS): 1, otherwise 0 if not
SF	No	Number of satellites in fix
SN	No	Average SNR of satellites tracked
SQ	No	Sequence number for Custom Telemetry (WSPRtv tx_seq)
ST	No	Realtime clock Time has been set (by GPS): 1, otherwise 0 if not
TK	No	Onboard temperature sensor reading now (Kelvin)
TT	Yes	Temperature reading transmitted in telemetry (Kelvin)
US	No	USB host connected: 1m otherwise 0 if not

Many of these variables are read-only (as indicated in the table above). All general purpose variables A to Z are read/write.

Variables may be set by the command syntax:

LET <variable> = <expression>

Examples:

```
LET A = 4
LET B = TK
LET C = D * (E + 2)
```

Variables may also be inserted into strings by using the # tag followed by the two letter variable name; the general purpose variables are referred to in this way using tags #VA to #VZ.

For example:

```
PRINT TK
PRINT "#TK"
```

These two statements produce the same result. Here we briefly introduce the PRINT statement (described in more detail later); when used in a PROGRAM that is running in DEBUG mode or RUN mode, it prints the prescribed string to the terminal screen. It's very useful when coding, to test the code. PRINT can also be used in the Command line application and is interpreted immediately.

Notes:

1. FS and FR commands are read/write, however writing them does not generally make much sense and should be done only with extreme caution; it should be left to the GPS calibration functions to accurately set system and reference frequencies.

2. All variables relating to GPS information, hold the value obtained from the most recent GPS statement.

3. SD and ST variables are set to 1 when the date and time (respectively) are obtained from the GPS. They remain at 1 for the rest of the operating session, until the power is switched off.

4. Reading the temperature variable (TK) causes the system to immediately read the current temperature from the onboard LM75 temperature sensor. Therefore every time you read this variable, the value returned may potentially be different.

5. The Telemetry temperature variable (TT) is set during the GPS statement, to the value read from the onboard temperature sensor. By default this will be encoded into the transmission sent by the TELE (telemetry) command. However, there may be occasions when you wish to transmit a different temperature in the telemetry. An example of this could be, that you have provided an external temperature sensor that is outside thermal insulation, and you want that value to be encoded into the telemetry rather than the warmer reading from the onboard sensor.

6. There are even more variables available, as #-tags in strings. Some of these can provide information to your program that is not available from the list of variables above; such as latitude and longitude, for example. These variables are detailed in the next section.

4.3   Tags in strings

A string can contain tags that are substituted with variables. This is a useful way to output information from the GPS and other variables. The string can be used for transmission (CW message for example), or for the PRINT statement, writing to files, etc.

Tag substitution is specified by the two character tag code preceded by a #. For example

```
PRINT "#LT"
```

prints the latitude to the terminal, for example:

```
5130.0517 N
```

The following table lists available tag codes:

Code	Example	Contents
A0	1864	Analog channel 0, value 0 to 4095 for voltage 0-3.3V
A1	2055	Analog channel 1, value 0 to 4095 for voltage 0-3.3V
A2	1906	Analog channel 2, value 0 to 4095 for voltage 0-3.3V
A3	2114	Analog channel 3, value 0 to 4095 for voltage 0-3.3V
A4	1927	Analog channel 4, value 0 to 4095 for voltage 0-3.3V
A5	2087	Analog channel 5, value 0 to 4095 for voltage 0-3.3V
A6	1920	Analog channel 6, value 0 to 4095 for voltage 0-3.3V
A7	2081	Analog channel 7, value 0 to 4095 for voltage 0-3.3V
A8	1919	Analog channel 8, value 0 to 4095 for voltage 0-3.3V
AT	23.5	GPS Altitude (meters)
BT	4276	Battery voltage (millivolts)
C0	123	Counter #0 value
C1	456	Counter #1 value
CH	245	Telemetry channel (0 to 599)
CL	1	GPS calibration completed successfully: 1, otherwise 0 if not
CS	G0UPL	Callsign as entered in the Configuration application (3 to 6 characters)
D0	0	Digital channel 0, value 0 or 1
D1	1	Digital channel 1, value 0 or 1
D2	0	Digital channel 2, value 0 or 1
D3	1	Digital channel 3, value 0 or 1
D4	0	Digital channel 4, value 0 or 1
D5	1	Digital channel 5, value 0 or 1
D6	0	Digital channel 6, value 0 or 1
D7	1	Digital channel 7, value 0 or 1
D8	0	Digital channel 8, value 0 or 1
D9	1	Digital channel 9, value 0 or 1
DA	0	Digital channel 10, value 0 or 1
DB	1	Digital channel 11, value 0 or 1
DC	0	Digital channel 12, value 0 or 1
DD	1	Digital channel 13, value 0 or 1
DE	0	Digital channel 14, value 0 or 1
DF	1	Digital channel 15, value 0 or 1
DG	0	Digital channel 16, value 0 or 1
DH	1	Digital channel 17, value 0 or 1
DI	0	Digital channel 18, value 0 or 1
DT	030921	Current date (UT, as DDMMYY)
FL		The first 20 characters of the last line read by the FILERD command
FS	8001207	System frequency (Hz), nominally 8 MHz
FQ	14097140	Transmitter frequency (Hz)
FR	24999997	PLL Reference frequency (Hz), nominally 25 MHz
GL	1	GPS 3D satellite lock acquired: 1, otherwise 0 if not
GS	0.32	GPS Speed (knots)
GV	A	GPS Validity bit: 1 for "A" flag, 0 for "V" flag
HP	0	Transmitter High Power mode: 1 = High power, 0 = Low power
LN	00008.5812 W	GPS position longitude
LT	5130.0517 N	GPS position latitude
M4	IO91	Maidenhead locator grid square (4 characters)
M6	IO91DK	Maidenhead locator grid subsquare (6 characters)
SC	7	Number of GPS satellites tracked
SD	1	Realtime clock Date has been set (by GPS): 1, otherwise 0 if not

SF	5	Number of GPS satellites used in Fix
SN	30	Average SNR of GPS satellites tracked
ST	50	Sequence number for Custom Telemetry (WSPRtv tx_seq)
ST	1	Realtime clock Time has been set (by GPS): 1, otherwise 0 if not
TK	300	Onboard temperature sensor reading now (Kelvin)
TM	055259	Current time (UT, as HHMMSS)
TT	300	Temperature reading transmitted in telemetry (Kelvin)
US	1	USB host connected: 1m otherwise 0 if not
VA	0	BASIC general purpose variable A
VB	0	BASIC general purpose variable B
VC	0	BASIC general purpose variable C
VD	0	BASIC general purpose variable D
VE	0	BASIC general purpose variable E
VF	0	BASIC general purpose variable F
VG	0	BASIC general purpose variable G
VH	0	BASIC general purpose variable H
VI	0	BASIC general purpose variable I
VJ	0	BASIC general purpose variable J
VK	0	BASIC general purpose variable K
VL	0	BASIC general purpose variable L
VM	0	BASIC general purpose variable M
VN	0	BASIC general purpose variable N
VO	0	BASIC general purpose variable O
VP	0	BASIC general purpose variable P
VQ	0	BASIC general purpose variable Q
VR	0	BASIC general purpose variable R
VS	0	BASIC general purpose variable S
VT	0	BASIC general purpose variable T
VU	0	BASIC general purpose variable U
VV	0	BASIC general purpose variable V
VW	0	BASIC general purpose variable W
VX	0	BASIC general purpose variable X
VY	0	BASIC general purpose variable Y
VZ	0	BASIC general purpose variable Z

Furthermore, the #-tag specification can be modified to choose only a selected part of the string.

The format for this is:

#<start character>.<character count><hash code>

Characters start at 1. The dot and character count may be omitted, in which case the tag substitution begins at the specified start character and continues to the end of the tag. The following examples should illustrate this:

Command	Result
PRINT "#TM"	055259
PRINT "#5.6TM"	59
PRINT "#3TM"	5259
PRINT "#1.2TM:#3.2TM:#5.2TM"	05:52:59

The final example illustrates the use of multiple #-tags substitutions in one string, in this case used to re-format the time to contain colon separators.

The VAL function converts a piece of text to a number. Suppose you wanted to find the number of hours, and number of minutes, and assign these to the BASIC general purposes variables H and M. The following code would accomplish this:

```
LET H = VAL "#1.2TM"
LET M = VAL "#3.2TM"
```

4.4 Operators

The following table shows the available operators:

+	Addition
-	Subtraction
*	Multiplication
/	Division
&	Bit-wise AND of two 64-bit variables
|	Bit-wise OR of two 64-bit variables
%	Modulus
=	Assignment

These may be used in Expressions and assignments (see below). Note that integer division uses the truncation method, the same as in the C programming language. Truncation rounds towards zero regardless of whether the result is positive or negative, the fractional part is removed. The % operator acts on negative numbers in the C style (sign is that of the numerator, not divisor).

4.5 LET statement (assignment)

A value is assigned to a variable using the LET statement. Example:

```
LET A = 5
```
This sets the value of variable A to 5.

4.6 Expressions

Variables, operators and functions may be combined in expressions. Parenthesis may be used. For example

```
LET A = 5
LET B = (A + 2) * 3
LET C = AT
```

This sets the value of variable A to 5, and B to 21. Variable C is set to the GPS altitude in meters.

Many other functions can also be used in expressions, these are described below.

4.7 Functions

4.7.1 IN function

Reads the <u>digital</u> value of one of the 19 GPIO pins; the result is 1 (high) or 0 (low). GPIO pins are numbered 0 to 18. If you read a pin numbered greater than 18, the pin number is interpreted modulus 19. For example if you read pin 22, it is the same as reading pin 3.

Syntax:

        IN <GPIO pin number expression>

Example:

        LET A = IN 6

Reads the digital value from pin 6 and puts it in variable A.

<GPIO pin number expression> is an expression. Therefore the following examples also work:

        LET P = 6
        LET A = IN P

or

        LET A = IN 4 + 2

Both of these examples also read the digital value from pin 6 and put it in variable A.

4.7.2 INA function

Reads the <u>analog</u> value of one of the 9 GPIO pins numbered 0 to 8. These pins are capable of both analog input and digital input. The voltage must be in the range 0 to 3.3V (supply voltage). The result is a 12-bit number representing the converted voltage to digital (Analog to Digital Conversion), which is to say, a number in the range 0 to 4095.

If you read a pin numbered greater than 8, the pin number is interpreted modulus 9. For example if you read pin 12, it is the same as reading pin 3.

Syntax:

        INA <GPIO pin number expression>

Example:

        LET A = INA 6

Reads the analog value from pin 6 and puts it in variable A.

<GPIO pin number expression> is an expression. Therefore the following examples also work:

        LET P = 6
        LET A = INA P

or

        LET A = INA 4 + 2

Both of these examples also read the analog value from pin 6 and put it in variable A.

4.7.3 RAND function

Returns a pseudo-random number in the range 0 to 2,147,483,647.

Syntax:

    RAND

Example:

```
LET A = RAND
```

Puts a random number in variable A. The random number seed is updated whenever the GPS statement executes, from information in the GPS data.

4.7.4 VAL function

Converts the contents of a string to a number. Additionally, start and end characters in the string may be optionally specified. The conversion continues until the first non-numeric character is encountered; if there are no numeric characters, the value is 0.

Syntax:

    VAL <string>
    VAL <string> <character count expression>
    VAL <string> <start character expression> <character count expression>

Examples:

```
LET A = VAL "1234567"
```

Converts the string to a number and therefore assigns value 1234567 to variable A.

```
LET A = VAL "1234567" 4
```

Converts only the first 4 characters of the string to a number. The leftmost 4 characters are used, because the start character has not been specified and 1 is assumed. The value 1234 is assigned to variable A.

```
LET A = VAL "1234567" 5 2
```

Converts only 2 characters of the string to a number, starting at character number 5. The value 56 is assigned to variable A.

```
LET A = VAL "123HELLO"
```

Assigns value 123 to variable A, because the conversion continues until the first non-numeric character is found (the H of HELLO).

```
LET A = VAL "HELLO123"
```

Assigns value 0 to variable A, because there are no numeric characters at the beginning (left) of the string.

```
LET A = VAL "1234567" 4 + 1 1 + 2
```

Assigns value 567 to variable A. This is because the <start character expression> and <character count expression> are both expressions. An expression can include variables, or other functions. In this case, the start character is 4 + 1 (which is 5) and the character count is 1 + 2 (which is 3). So 3 characters from the string are converted, starting at position 5.

Strings can contain #-tag substitutions:

```
LET H = VAL "#1.2TM"
LET H = VAL "#TM" 1 2
```

Both of these are different ways of doing the same thing. In the first one, the tag is replaced by the first two characters of the #TM (time) – start character is 1 and character count is 2. In the second example, the whole #TM string is used, but the VAL function has start character 1 and character count 2 specified. So in both cases, the first two characters (the hour of the time) are converted to a number and put in variable H.

4.7.5 INSTR function

Returns the position of a sub string within a string, or 0 if not found.

Syntax:

INSTR <string to be searched> <string to look for>
INSTR <string to be searched> <string to look for> <start position expression>

<start position expression> is an expression and is optional.

Examples:

```
LET B = INSTR "HANS" "NS"
```

puts the value 3 in B because the substring "NS" is found at position 3 of the search string "HANS".

```
LET B = INSTR "HANS" "Z"
```

puts the value 0 in B because the substring "Z" is not found at all in the search string "HANS".

```
LET B = INSTR "PROCRASTINATION" "A" 10
```

puts the value 11 into variable B, since the search for the substring "A" in the search string, is started at character position 10.

Start position is an expression, and of course you can also use #-tags in both the string being searched and the string being looked for. So:

```
LET B = INSTR "#TM" "#VZ"
```

looks for the value of variable Z in the current time. In my case the current time is 065924 and I have not put anything into variable Z in this operating session, so it remains at its default value of zero. So the value 1 is written into variable B because the substring "0" is found at character 1 of the string being searched.

4.7.6 LEN function

Returns the length of the string.

Syntax:

    LEN <string>

Example

    LET C = LEN "HANS"

Puts the value 4 into variable C.

    LET C = LEN "#TM"

puts the value 6 into variable C, because the #TM tag is replaced by the time, which I is a 6-character value of the form HHMMSS (Hours Minutes Seconds).

4.7.7 I2CR function and I2CR16 functions

Reads from an device attached to the I2C bus. Some devices have 8-bit registers (such as the Si5351A and LM75 chips), others have a much larger register space and a 16-bit address format (such as the I2C serial EEPROM chip). I2CR is for 8-bit devices. I2CR16 is for 16-bit devices.

Syntax:

    I2CR <device address expression> <register address expression> <data length>
    I2CR16 <device address expression> <register address expression> <data length>

Both addresses are expressed as decimal values and are expressions.

Data length is an expression and is optional; if omitted the default is 1 (one byte). Other acceptable values are 2, 3 or 4 (bytes).

The U4B has three existing devices attached to its I2C bus. You may add additional I2C sensors. The additional device addresses must not conflict with the existing three devices on the bus.

Existing U4B I2C devices:

Address	Device
160	EEPROM, lower half of 128KByte chip, 16-bit register address
162	EEPROM, upper half of 128KByte chip, 16-bit register address
192	Si5351A (Synthesizer chip), 8-bit register address
144	LM75 (Temperature sensor), 8-bit register address

It should be noted that the I2C bus address is often specified as a 7-bit value; this 7-bit value is put in the most significant bits of a byte, and the least significant bit is set to 0 for a Write operation or 1 for a Read operation. The <device address> parameter expected by the I2CR/I2CR16 functions is an 8-bit value which is really the Write operation 8-bit address. Therefore if your device specifies a 7-bit I2C address, you should multiply this by 2 for use in these functions.

Examples:

    LET A = I2CR 144 0

Reads register 0 from I2C device at address 144. This is the onboard LM75 temperature sensor and register 0 simply contains the temperature in Celsius. So value 26 is assigned to variable A (in my case now, where the temperature is 26C).

```
LET E = I2CR16 160 43845
```

Reads register 43845 from device 160. Device 160 is the 128K serial EEPROM chip, and it uses a 16-bit address space which is why the I2CR16 function is used. 43845 is the memory address.

The companion statements are I2CW and I2CW16 for writing to I2C devices, and these are described later. You should generally NOT write to the onboard system devices (Si5351A, EEPROM and LM75), particularly not the EEPROM which will corrupt the file system.

4.7.8 FILEEOF function

Returns 1 if the end of the specified file has been reached during read operations (EOF = End Of File), or 0 if the end of file has not been reached. If the file is not open for reading, the result is undefined.

Syntax:

FILEEOF <file number expression>

For example:

```
LET F = FILEEOF 2
```

sets variable F to 1 or 0 depending on whether the end of file 2 has been reached or not.

4.7.9 PACK function

Helps create packed BigNumber to transmit using Custom Telemetry command TELEC.

Syntax:

PACK <BigNumber expression> <Modulus expression> <Data expression>

For example:

```
LET B = PACK B 20 Y
```

In this example we are building a BigNumber for Custom Telemetry transmission in variable B. The first argument to the PACK function is the BigNumber so far (because we may have been packing other data values into BigNumber already). The second argument, here 20, is the number of possible values the data can take. The third argument, here Y, is the data being packed. So in this case, we have a data field, which is in variable Y, and which can have 20 values (0 to 19).

4.7.10   ERLON and ERLAT functions

These functions create BigNumber packing data for Enhanced Resolution Longitude (ERLON) and Enhanced Resolution Latitude (ERLAT). These are native types 100 and 101 respectively, used on the WSPRtv website for enhanced position resolution.

Syntax:

```
ERLON <Modulus expression>
ERLAT <Modulus expression>
```

For example:

```
LET T = ERLAT 10
LET N = ERLON 10
```

This creates modulus 10 (meaning, ten possible values) enhanced resolution numbers from the latitude and longitude data read from the GPS module, and prepares them for packing into BigNumber. The Maidenhead grid subsquare is effectively divided up into a 10 x 10 grid of smaller squares within the grid subsquare.

See the TELEC command for more examples.

4.8 Loops, Conditionals and Program Flow

4.8.1 FOR..NEXT loop structures

The FOR..NEXT loop structure is useful where you want to run a certain block of statements several times. A variable is used as the counter.

Syntax:

```
FOR <variable> = <expression> TO <expression>
       <Statements>
NEXT
```

For example:

```
FOR A = 1 TO 5
    PRINT A
NEXT
```

Running this program simply loops through the statement (PRINT statement, in this case) 5 times, and prints the value of the loop variable A to the terminal each time. The result on the terminal screen is:

```
1
2
3
4
5
```

Indentation is not important, it is ignored by the compiler and is only useful to make the program more human-readable.

FOR..NEXT statements may also be nested. For example:

```
FOR A = 1 TO 4
    FOR B = 1 TO 3
        PRINT A, B
    NEXT
NEXT
```

This prints the following to the terminal:

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
4 1
4 2
4 3
```

You can nest a maximum of 4 loops within each other.

4.8.2 IF..ELSE..ENDIF structures

The IF statement and related ELSE and ENDIF statements are used to conditionally control program flow.

Syntax:

IF <expression> <relational operator> <expression>

<Statements>

ELSE

<Statements>

ENDIF

Valid relational operators are:

Operator	Function
=	Equals
<	Less than
>	More than

The ELSE section and its statements is optional. There must always be an ENDIF statement.

For example:

```
LET A = 5
IF A > 3
    PRINT A
ENDIF
```

Running this program prints the value of A (5) to the terminal, since when A > 3, the statement "PRINT A" is executed. The following example extends this to use the ELSE statement too:

```
LET A = 1
IF A > 3
    PRINT A
ELSE
    LET A = 3
    PRINT A
ENDIF
```

This program prints 3; the reason for this is that A is initially set to 1, which is not more than 3; therefore the ELSE part of the structure is executed; this sets A to 3 and then prints it.

Note that when you are typing in the program, the indentation does not matter. Indentation is ignored by the compiler. It is only visually useful to make the program more human-readable.

4.8.3 GOTO

Causes program execution to jump to the specified line number.

Syntax:

GOTO <line number expression>

Line numbers are not necessary in QDOS BASIC. However if you want it to be possible to jump to a particular line of code, then you must use a (unique) line number on that line, so that the GOTO statement can find it.

The most common use is to allow a program to run repeatedly for ever. For example:

```
LET A = 1
10 PRINT A
LET A = A + 1
GOTO 10
```

This program sets variable A to 1, then prints it, then adds 1 to it. Finally the GOTO 10 statement causes the program to jump to line 10. So it just prints, adds 1, and repeats ad infinitum. Running the program prints the numbers to the terminal:

```
1
2
3
```

It just keeps on scrolling by, incrementing by one each time, until you halt the program.

4.8.4 Subroutines: GOSUB and RETURN

The GOSUB..RETURN structure is useful if you have a particular block of code, which we call a subroutine, that you want to execute more than once in your program. Adding this kind of program structure can be good practice in large programs as it keeps your program compact, efficient, neat and readable.

Syntax:

```
GOSUB <line number expression>
RETURN
```

GOSUB causes program execution to jump the subroutine at the specified line number. When the computer encounters the RETURN statement, it jumps back to the line following the one in was called from.

Line numbers are not necessary in QDOS BASIC. However if you want it to be possible to call a subroutine located at a particular line of code, then you must use a (unique) line number on that line, so that the GOTO statement can find it.

For example:

```
LET A = 1
GOSUB 100
LET A = 2
GOSUB 100
END
100 PRINT A + 4
RETURN
```

Running this program sets variable A to 1, then calls the subroutine at line 100; then it sets variable A to 2 and calls the subroutine again. Finally the END statement stops execution. The result printed on the terminal is:

```
5
6
```

Note that you can also make nested calls to other subroutines from within subroutines. But only 4 levels of nested subroutine calls are possible.

4.8.5 END

Ends the program.

Syntax:

```
END
```

For example:

```
LET A = 1
END
```

The first line is executed, which sets variable A to 1. Then the program is ended.

Note that if the computer runs out of program lines to execute, it ends automatically. This has already been seen in the examples in the previous section. Therefore END is not necessary as the last line in a program. But it can still be useful sometimes in the middle of a program if you want to END on a certain condition, or if there are subroutines following after the main program block.

4.8.6 RUN statement

Runs a specified program, then returns control to the calling program. Variables and other system states are not reset, when control returns to the calling program.

Syntax:

RUN <program name string>

Example:

```
RUN "GPS"
```

Runs a program called "GPS". If the specified program file does not exist, or if the program is not compiled, a run-time error is generated.

<program name string> can include a directory path specification if the program file is located in a sub-directory; no directory path specification means the root directory.

Note that program RUNs can also be nested: the called program can also call out to other programs. However, there is a nesting depth limit of 4.

4.8.7 REM (comments)

When a line begins with REM the rest of the line is ignored. This is useful for entering comments.

4.9   Statements

4.9.1 PRINT statement

The PRINT statement outputs one or more strings or expressions to the terminal.

Syntax:

PRINT <string>
PRINT <expression>
PRINT <expression>, <expression>, <expression>
PRINT <expression>; <expression>; <expression>

The items in a PRINT statement can be either strings (in which the #-tags if present, are evaluated) or numeric expressions, or a mixture thereof.

You can print multiple values on one line, by using a comma or semicolon to separate values. A comma separates each printed value with a space character, whereas a semicolon concatenates values together without any spaces.

Examples:

```
LET A = 5
PRINT A
```

Outputs the text "5" to the terminal.

```
LET A = 5
LET B = 7
PRINT A, B
PRINT B; B
PRINT A + B
```

Results in the following text on the terminal:

```
5 7
57
12
```

The PRINT statement can also be used directly in the Command line application, as well as in a BASIC program.

4.9.2 DELAY statement

Waits a specified number of milliseconds before continuing program execution.

Syntax:

DELAY <milliseconds expression>

Example:

```
DELAY 100
```

Waits 100 milliseconds (0.1 seconds).

4.9.3 SLEEP statement

Causes QDOS to sleep for a specified time, then wake up and resume execution at the next program line.

Syntax:

SLEEP <interval duration expression> <start minute expression>

The SLEEP statement has two parameters, which determine the time to wake up. Both are expressed as a number of minutes. The first parameter is an interval duration parameter. The second parameter specifies how many minutes past the hour to start the interval.

This is best illustrated by an example:

```
SLEEP 10 4
```

The two parameters here are 10 minutes and 4 minutes. This statement sets wakeup times every 10 minutes, starting at 4 minutes past the hour. The wakeup times will therefore be :04, :14, :24, :34, :44 and :54 minutes past each hour.

When the command is executed, the computer calculates the next wakeup time after the current time. For example, if the current time is 10:58:32 when this program line is executed, the computer will sleep until 11:04:00.

The SLEEP command is used when you need to start the following program line at a precise time. But it is also useful because when the computer is sleeping, power consumption is low.

4.9.4 GPS statement

The GPS statement switches on the GPS receiver module. It waits for a satellite lock to be obtained, then sets the computer's real time clock, and other parameters such as the altitude, latitude, longitude etc. The GPS statement also calibrates the oscillation value of the 8MHz system clock oscillator and PLL reference frequency.

Syntax:

> GPS <timeout expression>
> GPS <timeout expression> <wait expression>
> GPS <timeout expression> <command string>
> GPS <timeout expression> <wait expression> <command string>
> GPS <command string>

There are several ways to use the command. The typical usage is GPS with a single parameter, the timeout expression. This specifies the number of seconds to wait for a GPS 3D satellite lock, before giving up and proceeding with the next line of the program. This command switches on the GPS, waits for satellite lock, sets real time clock and other GPS parameters such as altitude, latitude, longitude; before finally powering down the GPS.

The "wait expression" is a way to cause the GPS module to remain active for a defined number of seconds once a 3D fix is obtained, before proceeding through the rest of the program. When this parameter is not specified, the default is 1. When specified, typically with a number larger than 1 (to ensure that all GPS NMEA sentences are received and parsed), the system will wait the specified number of seconds. This can improve the accuracy of the position fix.

A command string may also be passed to the GPS to alter its behavior. Command strings use a checksum as the last two characters, and there are online tools for calculating this parameter. When a command string is specified, the system waits 2 seconds for a response from the GPS; if none is received (which is typically the case if an unsupported command was given), it proceeds with the rest of the GPS operation.

The final way to use the GPS command is by passing a command string only. This differs in so far as the system waits 2 seconds for a response from the GPS, then abandons the command if no response is received; the important point is that the GPS is NOT powered down; this means that the effects of the command are retained, because the GPS still has power applied. This is essential if you wish to configure the GPS with more than one command, because otherwise they would be forgotten immediately when the GPS is powered down.

Examples:

```
GPS 300
```

Powers on the GPS, waits for satellite lock; sets GPS variables and does calibration. If the procedure is not completed in 300 seconds, the command is abandoned. At the end of the command (or when it is abandoned), the GPS is powered down.

```
GPS 300 10
```

Same as the previous example, but now it will wait an additional 10 seconds for the position fix accuracy to potentially be improved. Note that if the timeout (here 300 seconds) occurs before the 10 seconds have elapsed, the wait is abandoned and the GPS function returns (successful fix).

```
GPS 300 "$PMTK605*31"
```

This does the same thing, but first sends the command $PMTK605*31 to the GPS and waits up to 2 seconds for an answer.

```
GPS "$PMTK605*31"
```

sends the command to the GPS, and waits up to 2 seconds for an answer, but does NOT then power down the GPS.

4.9.5 Transmit statements: CW, HELL, JT9, JT65, WSPR, TELE, TELEC

The family of transmit statements causes the U4B to transmit on radio frequencies. The statements requires a number of parameters, which specify the transmission mode, sub-mode, frequency, and parameters specific to that transmission mode.

The following sections explain each transmission mode and the relevant parameters.

CW statement

Transmit a message using CW, FSKCW, QRSS or DFCW modes

Syntax:

CW <sub mode> <frequency> <speed> <FSK> <Message string>

The <sub mode> parameter expects values as follows:

sub mode	Explanation
0	Plain full-speed CW
1	QRSS: slow speed CW (on/off keyed)
2	FSKCW: slow speed frequency shifted CW (shift on key-down)
3	DFCW: dual frequency slow CW (dit and dah on different frequency shifted tones)

<frequency> is the transmission frequency in Hz.

<speed> is the standard Words Per Minute speed for CW sub mode; or for the other slow-speed modes is the number of seconds per "dit".

<FSK> is the frequency shift specification for FSKCW and DFCW sub modes; it is ignored for CW and QRSS sub modes.

The Message is the text to transmit, and may contain #-tags as explained previously. The maximum message string length is 64 characters. The actual transmitted message may be more than this once the dynamic #-tags are substituted.

Example:

```
CW 0 10106000 12 0 "AB2CD"
```

Transmits 12wpm CW on 10.106MHz with message "AB2CD".

CW transmissions can also contain customized frequency shifts which make it possible to transmit patterns. A customized section of message starts and ends with an asterix character. The first two digits represent the duration of each symbol, and subsequent characters represent the upward shift from the transmit frequency baseline, 1Hz per unit e.g. 1 means +1Hz, 2 means +2Hz, A means +10Hz etc up to a maximum of Z for +36Hz shift.
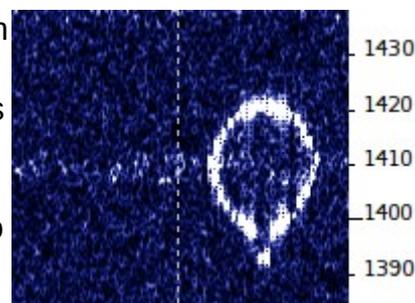
Example:

```
CW 0 14096920 5 0 "*01IFLCNAP8Q7R65S0S0S5R6R7Q8PANCLFI*"
```

Transmits the Balloon glyph pattern shown in the photograph.

HELL statement

Transmits the Hellshreiber family of sub-modes. The format of the HELL statement is

HELL <sub mode> <frequency> <speed> <FSK> <message>

The <sub mode> parameter expects values as follows:

Sub mode	Explanation
0	Slow-speed MFSK Hell (narrowband mode)
1	Standard full-speed Hellschreiber
2	Standard full-speed "DX" Hellschreiber

<frequency> is the transmission frequency in Hz.

<speed> is applicable only to the slow-speed MFSK narrowband sub mode. It is the number of seconds required to transmit one whole Hell character. Example settings are 42 (means 1 second per pixel) or 63 (1.5 seconds per pixel). The standard full-speed Hell sub modes have a fixed transmission baud rate, so the Speed parameter is ignored for these sub modes.

<FSK> is applicable only to the slow-speed MFSK narrowband sub mode. It is the height of 5 rows of pixels of the transmitted character, in Hz. This parameter is ignored for full-speed Hell sub modes which use on/off keying without any frequency shift.

<message> is the text to transmit, and may contain #-tags as explained above. The maximum message string length is 64 characters. The actual transmitted message may be more than this once the dynamic #-tags are substituted.

Example:

```
HELL 1 10138000 0 0 "AB2CD"
```

Transmits standard Hellschreiber on 10.138MHz with message "AB2CD".

JT9 statement

Transmits one of the JT9 family of sub-modes. The format of the JT9 statement is

> JT9 <sub mode> <frequency> <message>

The <sub mode> parameter defines the variant of JT9, and expects values as follows (sub mode is actually the number of minutes duration of the transmission):

Sub mode	Explanation
1	JT9-1 (most commonly used)
2	JT9-2
5	JT9-5
10	JT9-10
30	JT9-30

<frequency> is the transmission frequency in Hz.

<message> is the text to transmit, and may contain #-tags as explained above. JT9 messages are a maximum of 13 characters. Remember that this is the message length AFTER any dynamic #-tag substitutions have taken place. Furthermore, substituted characters must follow the rules of the JT9 protocol, concerning which characters are permissible in messages.

Example:

```
JT9 1 10145000 "AB2CD"
```

Transmits JT9-1 on 10.145MHz with message "AB2CD".

Note that a JT9 transmission command will automatically wait for the appropriate time to start transmitting, according to the JT9 protocol.

JT65 statement

Transmits the JT65 family of sub-modes. The format of the JT65 statement is

> JT65 <sub mode> <frequency> <message>

The <sub mode> parameter defines the variant of JT65, and expects values as follows:

Sub mode	Explanation
0	JT65-A
1	JT65-B
2	JT65-C

<frequency> is the transmission frequency in Hz.

<message> is the text to transmit, and may contain #-tags as explained above. JT65 messages are a maximum of 13 characters. Remember that this is the message length AFTER any dynamic #-tag substitutions have taken place. Furthermore, substituted characters must follow the rules of the JT65 protocol, concerning which characters are permissible in messages.

Example:

```
JT65 0 10145000 "AB2CD"
```

Transmits JT65-A on 10.145MHz with message "AB2CD".

Note that a JT65 transmission command will automatically wait for the appropriate time to start transmitting, according to the JT65 protocol.

WSPR statement

Transmits the WSPR family of sub-modes. The format of the WSPR statement is

WSPR <sub mode> <frequency> <power> <callsign>

The <sub mode> parameter defines the variant of WSPR, and expects values as follows:

Sub mode	Explanation
2	Standard WSPR (a.k.a. WSPR-2)
4	Extended mode WSPR, sends two WSPR messages, and 6-char subsquare etc
15	15-minutes SLOW WSPR (for use on LF)

<frequency> is the transmission frequency in Hz.

<power> is the transmitter power to be encoded into the WSPR protocol, expressed in dBm.

<callsign> is the callsign to be encoded into the WSPR protocol. It must comply with the rules of the WSPR protocol. When using the extended WSPR mode (sub mode 4) the callsign can also have a prefix or suffix, which must comply with the rules of the extended WSPR protocol.

Note that the Maidenhead Locator Square (or subsquare in the case of sub mode 4, extended WSPR) is automatically taken from the data extracted from the serial data stream of the GPS.

Where Extended WSPR is used (sub mode 4) the system automatically sends the two WSPR transmissions consecutively, with the appropriate encoding. You do not need to include two separate statements on different lines of BASIC. The two transmissions are sent automatically.

Example:

```
WSPR 2 10140200 10 "AB2CD"
```

Transmits WSPR on 10.1402MHz with power +10dBm and callsign "AB2CD".

Note that a WSPR transmission command will automatically wait for the appropriate time to start transmitting, according to the WSPR protocol; WSPR transmissions for standard 2 minute WSPR, for example, always commence on the 2nd second of even minutes.

TELE statement

A special case transmission statement which transmits WSPR and Telemetry using the QRP Labs telemetry-over-WSPR protocol messages. This consists of from 2 WSPR transmissions. It sends position, altitude, ground speed, battery voltage, temperature and GPS status.

The advantage of using TELE for data transfer, compared to other transmission modes, is that it automatically displays a live map of the balloon path on the QRP labs website (or other tracking websites), and your additional telemetry is also automatically collected and available for download.

TELE automatically transmits using the callsign, band, frequency, channel and timing schedule defined in the Configuration application. Only the callsign and band are user configurable. The frequency and timing schedule are determined by the channel, which is assigned uniquely to each U4B. In the configuration screen you have a choice of 3 channels (you can also set an arbitrary channel number using the CH variable). The transmission cycle repeats every 10 minutes.

The WSPR transmissions are as follows. Each of the two transmissions takes 1 minute 52 seconds and the system waits automatically for the correct interval between transmissions.

1	Standard WSPR transmission with Callsign and Frequency defined in the Configuration application
2	Telemetry transmission containing 5th and 6th character of Maidenhead grid subsquare, altitude, ground speed, battery voltage, temperature and satellite status

Syntax:

> TELE
> TELE <BigNumber>

The basic usage is simply TELE, which transmits two WSPR messages, 1 and 2 in the table above. These are automatically collected and plotted on a live tracking map on the QRP Labs website or other tracking website.

Advanced usage:

However, another way to use the TELE command is with a numeric parameter "BigNumber". This is a number between 0 and 189,734,399 (Larger numbers are processed modulus 189,734,400 to prevent overflow). This BigNumber is equivalent to just under 27.5 binary bits. You might wish to use this feature, for example, to send different data to the standard telemetry data, in the event that no GPS fix is obtained.

The principle behind this feature is that the standard telemetry transmission does not use all the available data payload space in the WSPR transmission. The WSPR transmission encodes two numbers n and m (see http://www.g4jnt.com/wspr_coding_process.pdf); n contains the packed callsign, m contains the packed locator and power fields. Given that the 1st and 3rd callsign characters in the U4B protocol are used for channelization, the remaining space available for the data payload in n and m is 632,736 and 615,600 numbers respectively.

In the U4B protocol standard telemetry, 5th and 6th Maidenhead subsquare characters and altitude are encoded in to n; groundspeed, battery voltage, temperature and GPS status are encoded into m. However, only 615,168 of the available 632,736 values in n are used, leaving 17,568 unused number combinations. Similarly only 604,800 of the available 615,600 values in m are used, leaving 10,800 unused number combinations.

The product of the two unused number sets is 17,568 x 10,800 = 189,734,400.

When you call the TELE command with an expression parameter, BigNumber, this BigNumber is split up into two pieces; one is sent in the unused space of n, and one in the unused space of m.

The resulting telemetry transmission in slot 1 (after the standard callsign transmission) is NOT standard telemetry, and does not interfere with, nor can it be confused with, standard telemetry; it contains this number in the range 0 to 189,734,399.

TELEC statement

This statement transmits a single Custom Telemetry transmission which may be decoded by WSPRtv or Traquito websites, see https://wsprtv.com (click in bottom right corner for help) and https://traquito.github.io/search/spots/dashboard respectively.

This additional telemetry was originally called "User Telemetry" in early versions of the U4B operating manual, then Extended Telemetry was proposed and agreed by the ccommunity on the PicoBalloons groups.io forum in December 2024. Subsequently this was superseded by an agreement on the picoballoons forum in March 2024, by the term "Custom Telemetry" of which the former "Extended Telemetry" is a subset. WSPRtv displays both Custom Telemetry and its subset Extended Telemetry, whereas the Traquito tracking website displays only Extended Telemetry.

WSPRtv Custom Telemetry is decoumented here: https://wsprtv.com/docs/user_guide.html Extended Telemetry is documented here: https://traquito.github.io/pro/telemetry/extended/

The TELEC statement replaces the TELEX statement, and the user telemetry form of TELE in earlier U4B firmware releases.

Flight operators are encouraged to join the picoballoons groups.io forum for lots of information on all aspects of picoballooning including tracking, telemetry, antennas, balloons etc.: https://groups.io/g/picoballoon; a good example of the use (first use actually) of the former TELEX statement is documented here: https://groups.io/g/picoballoon/topic/112539224

The TELEC statement has the following syntax:

TELEC <BigNumber expression>
TELEC <String>
TELEC <HdrRESERVED expression> <HdrType expression> <BigNumber expression>

When called with a single BigNumber argument, TELEC sends a single transmission of Custom Telemetry. BigNumber must be in the range 0 to 389,512,281,600 which is the 35.2 bit data payload capacity of a Custom Telemetry message. This number may be decoded on the WSPRtv website https://wsprtv.com. (click bottom right corner for help).

TELEC may also be called with a 6-character string argument, which is converted into a BigNumber.

When called with three arguments, TELEC sends a single transmission in the Traquito Extended Telemetry format. HdrRESERVED is a number in the range 0 to 3. Currently only ET0 (HdrRESERVED = 0) is supported. HdrType is a number in the range 0 to 15 specifying the telemetry type, see https://traquito.github.io/pro/telemetry/extended/#hdrtype. BigNumber must be in the range 0 to 608,612,940 which is the 29.2 bit data payload capacity of the Traquito Extended Telemetry format. ET0 may be decoded on the Traquito website

https://traquito.github.io/search/spots/dashboard, as well as on the WSPRtv website https://wsprtv.com.

As per the specification, a Custom Telemetry transmission may be sent in any transmission slot in the 10 minute cycle, or in multiple slots or even in ALL five slots (there is no requirement in the protocol, for a preceding standard WSPR transmission). However both Traquito and WSPRtv websites currently only work with one or more Custom Telemetry transmissions which follow the standard WSPR Callsign and standard telemetry messages sent by the TELE statement.

Example using PACK, ERLON, ERLAT and TELEC to send Custom Telemetry.

The following example was used during testing. Firstly, a custom message definition is shown below, using the advanced definition page on the WSPRtv website:
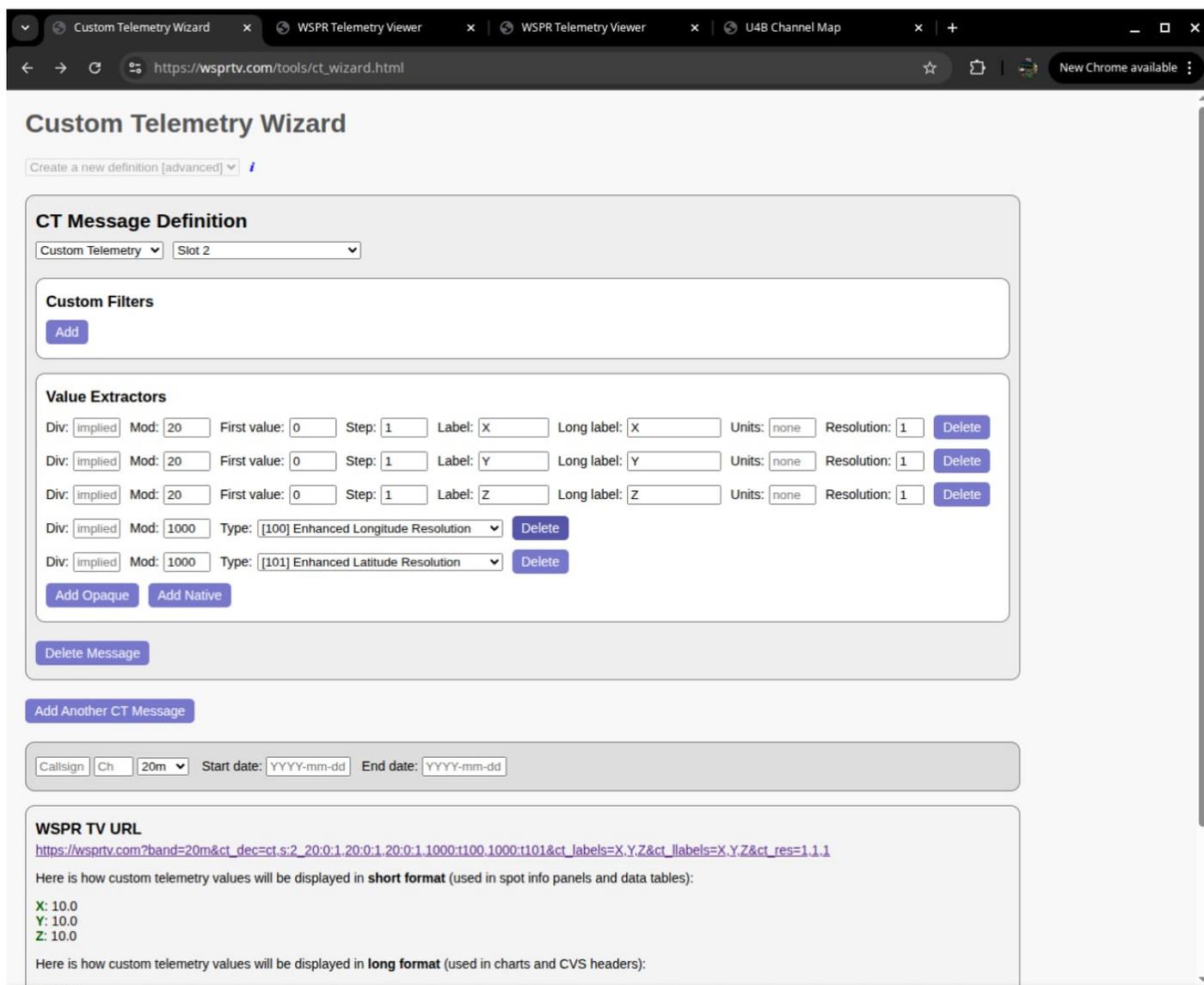
According to this specification, five data fields are going to be packed into the BigNumber sent using Custom Telemetry. In order from least significant to most significant parts of BigNumber, these are:

- X: data having 20 possible values starting at 0 and in steps of 1.
- Y: data having 20 possible values starting at 0 and in steps of 1.
- Z: data having 20 possible values starting at 0 and in steps of 1.

- Native type 100: the Enhanced Longitude Resolution with modulus 1000
- Native type 101: the Enhanced Latitude Resolution with modulus 1000

The enhanced resolution longitude and latitude divide by the Maidenhead grid subsquare into a 1000 x 1000 grid of tiny squares. One million squares in a Maidenhead grid subsquare. This gives a resolution of only a couple of meters. It's overkill! But remember: one of the ways native types such as the enhanced position resolution types 100 and 101 can be used, is simply to put unused data payload space to some good purpose. In my case, I have a highly artificial example with three small data values, each only using 4-point-something bits. So using just under 10 bits of payload for each of the enhanced resolution types is a nice way to use up the unused bitspace – and most importantly would let me test very well since I know where my house is.

This is the test program I used for this:

```
LET HP = 1
10 GPS 300
LET X = X + 1
IF X = 4
    LET X = 0
ENDIF
LET Y = Y + 1
IF Y = 7
    LET Y = 0
ENDIF
LET Z = Z + 1
LET T = ERLAT 1000
LET N = ERLON 1000
LET B = 0
LET B = PACK B 1000 T
LET B = PACK B 1000 N
LET B = PACK B 20 Z
LET B = PACK B 20 Y
LET B = PACK B 20 X
TELE
TELEC B
GOTO 10
TELEC - 22 lines (5%)
Compile successful! 146 bytes written
```

Program explanation:

First I switch on High Power mode. Then call the GPS statement to get the current position data. I increment each of X, Y, Z variables. Some logic resets X when it reaches 4, and resets Y when it reaches 7. Meanwhile Z has no reset logic, which means it will just continue incrementing by one every 10 minute cycle; but because we defined the Z data field in WSPRtv to have only 20 values, and will call the PACK function accordingly, the PACK function automatically applies a modulus 20 to the input Z value, avoiding any overflow between data fields.

Next,

```
LET T = ERLAT 1000
```

puts the Enhanced Resolution value of Latitude into variable T. This has 1000 possible values (the argument of ERLAT).

Next,

```
LET N = ERLON 1000
```

puts the Enhanced Resolution value of Longitude into variable N. This has 1000 possible values (the argument of ERLON).

The next task is to build the BigNumber. I used variable B for this. The first step is to set B to zero, then start adding the data fields into it. The data fields must be added in REVERSE order to how they are defined in WSPRtv. The definition in WSPRtv is from least significant to most significant bits in BigNumber; here we pack in the opposite direction, largest first. Apart from that, it's easy to see that the number is built up in the same (but reverse) order and modulus specification as on the WSPRtv value specifier declaration screen:

```
LET B = 0
LET B = PACK B 1000 T
LET B = PACK B 1000 N
LET B = PACK B 20 Z
LET B = PACK B 20 Y
LET B = PACK B 20 Z
```

The final steps are just

```
TELE
```

```
TELEC B
```

which transmit the standard WSPR (Callsign/Locator/Power), Standard Telemetry (5th and 6th Maidenhead grid subsquare, Altitude, Groundspeed. Battery voltage, Temperature and GPS status), and finally the 3rd WSPR transmission which is the Custom Telemetry BigNumber, variable B. Lastly of course, GOTO 10 goes back to near the beginning so the cycle repeats.

4.9.6 OUT statement

Sets a GPIO pin state.

Syntax:

OUT <GPIO pin expression> <Value expression>

There are 19 GPIO pins numbered 0 to 18. Additionally there is an special purpose IO pin 19, which is connected to the onboard LED on the U4B. If you specify a pin number higher than 19, the OUT command has no effect.

Example:

```
OUT 5 1
```

Sets the value of GPIO pin 5, to 1 (high, means 3.3V).

```
OUT 3 0
```

Sets the value of GPIO pin 3, to 0 (low, means 0.0V).

```
OUT 19 CL
```

Sets the LED state to the value of CL. CL is a global variable which is 1 if the last GPS statement resulted in a successfully completed calibration. So this example switches on the LED if the GPS calibration was successful.

4.9.7 I2CW, I2CW16 statements

Writes to a device attached to the I2C bus. Some devices have 8-bit registers (such as the Si5351A and LM75 chips), others have a much larger register space and a 16-bit address format (such as the I2C serial EEPROM chip). I2CW is for 8-bit devices. I2CW16 is for 16-bit devices.

Syntax:

> I2CW <device address expression> <register address expression> <value expression>
> I2CW16 <device address expression> <register address expression> <value expression>

Both addresses are expressed as decimal values and are expressions. The value expression is sent as an 8-bit value (1 byte) which is decimal 0 to 255. Larger numbers are sent modulus 256. Whilst I2CR and I2CR16 expressions can read multi-byte data values from an I2C device, there is no provision for multi-byte transfer in the Write commands.

The U4B has three existing devices attached to its I2C bus. You may add additional I2C sensors. The additional device addresses must not conflict with the existing three devices on the bus.

Existing U4B I2C devices:

Address	Device
160	EEPROM, lower half of 128KByte chip, 16-bit register address
162	EEPROM, upper half of 128KByte chip, 16-bit register address
192	Si5351A (Synthesizer chip), 8-bit register address
144	LM75 (Temperature sensor), 8-bit register address

It should be noted that the I2C bus address is often specified as a 7-bit value; this 7-bit value is put in the most significant bits of a byte, and the least significant bit is set to 0 for a Write operation or 1 for a Read operation. The <device address> parameter expected by the I2CR/I2CR16 functions is an 8-bit value which is really the Write operation 8-bit address. Therefore if your device specifies a 7-bit I2C address, you should multiply this by 2 for use in these functions.

Examples:

```
I2CW 129 5 48
```

Writes value 48 to register 5 of I2C device at address 129.

```
I2CW16 I2CR16 164 43846 28
```

Writes value 28 to register 43846 of device 164.

You should in general not be tempted to write to the onboard I2C peripherals, particularly not the EEPROM which can corrupt the file system.

The companion functions used in expressions, are I2CR and I2CR16 for reading from I2C devices, described earlier.

4.9.8 File handling statements: FILE, FILEWR, FILERD, FILEDEL

FILE statement

Open a file for reading or writing.

Syntax:

>   FILE <file number expression> <access mode string> <file name string>

<file number expression> specifies a file number to assign to this opened file. Valid file numbers in U4B are 1 and 2 only. Any other file number will result in a runtime error.

<access mode string> must be one of the following values:

Mode value	Action
"R"	Open the file for reading. If the file doesn't exist, a runtime error will occur.
"W"	Open the file for writing. If there is an existing file with this name, it is erased.
"A"	Open the file for writing, appending to the end of the file. If there is no existing file with this name, one is created. If there is an existing file, it is opened and any new data is written at the end of the file.

When a file that is opened for reading, cannot be written to using the FILEWR function. Correspondingly, a file opened for writing, cannot be read from using the FILERD function.

It is not necessary to explicitly close a file. When the program is ended, the file is closed automatically. Or if you issue a new "FILE" with the same file number, any previous file opened at that file number, is closed automatically.

Example:

```
FILE 1 "A" "LOGFILE"
```

Opens a file called "LOGFILE" for writing (appending to the end of the file), and sets it as file number 1.

```
FILE 2 "R" DATA2"
```

Opens a file called "DATA2" for reading.

FILEWR statement

Writes to a file.

Syntax:

>   FILEWR <file number expression> <data expression>, <data expression>, …

One or more data expressions can be written (appended) to the file and are comma separated. A data expression can be a string or a numeric expression. There is no comma after the <file number expression> parameter. Each FILEWR command writes a line to the file, and the line is terminated with a linefeed /n character (ASCII 10).

A file must have been correctly opened for writing prior to executing the FILEWR command, or it will fail with a runtime error.

Examples:

```
FILE 1 "W" "EXAMPLE"
FILEWR 1 "HELLO"
```

opens a file "EXAMPLE" for writing as file number 1, and writes "HELLO" to it.

```
FILE 2 "W" "DATA3"
LET A = 1
LET B = 2
LET C = 3
FILEWR 2 A, B, C
```

opens a file named "DATA3" for writing as file number 2, and writes a line to it: "1,2,3"

FILERD statement

Reads from a file.

Syntax:

FILERD <file number expression> <variable>, <variable>, …

One or more variable names can be supplied and the system will attempt to read comma separated data from the file, convert it to numbers, and set the corresponding variables. There is no comma after the <file number expression> parameter.

A file must have been correctly opened for reading prior to executing the FILEWR command, or it will fail with a runtime error.

Example:

```
FILE 1 "R" "DATA3"
FILERD 1 D, E, F
PRINT D, E, F
```

When the file DATA3 was created as per the FILEWR example in the previous section, this example will open file "DATA3" for reading, and read its comma delimited contents into variables D, E and F. The print statement will print these values to the terminal, appearing as:

```
1 2 3
```

Note that if your program needs to read all the data lines from a file in a loop, you can detect the end of the file using the FILEEOF function described in a previous section. For example, if "DATA3" contains multiple lines:

```
FILE 1 "R" "DATA3"
LET C = 0
10 IF FILEEOF 1 = 0
    LET C = C + 1
    FILERD 1 D, E, F
    PRINT D, E, F
```

```
        GOTO 10
ELSE
        PRINT "FINISHED! READ #VC ROWS!"
ENDIF
```

This example opens file "DATA3" for reading. It then loops through, reading data lines from the file, and parsing them into variables D, E and F, which it prints to the screen. It also keeps a line counter in variable C. While it has not reached the end of the file, the program loops back via GOTO 10 and repeats the read. When the end of the file is reached, the program prints "FINISHED! READ 6 ROWS!" (for example, if there were 6 lines in the file.

You can also access the first 20 characters of a file line using the #FL tag in a string. For example, if file DATA3 contains a single line "1,2,3" as in our above example:

```
FILE 1 "R" "DATA3"
FILERD 1
PRINT "#FL"
```

will display on the terminal

```
1,2,3
```

You can then use VAL "#CS" if you want to parse characters from the string, convert them to numbers etc.

FILEDEL statement

Deletes a file entirely from the file system.

Syntax:

> FILEDEL <filename string>

Example:

```
FILEDEL "TEST"
```

deletes a file named "TEST" from the file system.

If the file named "TEST" does not exist, the command has no effect.

4.9.9 COUNTER statement

Sets up one of two 16-bit counters.

Syntax:

> COUNTER <counter number>
> COUNTER <counter number> <mode>
> COUNTER <counter number> <mode> <polarity>
> COUNTER <counter number> <mode> <polarity> <filter>

U4B contains two 16-bit counters which can be enabled and connected to GPIO pins 0 and 6, respectively. These counters can be enabled in addition to any existing use of the GPIO pin that you may have; even if you are using the pin as an output under BASIC control, the counter can still count pulses. The counters are numbered 0 and 1. Both counters can count up, or down. Counter 0 can be triggered on a positive-going or negative-going edge; counter 1 is always triggered on the positive-going pulse edge. Both counters can have a filter enabled, whose value is from 0 to 15; a pulse must be consistently applied for the duration of this number of system clock cycles.

Parameter explanation:

<counter number> is 0 or 1. There are two counters available, this parameter specifies which one is being enabled by this statement.

<mode> is 0 for an up-counter, and 1 for a down-counter. When the limit is reached - 65,535 for an up-counter, 0 for a down-counter - the counter value rolls over to 0 or 65,536 respectively.

<polarity> is 0 for positive-going (leading, rising) pulse edge; 1 for the negative going (falling) pulse edge. Note that this parameter is only used on counter #0, it has no effect on counter #1.

<filter> is a value from 0 to 15, which specifies how many clock cycles to check the input for, before allowing it to trigger the counter.

The mode, polarity and filter parameters are optional; one of the four above syntax forms must be used. When not specified, the default values are zero. The counter number must always be specified.

Example:

```
COUNTER 0
```

enables counter 0, which means that counter 0 is clocked in up-counter mode by positive-going, on GPIO pin 0 without filtering.

```
COUNTER 0 1 1 3
```

enables counter 0, in down-counter mode with negative-going pulses (falling edges) and filter value 3.

The counter value is available for retrieval, and for setting, using the C0 and C1 parameters for counters #0 and #1 respectively. C0 and C1 can be used in expressions just like any other variable, and you can set the counters to zero or any desired value using LET, for example:

```
LET C0 = 0
```

5.   QRP Labs tracking

You can use the U4B tracker however you like. But the easiest way to use it for tracking a balloon flight, is using the TELE statement to transmit the QRP Labs telemetry format, and using the QRP Labs web site to view the live tracking map and download your data. To do this, you need to have registered a customer account on the QRP Labs shop http://shop.qrp-labs.com.

Note that Traquito and WSPRtv http://wsprtv.com offer superior tracking to the QRP Labs tracking map.

Note that it is possible, with limitations, to operate multiple flights with the same callsign, as long as they are on different channels. However, since there are 30 permutations of channel and band, which all share the same first (standard WSPR) transmission time and frequency, you have to ensure that your choice of channels does not cause this conflict. Otherwise reports from your various different balloon flights cannot be differentiated. This is one more reason why there are three channels to choose from. You can use the Configuration screen to check this. The Configuration screen will show, for your choice of band and channel, the frequency and time slot which will be used. You should make sure that no two of your balloon flights have the same Callsign AND band AND frequency AND time slot. In this way, one operator can operate up to 20 active flights with the same callsign, simultaneously.

On logging in to your account you will see a section titled "My Balloons" with a link "Modify your balloon flights".
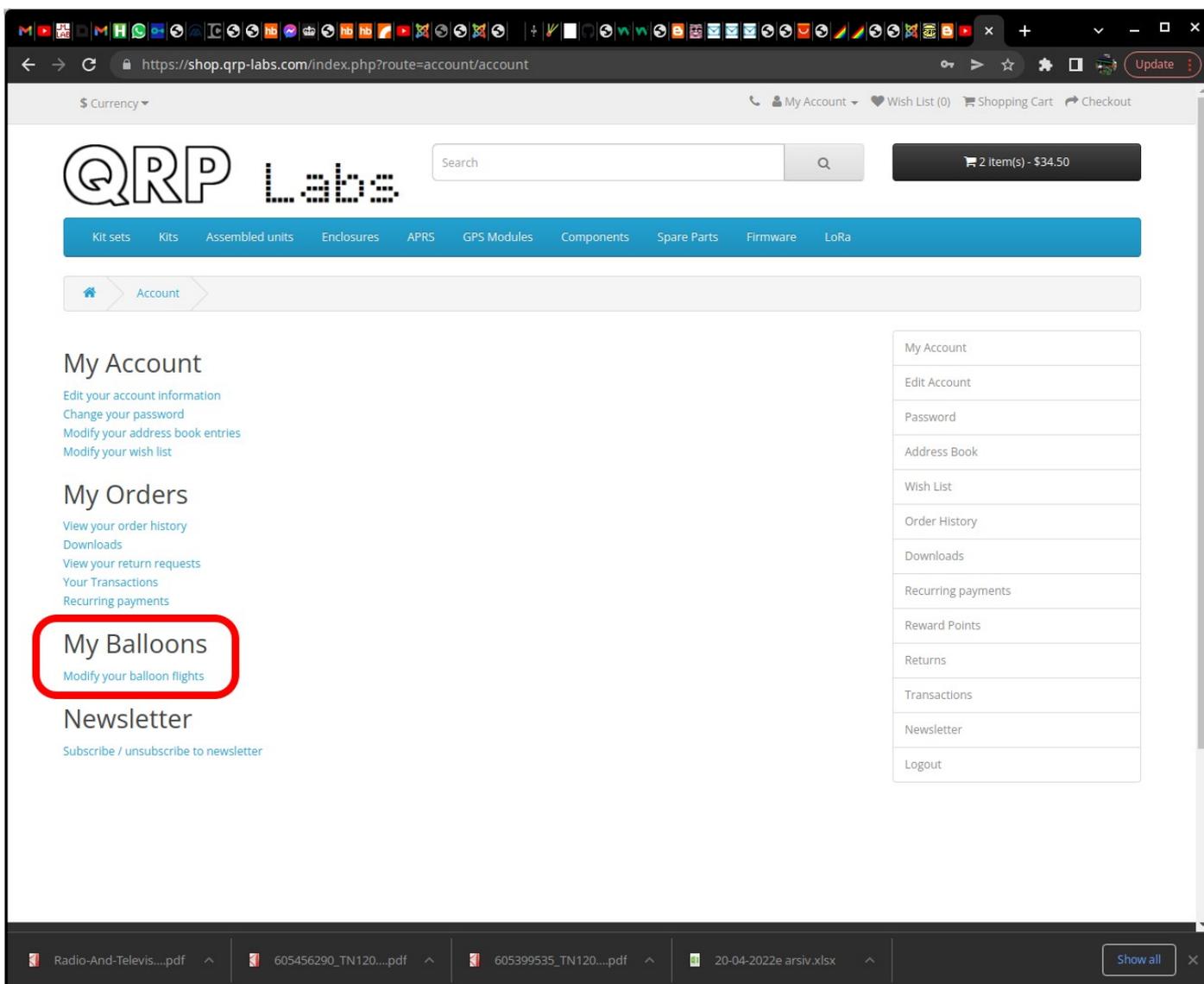
Now if you click that link ("Modify your balloon flights") you will see a listing of all the Balloon flights that you have registered:
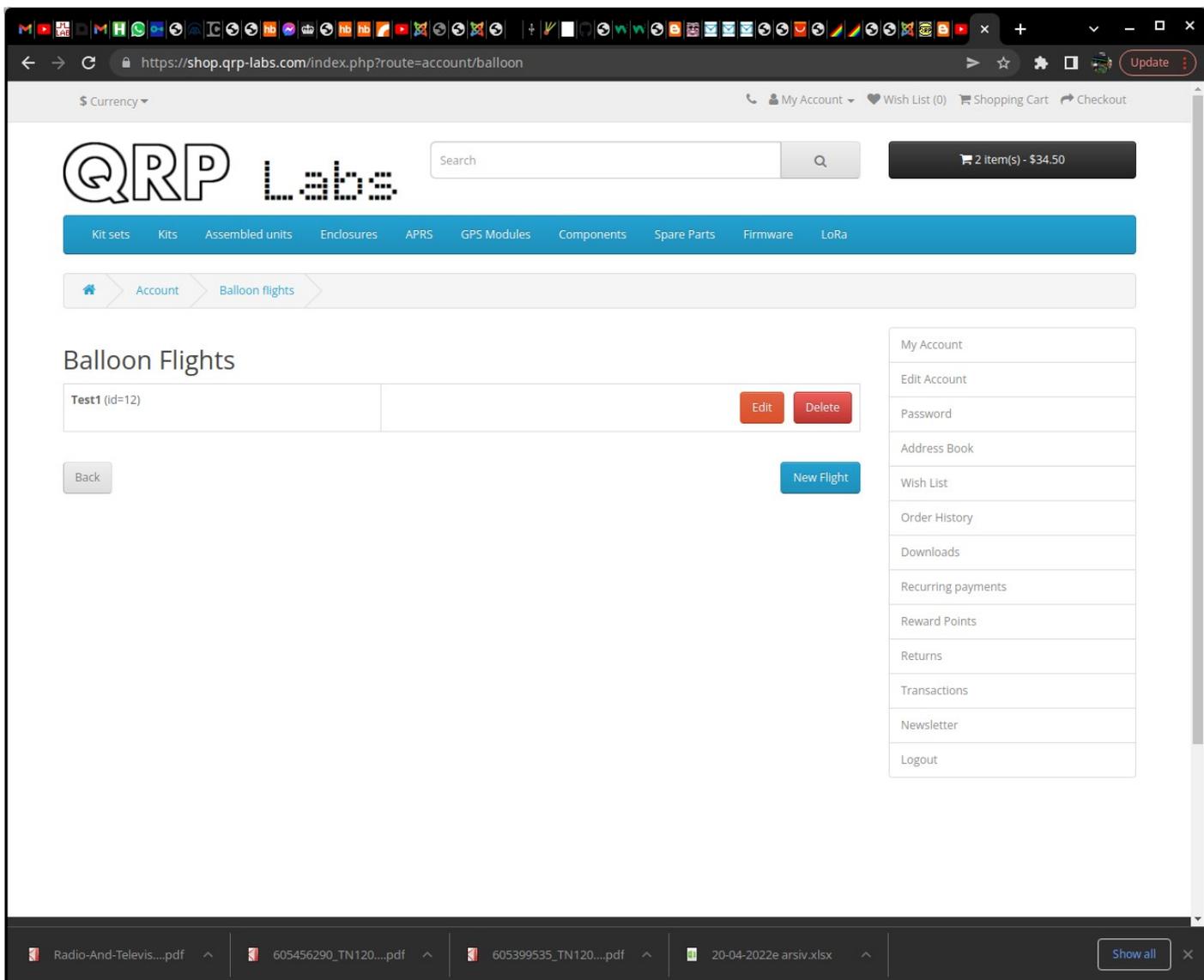
There are three buttons, towards the right.

"New Flight":   Click this button to register a new flight

"Edit":   Click this button, on the flight row of interest, to amend the flight details

"Delete":   Click this button, on the flight row of interest, to delete the flight. Note that this is an irreversible action so don't do it unless you are sure.

Clicking either the "Edit" button or "New Flight" button will take you to the detailed information for that particular flight:
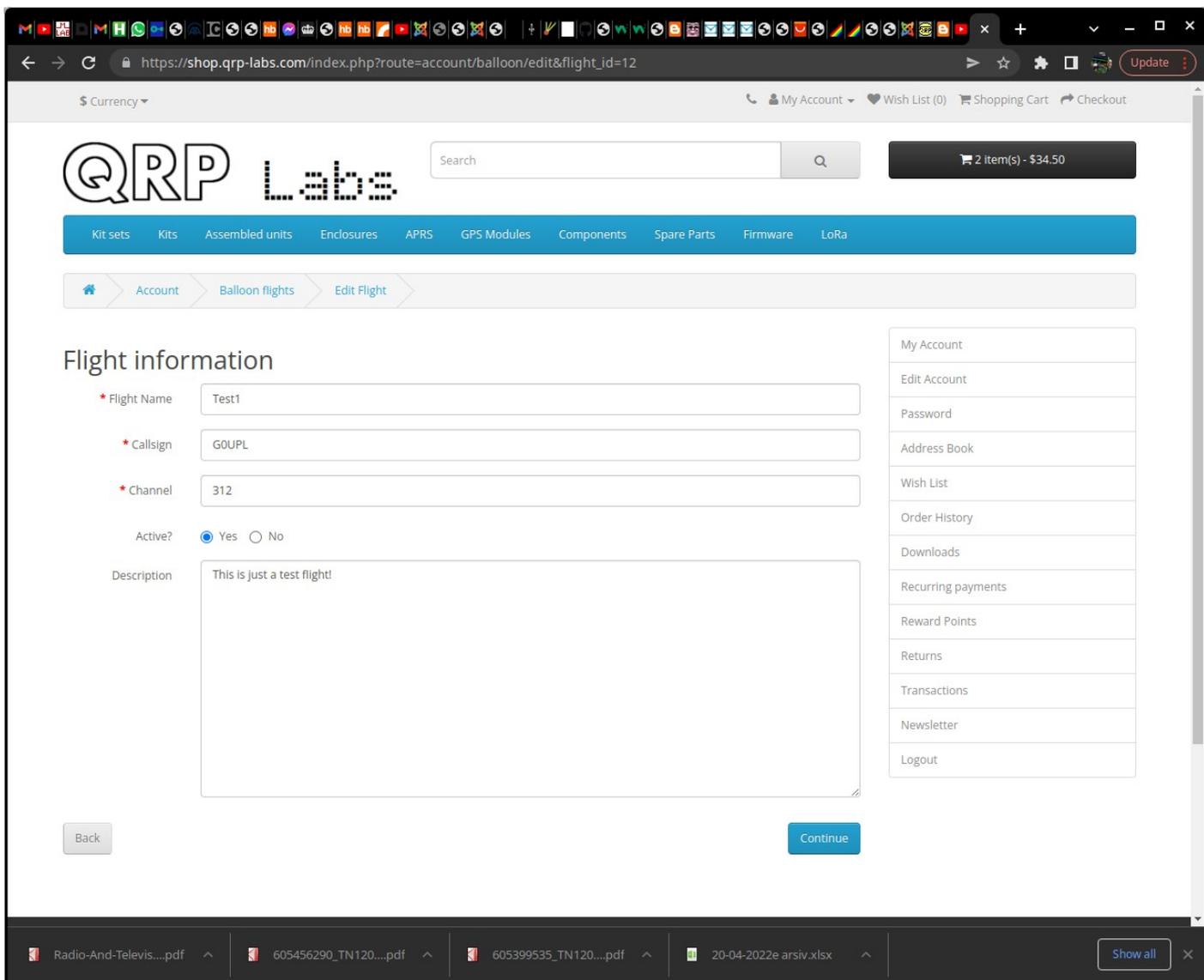
Flight name: The name you wish to use for the flight. To avoid confusion, it will be best to avoid using the same name as other people have used for their flight(s).

Callsign: The callsign that you wish to use for the flight. This must match the callsign you have entered in the Configuration screen of the U4B tracker.

Channel: This is a number from 0 to 599 which specifies the channel of operation for the U4B TELE BASIC statement. This number must match the channel number that you have selected in the Configuration screen of the U4B tracker, or the CH variable if you have set it to your own choice of channel.

Active?: Select Yes if this is an active flight that you wish to track, No if it is not.

Description: Anything you like, to describe your flight; this information will be shown on your own individual flight page.

When you have entered the information for your flight, click the "Continue" button to save it.

Now go to the QRP Labs website page http://www.qrp-labs.com/tracking – here you will see a list of active flights, the callsign and Channel.

NOTE: currently the page only works for http:// so be sure to set this. Most browsers try to use https:// and the pages will appear blank! So you need to manually edit the URL to http://
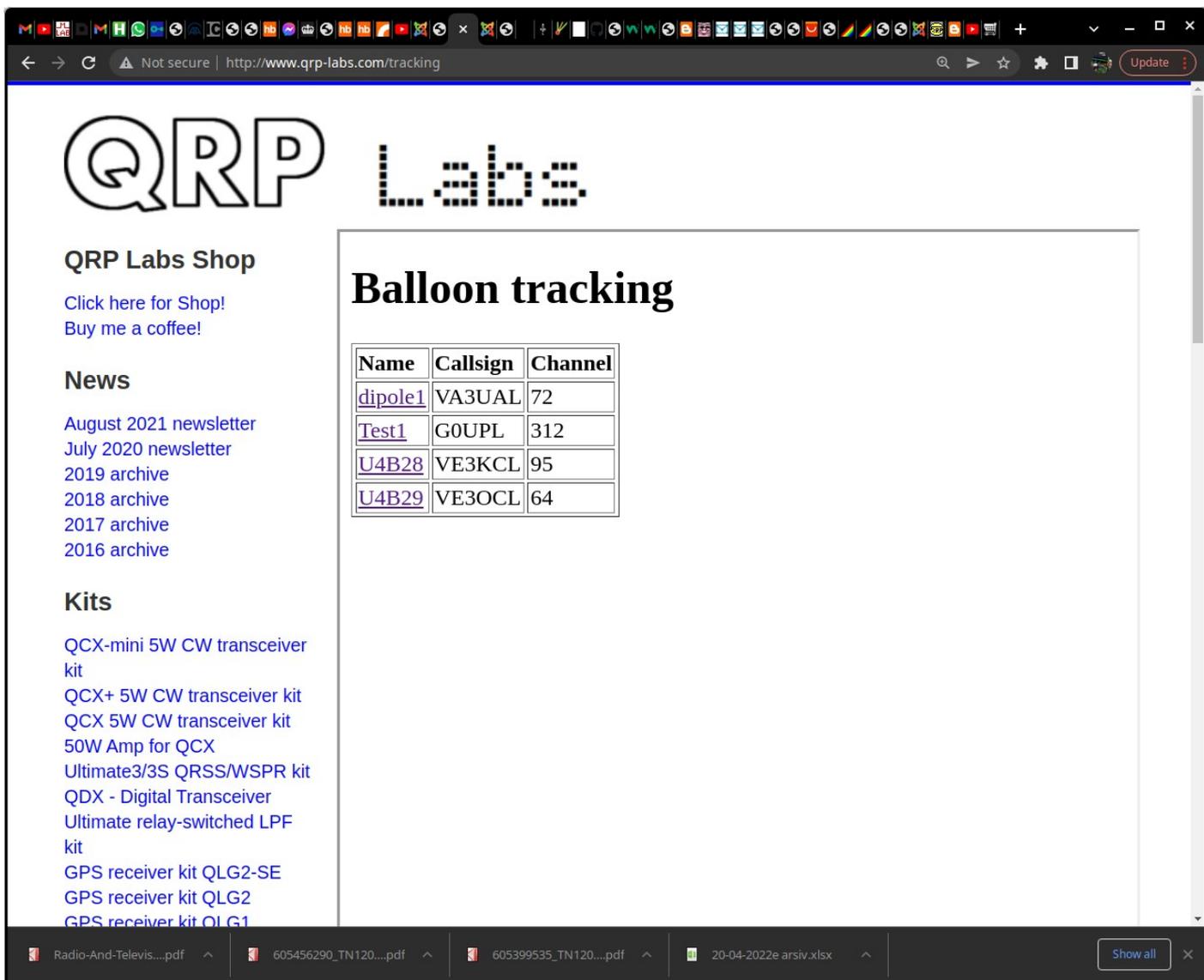
Now click your flight. For the most clear display it will be best to right click it and select "open in new tab".

The page that you will now see contains the flight name, callsign, channel, the description you entered on the flight registration page in the QRP Labs shop account section, and a live tracking map showing the latest telemetry. This page will be further enhanced in the near future, to allow downloading your entire flight data for analysis in Excel etc.

Note that the laps of our planet are colour coded according to the standard resistor colour code used in electronics, and repeat every 10 laps (should you get that far!):

1 = Brown (for lap 1, 11, 21 etc)
2 = Red
3 = Orange
4 = Yellow
5 = Green
6 = Blue
7 = Purple
8 = Grey
9 = White
0 = Black (for lap 10, 20, 30 etc)

Flight: U4B28

Callsign: VE3KCL

Channel: 95

This is the 28'th test flight of the new U4B tracker. Callsign is VE3KCL, WSPR transmission on 20m, is minute :08 and telemetry in minute :00 and telemetry call 0x4xxx Decoding of TELEN #1: 339907 103502 .... 3399 voltage in millivolts alternates from max and min...... 07 transmission number since startup or after a crash..... 1 high power mode (if blank, low power mode is assumed) ...... 035 time to gps lock in seconds maximum 333 means timeout ...... 02 reflection from clouds on down facing led sensor ... the numbers are linear till 50 and then are sort of log based to 99 a low number would indicate low reflection a reflection of 60 would be a linear number of 600 or so... lots of light from below.

Updated 2022-04-21 05:28:00 UT, Loc=RM59QE, Duration=, Distance=
Alt=11920m, Speed=94knots, Batt=3.37V, Temp=-15C, GPS=1
WSPR Frequency: 14097180 from 12 reports.

6. Resources

- For updates and tips relating to this kit please visit the QRP Labs U4B kit page http://qrp-labs.com/u4b
- For any questions regarding the assembly and operation of this kit please join the QRP Labs group, see http://qrp-labs.com/group for details
- Flight operators are encouraged to join the picoballoons groups.io forum for lots of information on all aspects of picoballooning including tracking, telemetry, antennas, balloons etc.: https://groups.io/g/picoballoon

7. Document Revision History

1.00	21-Apr-2022	First draft version version 1.00
1.00_001	26-Apr-2022	1.00_001 adds COUNTER statement and functionality
1.00_003	01-May-2022	Hardware test, Ctrl-R to display raw GPS data Hardware test, Add sys and ref freq calibration
1.00_003a	12-May-2022	Corrected error in the OUT statement (no comma to be used)
1.00_005	07-Jul-2025	Firmware version 1_00_005
1.00_005a	11-Jul-2025	Amended text description of TELEX command
1.01_000	23-Mar-2026	Firmware version 1_01_000
1.01_002	27-Mar-2026	Firmware version 1_01_002