

# Parkes Ground

---

Jack Woodman

Revision 1.4 - August 2020

## **Project Overview**

*Parkes* is the first generation of ground-based flight computers developed as part of the *Vega-Parkes* flight software package. The system is able to both send command and receive telemetry downlinks, whilst displaying live feedback on a simple UI.

The *Parkes* UI is designed to be simple and easy to use, especially during a busy launch sequence. The system runs on any Linux-based computer outputting to a 16x2 LCD display. *Parkes* is controlled using just three buttons: *select*, *cycle*, and *back*. This makes it easy to navigate the UI, select menu options and access the different features within *Parkes*. The system supports a number of communications protocols right of the box, with support for Bluetooth, WiFi, 433MHz radio and 2.4GHz radio all intended for the full release, and includes a full emulator, allowing for interaction with the *Parkes* system using any operating system that supports Python 3.

# Communications

## VAMP

The *Vega-Parkes* system communicates using a series of data packets, known as *VAMPs*. Each *VAMP* takes the form of a 22 digit string, comprising of four '\_' separated fixed-length sections. The sections are as following:

**VAMP Structure**

Lead:	Length:	Description:
<b>V</b>	5 digits	Velocity - range = 00000 - 99,999m/s
<b>A</b>	4 digits	Altitude - range = 0000 - 9999m
<b>M</b>	1 digit	Mode ID - range = 0 - 9
<b>P</b>	5 digits	Packet ID - range = 00000 - 99999 packets

**Example:** "a00342\_v0092\_m4\_p00983"

## Mode ID

The *Mode ID* *VAMP* value represents the current state of the *Vega* system. IDs 2-6 are Flight Modes, 7-9 are Diagnostics Modes, and ID 0 is Idle Mode.

## Packet ID

The Packet ID is an incremental count of how many packets have been generated and sent by the *Vega* platform, with a maximum value of 99,999 packets. *Parkes* supports an overflow but this shouldn't be necessary for most flights. *Parkes* registers each incoming packet, and keeps track of how many have been received in a *local* count.

The ratio of received packets (represented by the local count) to generated/sent packets

(represented by the Packet ID) is used to calculate the signal strength of the connection.

**Mode ID List**

ID:	Description:
<b>0</b>	Idle / Handshake Init
<b>1</b>	Armed / Ground Idle
<b>2</b>	Powered Flight
<b>3</b>	Unpowered Flight
<b>4</b>	Ballistic Descent
<b>5</b>	Chute Descent
<b>6</b>	Landing / Safe
<b>7</b>	Error / Unknown State
<b>8</b>	Heartbeat
<b>9</b>	Testing

# Communications

---

## Handshake

*Handshake* is a protocol by which *Parkes*, initiates a connection with *Vega* and establishes ID of both parties. On initiation, *Handshake* makes a call and response attempt every half second, for 30 seconds. If a response is received from a *Vega* system, *Parkes* accepts the *Vega* as a partner and completes the connection setup. Otherwise, *Handshake* reaches a timeout and returns to the connection menu.

The initiation VAMP for the *Handshake* process is:

**v10000\_a1000\_m0\_p00000**

## Heartbeat

*Heartbeat* is a protocol by which *Parkes*, following a successful handshake with a partner *Vega*, will continually monitor the strength of the connection, even while the user is otherwise operating the system. Each 'beat' is sent on a 500ms interval. *Heartbeat* runs as a threaded child process, which will continue until the process is either joined back to the parent process through the kill function, or the telemetry system is required for active use. The *Heartbeat* process stores the received Packets into the *telemetry* dictionary, as well as a signal strength calculation into a neighbouring entry.

The initiation VAMP for the *Heartbeat* process is:

**v10000\_a1000\_m8\_p00000**

The following active VAMPs take the form of two main alternating Packets, with an incrementing Packet ID. This pattern is as follows:

**v00000\_a0000\_m8\_p00000**

**v11111\_a1111\_m8\_p00001**

**v00000\_a0000\_m8\_p00002**

## Computer Identification

With the addition of a third computer, *Epoch*, to the communications loop, the Packet ID of each transmission can be used to select which computer the command is addressed to. This is because the *mode* identifier is used to send commands, and we don't want a *mode 1*

command to Vega (echo response) to accidentally trigger mode 1 on Epoch (command fire). The first number of the ID is used to address the computer, and the following four can be used to send information.

## Computer Identification

ID:	Name
0	Vega Flight Comp
1	Epoch Launch Comp
2	Parkes Flight Comp

For example, the ignition command to Epoch is "10000", and confirmation from Epoch to Parkes is "20000". Failure would be reported to Parkes as "21111", and to Vega as "01111". Obviously this doesn't apply during flight - within the flight loop all computers will receive and process all radio transmissions, which will almost exclusively come from Vega.

# Update History

## 0.1-0.2

The first two builds of *Parkes* formed the first usable version of the software, including basic menu functionality, a low-level display emulator and some support for confirmation setting.

## 0.3

The third build of *Parkes* included a number of configuration updates, as well as some under-the-hood improvements. These include:

- Dynamic length function introduced as `format_length()`
- Beep volume selection added to CONFIG
- Cursor reset function added to CONFIG
- Ability to view live configuration values added to CONFIG
- Added loading progress bar to STARTUP
- Introduced full *Parkes* emulator with keyboard support

## 0.4

The fourth build of *Parkes* included a number of major internal changes, as well as rounding-off major feature sets in anticipation of telemetry support. Updates include:

- Total overhaul of the basic OS structure, swapping on-the-fly function calling for a levelled 'inception' setup.
- Full support for *Handshake* and *Heartbeat* protocols with Vega.
- Added proper import setup, importing data from .txt file and converting data into commands and configuration values
- Implemented proper reboot and shutdown support, reinitialising config data on each reboot and destroying on shutdown
- Rebuilt function select, replacing if-statements with function dictionary
- Support for serial communication via 433Mhz radio and VAMP over normal serial.

## 0.5

The fifth iteration of *Parkes* is focussed on efficiency and debugging, reducing the bulkiness of the program and instituting more refined, efficient process and protocols while making error detection and debugging easier than ever.

Updates include:

- `dsp_menu()` function replaces individual menu setups for each interaction opportunity, removing nearly 200 lines.
- *Parkes Error Handler 2.0* added, giving additional support for error logs and general error handling
- *Parkes* configuration system overhaul, allowing for additional support for new config values and more efficient setup
- Added ability for *Parkes* to update *VFS* configuration values on-the-fly over radio
- Overhauled emulator support, allowing for on-the-fly swapping between hardware and software I/O, even while *Parkes* is running.
- Added file creation tools imported from *VFS*
- Rebuilt `hardware_update_display()` to fix a number of display bugs
- Enormous bug squashing effort, clearing bug-board to date.

## 0.6

The sixth iteration of *Parkes* is focussed on support for the *Epoch* launch computer, as well as refining launch authorisation, command and control loops and functions to really smooth out and lock in the process of launching the rocket in a robust, reliable way.

Updates include:

- Support for *Epoch* launch computer (command and control)
- Launch polling systematically asks Vega and Epoch if they are ready for flight, and continuously keeps an eye on local and remote abort conditions
- Ability to switch launch control into a mode that does not require Vega in the loop, in order to launch 'dumb' rockets
- Overhaul of launch authorisation and confirmation checks
- Added the ability for *Parkes* to command a single ignition via relay
- Transmission address support to send commands to certain computers
- Separation of remote launch via *Epoch* and local launch via *Parkes*.
- Implementing better programming practices

## Error Codes

### Non-Fatal Error

Code:	Error:
<b>E000</b>	Non-fatal runtime error
<b>E001</b>	Function could not run
<b>E010</b>	Telemetry dictionary is corrupted

### Fatal Error

Code:	Error:
<b>E100</b>	Fatal configuration error
<b>E101</b>	Failed startup_test()



Code:	Error:
<b>E102</b>	\$expected_value.set=int command missing int argument
<b>E103</b>	\$expected_value command missing operation argument
<b>E104</b>	Found configuration values does not equal expected configuration values
<b>E105</b>	<i>Parkes</i> version is out of date, update required
<b>E106</b>	Configuration file is out of date, update required
<b>E107</b>	hot_run not configured to bool
<b>E120</b>	EMPTY
<b>E121</b>	EMPTY
<b>E122</b>	VAMP could not be verified as string
<b>E123</b>	VAMP could not be verified as tuple
<b>E199</b>	error() function unable to determine error type

#### Warning

Code:	Error:
<b>E200</b>	Could not complete operation
<b>E201</b>	Could not set value
<b>E202</b>	Startup tests have been disabled, proceed with caution
<b>E203</b>	
<b>E210</b>	Forcing launch may cause fatal issues, proceed with caution
<b>E250</b>	Heartbeat couldn't be killed: no heartbeat active
<b>E251</b>	hb_count is not 0 - resetting hb_count value

Code:	Error:
<b>E252</b>	hb_status could not be determined
<b>E290</b>	Hotfire is dangerous, proceed with caution
<b>E291</b>	Preflight checks failed - autosequence abort
<b>E292</b>	Vega poll returned NO GO - autosequence abort
<b>E293</b>	Vega poll could not be resolved - autosequence abort
<b>E294</b>	Vega did not confirm autosequence start - autosequence abort
<b>E295</b>	Parkes QuickCheck failed - countdown abort
<b>E296</b>	Epoch poll could not be resolved - autosequence abort
<b>E297</b>	Epoch did not confirm autosequence start - autosequence abort
<b>E298</b>	Epoch poll returned NO GO - autosequence abort

#### Passive

Code:	Error:
<b>E302</b>	Display update function failed - length error
<b>E310</b>	Error creating VAMP - not enough variables
<b>E311</b>	Error deconstructing VAMP - data may be corrupted
<b>E312</b>	Heartbeat connection timeout: could not establish connection
<b>E320</b>	Vega config update totally failed
<b>E321</b>	Vega config update partially failed
<b>E322</b>	Expected E321 data but did not receive it
<b>E399</b>	go_reboot / go_kill detected

#### System Message

Code:	Error:
<b>E900</b>	Generic system message
<b>E901</b>	Shutdown process initiated

Code:	Error:
<b>E909</b>	Shutdown complete
<b>E910</b>	Opened port - parkes_radio
<b>E911</b>	Heartbeat loop initiated
<b>E912</b>	Heartbeat confirmation received
<b>E913</b>	Handshake good - entering multithread
<b>E913</b>	Heartbeat kill command received
<b>E990</b>	Vega poll returned GO - Vega is configured for flight
<b>E998</b>	Launch countdown commit
<b>E999</b>	Ignition!

## Function Definitions

As of Parkes 0.4, functions are stylised in the form *xyz\_function()*, whereby *xyz* represents a three letter classifier that categorises the function into a certain purpose, and *function* representing the unique function name. This prevents confusion and allows for a function names from different areas of *Parkes* to be reused. For example, *con\_countdown()*, *lch\_countdown()* and *disp\_countdown()* may all exist at the same time. Verbosity is preferred over over-simplification however, to reduce confusion.

Classifier	Type
<b>sys</b>	Functions relating to high level operation in the global frame.
<b>con</b>	Functions relating to the configuration submenu and related processes.
<b>lch</b>	Functions relating to the launch submenu and related processes.
<b>cne</b>	Functions relating to the connect submenu, related processes, and radio transmission/reception processes.
<b>dsp</b>	Functions that display data.
<b>cfg</b>	Functions that handle configuration setup
<b>bug</b>	Functions strictly for debug purposes.

**novae space 2020**