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 *VAMP*s. 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:
V	5 digits	Velocity - range = 00000 - 99,999m/s
A	4 digits	Altitude - range = 0000 - 9999m
M	1 digit	Mode ID - range = 0 - 9
P	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.

Mode ID List

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

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.

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

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 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
- Rebuild 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
- Overhauled emulator support, allowing for on-the-fly swapping between hardware and software I/O, even while *Parkes* is running.
- Rebuilt hardware_update_display() to fix a number of display bugs
- Enormous bug squashing effort, clearing bug-board to date.

Error Codes

Non-Fatal Error

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

Fatal Error

Code:	Error:
E100	Fatal configuration error
E101	Failed startup_test()
E102	$expected_value.set=int command missing int argument
E103	$expected_value command missing operation argument
E104	Found configuration values does not equal expected configuration values
E105	*Parkes* version is out of date, update required
E106	Configuration file is out of date, update required
E107	hot_run not configured to bool
E120	EMPTY
E121	EMPTY
E122	VAMP could not be verified as string
E123	VAMP could not be verified as tuple
E199	error() function unable to determine error type

Warning

Code:	Error:
E200	Could not complete operation
E201	Could not set value
E202	Startup tests have been disabled, proceed with caution
E203	Display update function failed - length error
E210	Forcing launch may cause fatal issues, proceed with caution
E250	Heartbeat couldn't be killed: no heartbeat active
E251	hb_count is not 0 - resetting hb_count value
E252	hb_status could not be determined

Passive

Code:	Error:
E310	Error creating VAMP - not enough variables
E311	Error deconstructing VAMP - data may be corrupted
E312	Heartbeat connection timeout: could not establish connection
E399	go_reboot / go_kill detected

System Message

Code:	Error:
E900	Shutdown process initiated
E901	Shutdown complete

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