PSAS Telemetry Server Documentation

Release

Nathan Bergey, Ashley DeSimone, Bogdan Kovch, Karl Hiner, Emi

July 28, 2015

Contents

Contents

CHAPTER 1

Project Structure

```
.
|-- build_for_flight.sh (use on flight computer)
|-- cg.conf (code gen configuation)
|-- doc (project documentation)
|   |-- ...
|-- elderberry (Event loop code gen submodule)
|   |-- ...
|-- fc_bin (Executable commands like ARM and SAFE)
|   |-- ...
|-- main.miml (project configuation)
|-- Makefile
|-- README.md
|-- requirements.txt
`-- src (FC source code)
    |-- devices (device data handling)
    |   |-- ...
    | -- utilities (helpers and includes)
    |   |-- ...
    | -- main code
    `-- ...
```

CHAPTER 2

Flight Hardware

All flight data is handled via Ethernet. On the flight computer we have a hard-coded set of network addresses, recorded here.
When running locally the ports are the same, but localhost is used for all IP addresses.

2.1 Avionics Network Information

2.1.1 Flight Computer

- IP Addr: `10.0.0.10`
- MAC Addr: ?
- Data Listen Port: `36000`

PSAS is a student aerospace engineering project at Portland State University. We are building ultra-low-cost, open source rockets that feature some of the most sophisticated amateur rocket avionics systems out there today.

The primary flight computer is the brains of the rocket, and our playground for testing future rocket technology. It's written in C with an event loop model.

We made a custom code generator to do all the event glue and boilerplate, allowing us to focus on writing functionality.

CHAPTER 3

Install

3.1 Requirements

The code generator runs on python3 using the pyyaml package. It's recommended to use a python virtual environment like this:

```
$ sudo apt-get install python3 libyaml-0-2 python-pip virtualenvwrapper
$ mkvirtualenv -p <path/to/python3/install> av3fc
(av3fc)$ pip install -r requirements.txt
```

Note: If this is your first time using python virtual environments, remember to kill your shell and open a new one after installing virtualenvwrapper for the first time (you only have to do this once).

3.2 Building

Due to the need for user module abstraction, the build process for the framework is a little more complicated than that of a typical C application. Here is the general build process:

1. User module MIML files and the Main.miml file are passed into the code generator.
2. The code generator, upon successful parsing and validation, creates the fcfmain.c file that include the intermodular data handlers and a Miml.mk Makefile include file.
3. The Makefile imports the Miml.mk and should successfully compile, link and build the executable "fc."

To help uncomplicate this process, however, the Makefile has been created so that the user only needs to run:

```
(av3fc)$ make
```

For the project to complete all three steps.

CHAPTER 4

Running

The finished executable will be in the top level directory and is called ‘fc’. To start the flight computer run:

```
$ ./fc
```

It will start and wait for some kind of sensor input or command.