Faraday Software Documentation

Release 0.1.0

FaradayRF

Jul 21, 2017

Contents

The Faraday radio software is a collection of toolsets, applications, and useful code to fully implement a Faraday radio with a computer connection. For a high-level overview of Faraday code check out our code page! ** Early users of Faraday are expected to be developers, helping define how Faraday is implemented and the methods in which it helps solve amateur radio tasks**.

The core applications are the heart of Faraday which help the user and developer interface with the radio. All applications attempt to remain RESTful, adhearing to tried and true web application practices. This fundamentally shifts radio experimentation towards an application based endeavor, disassocaiting low-level RF protocol issues from actual use of the radio.

CHAPTER 1

Quickstart Information

Installation

Placeholder

Getting Started

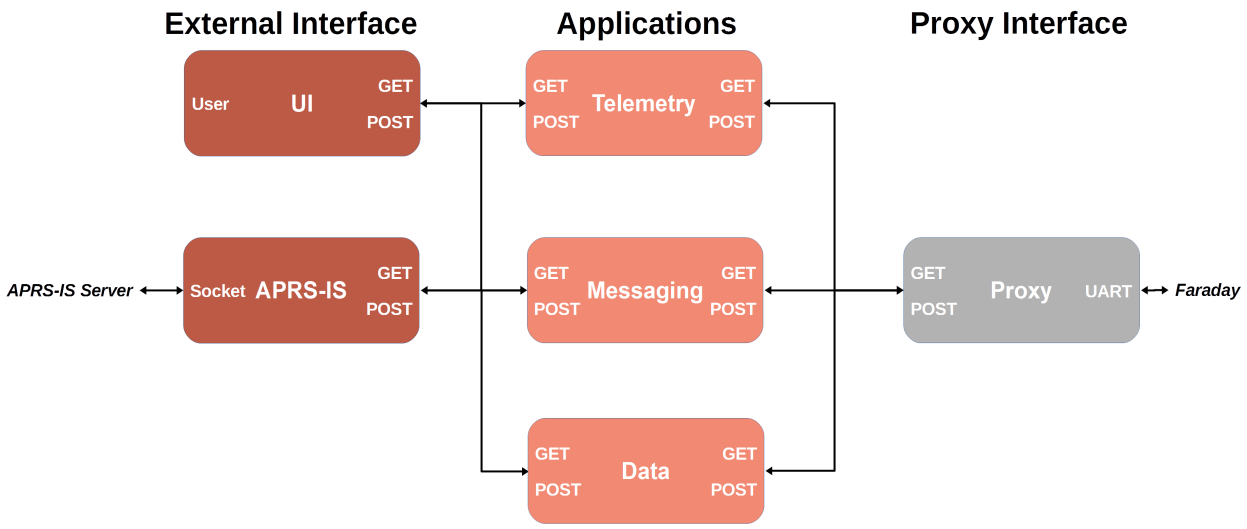
Faraday is designed to be simple and intuitive. The entire philosophy is based on amateur radio being an amazing learning tool as well as useful sandbox for experimentation. Let's start at the bare basics, starting up the Faraday software

Starting Proxy

To start proxy simply double-click on the proxy.py script. You should see the python script running and server information present as you interact with it.

CHAPTER 2

Core Applications



Proxy

Proxy

The Proxy application is the interface between a Faraday UART USB compliant radio and the Localhost interface on a computer. It handles all serial COM port communications presenting data to the user through a RESTful interface. Nearly all applications will use the Proxy to communicate over the radio hardware. Unless you are absolutely sure you need to bypass Proxy it is highly encouraged to use it.

Basic Operation

Proxy works on the premise of spinning up a thread which has the sole purpose of querying every "port" (0-255) of the Faraday radio to see if new data is available. This is called the "UART Worker". Upon data being available the thread will receive the data and place it into a buffer queue. This queue is 100 packets long and is a FIFO resulting in old data being popped off if no user is requesting data. The same occurs in reverse.

A flask server runs in the main process which provides a RESTful interface for the Proxy. When the RESTful interface is queried with a GET request the thread-safe queue will pop off packets from the left for the requested Faraday "port". This data is served to the user in a JSON dictionary. If the RESTful interface received a POST request to send data to Faraday then the flask server will place the packet onto the queue from the right. Every 10ms the UART Worker checks to see if there is any data for any port in the transmit queue. If present, this data is immediately sent to Faraday via USB UART.

API Documentation

Send/Receive Faraday Data

Checks the proxy queue for the specified port. If packets are present in the qeue they are returned as a JSON dictionary as an HTTP response. Additionally, the POST method will add packets to the POST queue which are sent to Faraday on a periodica basis. Invalid parameters are responded with appropriate HTTP responses and relevant warning messages.

- **http://localhost:8000/** Port 8000 of localhost (127.0.0.1)
- **Method** GET | POST
- **URL Params**

  Required

  - `Callsign=[String]`
  - `NodeID=[Integer]`
  - `Port=[Integer]`

  Optional

  - `Limit=[Integer]`
- **Data Params** When making a POST request Proxy expects to receive JSON data in the body with an object containing a string "data" and value being an array of BASE64 encoded strings each 164 characters long. The header of the POST request must specify the content type as application/json. Only a data array of 100 packets is accepted.
- **Success Response** GET
  - **Code: 200 OK Content**: `[{"data": "<164 Characters of BASE64>"},{"data": "<164 Characters of BASE64>"},...]`
  - **Code: 204 NO CONTENT Content**: Empty string

  POST
  - **Code: 200 OK Content**: `{ "status": "Posted x of y Packet(s)" }`
  - **Code: 204 NO CONTENT Content**: Empty String
- **Error Response** GET
  - **Code: 400 BAD REQUEST Content**: `{ "error": "<Python exception string>" }`

POST

- **Code: 400 BAD REQUEST Content**: `{ "error": "<Python exception string>" }`

- **Sample Call** GET:

```
# Simple GET request putting response into variable r
# Port 5 is the telemetry port of Faraday so we will receive three telemetry␣
↪packets
payload = {'callsign': 'kb1lqd','nodeid': 22,'port':5,'limit': 3}
r = requests.get('http://localhost:8000/',params=payload)

# Printing text response from Proxy
print(r.text)

# Printing with requests built-in JSON decoder
for item in r.json():
    print item, '\n'
```

Printing text output from Python sample GET call:


```
[
 {
  "data":
↪"AwBhS0IxTFFEBXsDBhZLQjFMUUQFewMGFggAAAACANAHMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwM
↪"
 },
 {
  "data":
↪"AwBhS0IxTFFEAADDBhZLQjFMUUSxAAAGFgkAAAACANAHMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwM
↪"
 },
 {
  "data":
↪"AwBhS0IxTFFEBXsDBhZLQjFMUUQAAMMGFgoAAAACANAHMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwM
↪"
 }
]
```


Printing output of requests built-in JSON decoder from Python Sample GET Call:


```
{u'data': u
↪'AwBhS0IxTFFEBXsDBhZLQjFMUUQFewMGFggAAAACANAHMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwM
↪'}

{u'data': u
↪'AwBhS0IxTFFEAADDBhZLQjFMUUSxAAAGFgkAAAACANAHMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwM
↪'}

{u'data': u
↪'AwBhS0IxTFFEBXsDBhZLQjFMUUQAAMMGFgoAAAACANAHMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwM
↪'}
```


POST:

```
# Create dictionary with array of one BASE64 string which turns LED 1 ON
# Send POST data to port 2 of Faraday radio which is the command port
content = {'data': ["BQZAAACA/
↪wwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
↪"]}
```

```
payload = {'callsign': 'kb1lqd','nodeid': 22,'port':2}

r = requests.post('http://localhost:8000/', params=payload, json=content)

# Print text response from Proxy
print(r.text)
```

Printing text output response from Python Sample POST call:

```
{"status": "Posted 1 of 1 Packet(s)"}
```

- **Notes**
  - **10/9/2016**
    - Proxy uses the built-in debug server of Flask. For higher performance one may need to move to an Apache or WSGI based server.
    - All text responses from Proxy are to be JSON
    - No attempt to secure local communications have been made yet with HTTPS as these services are intended to stay localhost.
    - Running on http://localhost:8000 allows port 80 to be used by a User Interface (UI) application.

The Proxy application is the gateway to Faraday via USB UART communications. With Faraday plugged into a USB port on a computer, Proxy interfaces the radio with the user over a RESTful localhost interface. A basic description of proxy operation as well as the API documentation is located at the above link.

Telemetry

Telemetry Application

The Telemetry application interfaces the Proxy and provides a RESTful interface to decoded telemetry from Faraday as well as the ability to save telemetry to a local storage facility such as SQLite.

Messaging

Messaging Application

The Messaging application interfaces the Proxy and provides the user an ability to RESTfully send/receive text-based messages of arbitrary length.

Data

Data Application

The Data application interfaces the Proxy and provides a RESTful interface to send/receive data of arbitrary length. This could be text files, images, binary, or any other form of data.

APRS-IS

APRS-IS Application

The APRS-IS application interfaces the telemetry, messaging, and data application with the APRS-IS system via a web socket. It provides basic interfacing to the APRS system to allow Faraday radios to appear on the APRS interfaces such as aprs.fi

User Interface

User Interface Application

The User Interface (UI) application interfaces the telemetry, messaging, and data applications ith a web-based control and viewing application. The UI is essentially a dashboard with bidirectional interfacing to Faraday allowing one to see data from the Faraday radios, control local and remote radios, and configure local radios settings.

Toolset Module Documentation

FaradayIO Toolset Documentation

The Faraday IO toolset is a collection of Python scripts that perform input/output, packet creation, packet parsing, and other functions that provide a basic level of interaction with a Faraday digital radio. This module is specifically designed to be used in the creation of applications that wish to interact with a Faraday device over the localhost proxy server RESTful API.

faradayio - Faraday Proxy Input/Output Toolset

Faraday IO is a basic Python toolset that can be used to interact with the RESTful API connecting the Faraday Digital Radio to a localhost server. This toolset includes basic input and output functionality along with command and parsing definitions to jumpstart application development.

faradaycommands - Command Application Toolset

Faraday Commands is a collection of packet creation tools and predefined common commands focused on the command and control functionality that the Faraday "Commands" application provides. The application for Commanding contains its own packet format as observed in the OSI layer and this toolset provides easy to use commands and tools to create command packets.

commandmodule - Command Application Packet Creator

The commandmodule Faraday Python module is a collection of packet creation functions that create command application packets.

cc430radioconfig - CC430 Radio Configuration Module

The commandmodule Faraday Python module is a collection of functions that perform calculations and other actions for the CC430 radio module settings.

checksum - A Simple Error Detection Checksum 16 Bit

The commandmodule Faraday Python module contains a function(s) that compute a simple 16 bit checksum used for error detection and correct.

deviceconfig - Faraday Flash Configuration Tool

The deviceconfig Faraday Python module contains class objects and function(s) that help create a flash configuration packet payload to update the Faraday non-volatile configuration.

gpioallocations - Faraday GPIO Port Allocations Definitions

This module is a simple collections of port definitions that describes what a GPIO's function or connection is. This is useful especially when creating commands to update GPIO state.

Note: The source code below is static and not auto-generated!

```
#Port 3
GPS_RESET = 0b00001000
GPS_STANDBY = 0b00010000
LED_1 = 0b01000000
LED_2 = 0b10000000

#Port 4
DIGITAL_IO_0 = 0b10000000
DIGITAL_IO_1 = 0b01000000
DIGITAL_IO_2 = 0b00100000
DIGITAL_IO_3 = 0b00010000
DIGITAL_IO_4 = 0b00001000
DIGITAL_IO_5 = 0b00000100
DIGITAL_IO_6 = 0b00000010
DIGITAL_IO_7 = 0b00000001

#Port 5
DIGITAL_IO_8 = 0b00000100
DIGITAL_IO_9 = 0b00001000
MOSFET_CNTL =  0b00010000
```

telemetryparser - Faraday Telemetry Packet Parser

This module is a collection of class objects and functions that perform actions for the Faraday "Telemetry" application, specifically parsing the packets received from a Faraday device.

FaradayIO Toolset Tutorials

Faraday IO is a basic Python toolset that can be used to interact with the RESTful API connecting the Faraday Digital Radio to a localhost server. The tutorials presented will walk from basic input/output operations to the process of building larger applications using the toolset.

Starting Proxy

All tutorials for FaradayIO require an active "Proxy" to be running. To start proxy simply double-click on the proxy.py script. You should see the python script running and server information present as you interact with it. To learn more about configuring and starting Faraday Proxy view the tutorial below.

Tutorials

Configuration and Initialization Of Faraday Proxy

Basic Input/Output With Faraday IO

This tutorial will teach how to perform basic input and output functionality using the FaradayIO toolset. Retrieving data from the telemetry application preinstalled onto faraday is goal of this tutorial. Full example code is available in the GIT repository.

Prerequisites

Ensure that these steps and configurations have been performed prior to using the tutorial.

- **Faraday Proxy Configuration INI File**
- In section [telemetry] set *autoretrieve' = False
- **Faraday Proxy**
- Successfully start the proxy service

Tutorial

Tutorial/Example Source Code: https://github.com/FaradayRF/Faraday-Software/tree/master/Tutorials/FaradayIO/Tutorial_0

The example code provided retrieves telemetry data from the locally connected (USB) Faraday Device, decodes the "Proxy Server" JSON BASE64 data, and parses the Faraday Telemetry (Packet Type #3) into a readable format.

Notable Key Points

When the example code is correctly the output data will be similar to the output below.

```
--- Telemetry Datagram ---
Telemetry Packet Type: 3
Telemetry RF Source: 0
Telemetry Payload Length: 97
Telemetry Packet 16 Bit Checksum: 5718
Telemetry Packet Payload Data: 'KB1LQD\x05{\x03\x06\x07KB1LQD\x05
↪{\x03\x06\x07\x12\x0b\x06\x13\x01\t\xe0\x073352.4203N11822.6005W34.30000M0.22021.
↪24\x00\x07`\x06\xf8\x05e\x04\xcc\x04\x86\x04\x87\x04s\x00\x00\x00\x1c\x0b\x03\x00\x00\x1c␣
↪\x00\x00\x00\x00\x00\x00\xd8\x06\x07KB1LQD\xff\x00\x00\x06\x07\x1e\n\x06'


--- Telemetry Packet #3 ---
Index[0]: Source Callsign KB1LQD{
```

```
Index[1]: Source Callsign Length 6
Index[2]: Source Callsign ID 7
Index[3]: Destination Callsign KB1LQD{
Index[4]: Destination Callsign Length 6
Index[5]: Destination Callsign ID 7
Index[6]: RTC Second 18
Index[7]: RTC Minute 11
Index[8]: RTC Hour 6
Index[9]: RTC Day 19
Index[10]: RTC Day Of Week 1
Index[11]: RTC Month 9
Index[12]: Year 57351
Index[13]: GPS Lattitude 3352.4203
Index[14]: GPS Lattitude Direction N
Index[15]: GPS Longitude 11822.6005
Index[16]: GPS Longitude Direction W
Index[17]: GPS Altitude 34.30000
Index[18]: GPS Altitude Units M
Index[19]: GPS Speed 0.220
Index[20]: GPS Fix 2
Index[21]: GPS HDOP 1.24
Index[22]: GPIO State Telemetry 0
Index[23]: RF State Telemetry 7
Index[24]: ADC 0 96
Index[25]: ADC 1 1784
Index[26]: ADC 2 1381
Index[27]: ADC 3 1228
Index[28]: ADC 4 1158
Index[29]: ADC 5 1159
Index[30]: ADC 6 1139
Index[31]: CC430 Temperature 0
Index[32]: ADC 8 28
Index[33]: N/A Byte 2819
Index[34]: HAB Automatic Cutdown Timer State Machine State 0
Index[35]: HAB Cutdown Event State Machine State 0
Index[36]: HAB Automatic Cutdown Timer Trigger Time 7200
Index[37]: HAB Automatic Cutdown Timer Current Time 0
```

Tutorial 1 - Local Command and Control

This tutorial will walk you through basic local device command and control as well as remote device command and control over RF. It is important to note that this tutorial's command and control is provided by an "application" operating on the Faraday CC430 and we are issuing packets that invoke actions to occure.

Key Points - Basic command application usage

- Toggle LED GPIOs using both "FaradayIO" predefined commands and the generic GPIO function
- Send and "ECHO" command and recevied, decode, and display the resulting echo'd message

Tutorial Example/Source Code: https://github.com/FaradayRF/Faraday-Software/tree/master/Tutorials/FaradayIO/Tutorial_1

Tutorial 1a - Basic Local Device IO Commands

This simple example tutorial shows the use of the command application's GPIO control functions. The code is well commented and should be self-document.

Notable Key Points

Turning ON a GPIO (such as an LED) using the pre-defined functions is very simple as shown below. Note that the "local_device_callsign" and "local_device_node_id" are identifiers to command a specific unit attached to a single computer with multiple Faraday units connected. This will be more useful in later tutorials.

```
#Turn LED 1 ON LOCAL
command = faraday_cmd.CommandLocalGPIOLED1On()
faraday_1.POST(local_device_callsign, local_device_node_id, faraday_1.CMD_UART_PORT,␣
↪command)
```

The pre-defined functions simply invoke the base GPIO toggling function below. The key point to learn is that a value of LOW (0) performs *no action* and a value of HIGH (1) *performs either a TOGGLE HIGH or TOGGLE LOW* depending on the function arguments. You cannot attempt to toggle HIGH and LOW in a single command. A GPIO header file is provided (gpioallocations.py) that maps the pins on the CC430 to Faraday functions.

```
#Turn Both LED 1 and LED 2 OFF simultaneously
command = faraday_cmd.CommandLocalGPIO(0, 0, 0, (gpioallocations.LED_
↪1|gpioallocations.LED_2), 0, 0)
faraday_1.POST(local_device_callsign, local_device_node_id, faraday_1.CMD_UART_PORT,␣
↪command)
```

After transmitting a command "ECHO" packet with a supplied payload (that will be echo'd) Faraday takes a non-zero amount of time to respond. this is typically slower than the very powerful computer running the Python code and the GetWait() function blocks the script until a data message is received (assumed to be the message inteded).

```
#Retrive waiting data packet in UART Transport service number for the COMMAND␣
↪application (Use GETWait() to block until ready or return False).
rx_echo_raw = faraday_1.GETWait(local_device_callsign, local_device_node_id, faraday_
↪1.CMD_UART_PORT, 1, False) #Wait for up to 1 second
```

Running this example toggles the LED's on Faraday and produces the following output:

```
*** Remote Interpreter Reinitialized  ***
>>>
Original Message:  This will ECHO back on UART
RAW Received BASE64 ECHO'd Message:␣
↪VGhpcyB3aWxsIEVDSE8gYmFjayBvbiBVQVJUAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAD/
↪////////////////////////////////////////////////////////////////////////////
Decoded received ECHO'd Message: This will ECHO back on UART
>>>
```

Tutorial 1b - Commanding And Parsing All Telemetry Packets

The purpose of this tutorial is to command a local Faraday unit to transmit all 3 current types of telemetry packets and how to use the supplied parsing tools to decode them.

Note: Make sure that Faraday Proxy is successfully running prior to running the example script!

- **Command transmisson of different telemetry packets**
  - Packet Type #1: System Settings (i.e. Frequency, RF power...)
  - Packet Type #2: Device Debug Information (Boot counter, reset counters, error flags, etc...)
  - Packet Type #3: Standard Faraday Telemetry (ADC, GPIO, GPS, etc...)
- **Using FaradayIO parsing tools to decode received telemetry packets from the proxy server**
  - Introduce the "Flush" function in FaradayIO
  - Extracting smaller packets from larger datagram payloads
- Overview of the telemetry packet formats and data available

Example Notes

Telemtry Packet #3 - Standard Telemetry

This telemetry packet is the main telemetry that most people will want from a Faraday as it contains the ADC, GPIO, GPS, and other information.

Below is an example of output data when the the debug argument is equal to True thus printing parsing information for educational purposes. This function otherwise returns a list of parsed elements using the standard python Struct module.

```
>>>
--- Telemetry Packet #3 ---
Index[0]: Source Callsign KB1LQD"*
Index[1]: Source Callsign Length 6
Index[2]: Source Callsign ID 7
Index[3]: Destination Callsign KB1LQD
Index[4]: Destination Callsign Length 6
Index[5]: Destination Callsign ID 7
Index[6]: RTC Second 14
Index[7]: RTC Minute 30
Index[8]: RTC Hour 17
Index[9]: RTC Day 23
Index[10]: RTC Day Of Week 5
Index[11]: RTC Month 9
Index[12]: Year 57351
Index[13]: GPS Lattitude 3352.4159
Index[14]: GPS Lattitude Direction N
Index[15]: GPS Longitude 11822.6014
Index[16]: GPS Longitude Direction W
Index[17]: GPS Altitude 40.26000
Index[18]: GPS Altitude Units M
Index[19]: GPS Speed 0.190
Index[20]: GPS Fix 2
Index[21]: GPS HDOP 0.94
Index[22]: GPIO State Telemetry 0
Index[23]: RF State Telemetry 7
Index[24]: ADC 0 96
Index[25]: ADC 1 2237
Index[26]: ADC 2 2041
```

```
Index[27]: ADC 3 2023
Index[28]: ADC 4 1928
Index[29]: ADC 5 1934
Index[30]: ADC 6 1878
Index[31]: CC430 Temperature 0
Index[32]: ADC 8 29
Index[33]: N/A Byte 2816
Index[34]: HAB Automatic Cutdown Timer State Machine State 0
Index[35]: HAB Cutdown Event State Machine State 0
Index[36]: HAB Automatic Cutdown Timer Trigger Time 7200
Index[37]: HAB Automatic Cutdown Timer Current Time 0
('KB1LQD\x93*\x00', 6, 7, 'KB1LQD\x00\x02\x00', 6, 7, 14, 30, 17, 23, 5, 9, 57351,
↪'3352.4159', 'N', '11822.6014', 'W', '40.26000', 'M', '0.190', '2', '0.94', 0, 7,␣
↪96, 2237, 2041, 2023, 1928, 1934, 1878, 0, 29, 2816, 0, 0, 7200, 0)
} 14
```

Below are excerpts and explainations of key points to understand from this

Tutorial 1 - Remote RF Command and Control

Tutorial 3 - Writing A Basic Messaging Application

License

Faraday is licensed under the **GNUv3 with an additional exception for network interfaces**. Read the license to learn more.