Introduction

Pumpkin's General Use Telecommand System (GUTS) is a robust, ready to use, and extensible collection of flight software for cubesat missions.

What does GUTS provide?

- GUTS Services
  - A set of microservice applications that implement essential functionality to interface with the operating system, spacecraft bus hardware, and spacecraft payload hardware. Exposes functionality via HTTP endpoints servicing GraphQL requests. Returns JSON responses allowing usage of many different programing languages.
  - Ready-to-use microservices for:
    - Telemetry and command handling
    - Radio communications and encryption
    - Interfacing with Pumpkin's cubesat modules
    - Solar array articulation
    - File uplink and downlink
    - ADCS control
    - GNSS time sync
    - Thruster control
    - Mission logic (CONOPS)
    - Error condition detection and recovery
- GUTS Linux
  - Yocto based embedded Linux optimized for GUTS flight software
  - Highly extensible and configurable for specific missions
  - Bundles GUTS flight software into ready-to-use images
  - Low-effort porting to any hardware supported by a Yocto layer
  - Fault-tolerant boot and update mechanisms
- An SDK for developing flight software in Rust, C, and optionally Python
  - libraries for speeding up development
  - code examples

Supported Hardware

Pumpkin Hardware

- C&DH / main computer
  - Pumpkin MBM2 + Beaglebone Black
  - Additional hardware support can be added thanks to Yocto's extensibility
- Solar Array
  - Pumpkin Dual Articulated Deployable Solar Array (DASA)
- Power Systems and Batteries
  - Pumpkin Electrical Power System Module (EPSM)
  - Pumpkin Battery Module (BM2)
  - Pumpkin Linear Electrical Power System (LEPS)
- GNSS
  - Pumpkin GNSS Receiver Module (GPSRM)

Third Party Hardware

- Radio
  - IQ Wireless Xlink Radio
  - NearSpace Launch EyeStar-S4 Iridium Radio
- ADCS
  - AAC Hyperion IADCS400
- Thruster
  - AASC Solid State Thruster

Other Resources

- SupMCU Reference Manual
- RS3/FlatSat User Manual
- Novatel OEM7 Receiver User Documentation

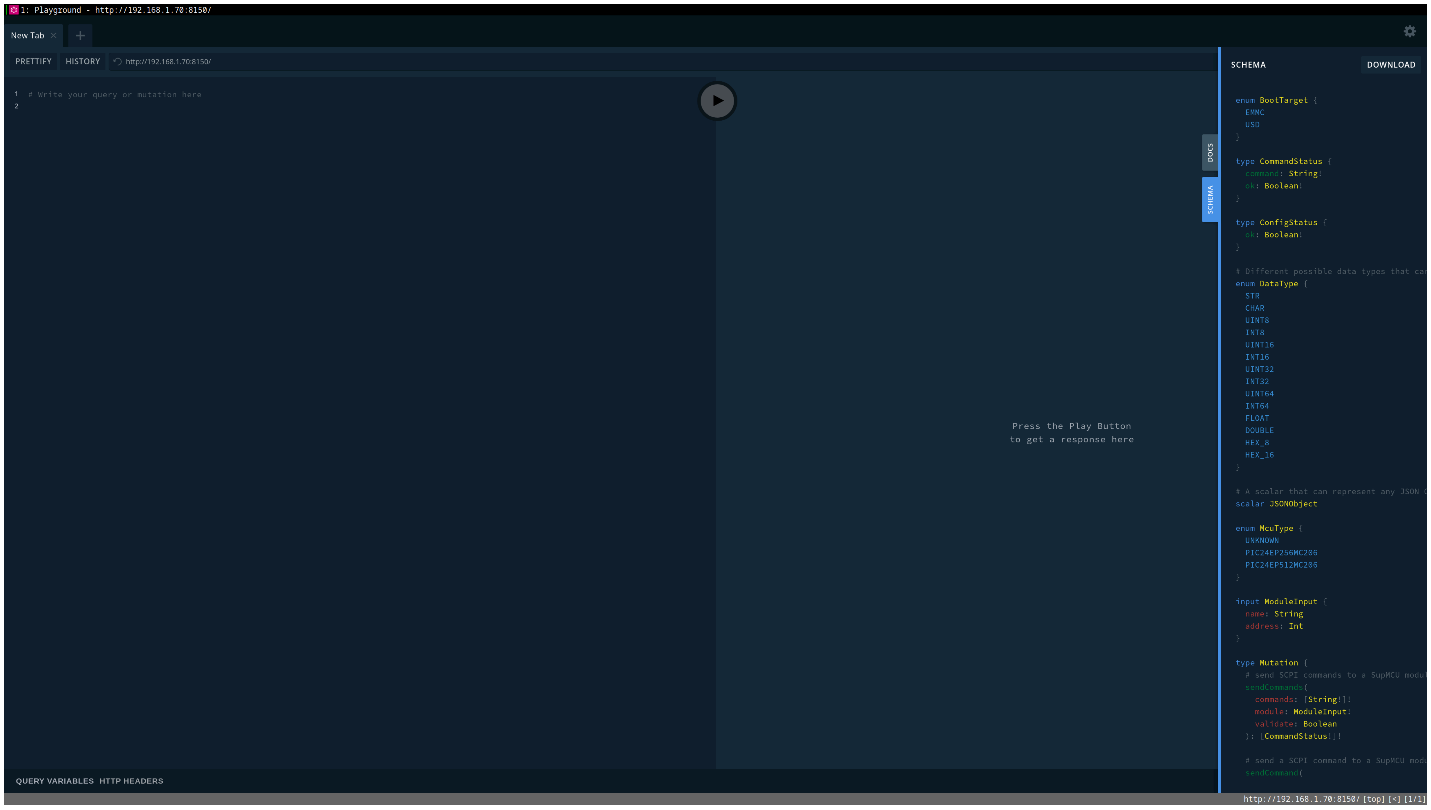
Overview

GUTS includes many ready-to-use microservices that enable the functionality of your cubesat. GUTS services are written in Rust and serve GraphQL APIs over IP to interact with other software and/or users.

Graphql Playground

GUTS services present a graphql playground interface over http for development and experimentation. This is a great way to explore a service's functionality and run tests. When connected to your cubesat over Ethernet, simply navigate to `http://{cubesat-ip}:{service-port}/` in your web browser.

For example, the graphql playground for the MCU service can be accessed at http://192.168.1.70:8150/:



```
# Write your query or mutation here
```

```
Press the Play Button
to get a response here
```

```
enum BootTarget {
  EMMC
  USD
}

type CommandStatus {
  command: String!
  ok: Boolean!
}

type ConfigStatus {
  ok: Boolean!
}

# Different possible data types that ca
enum DataType {
  STR
  CHAR
  UINT8
  INT8
  UINT16
  INT16
  UINT32
  INT32
  UINT64
  INT64
  FLOAT
  DOUBLE
  HEX_8
  HEX_16
}

# A scalar that can represent any JSON
scalar JSONObject

enum McuType {
  UNKNOWN
  PIC24EP256MC206
  PIC24EP512MC206
}

input ModuleInput {
  name: String
  address: Int
}

type Mutation {
  # send SCPI commands to a SupMCU modul
  sendCommands(
    commands: [String!]!
    module: ModuleInput!
    validate: Boolean
  ): [CommandStatus!]!

  # send a SCPI command to a SupMCU modu
  sendCommand(
```

```
http://192.168.1.70:8150/ [top] [<] [1/1]
```

To see all the GUTS services currently installed on your cubesat, inspect the `/etc/guts.d` directory.

GUTS services include systemd units for automatically starting them at boot.

MCU Service

The MCU service provides a high-level interface to Pumpkin's SupMCU modules.

Background

A Pumpkin cubesat bus consists of SupMCU hardware modules (see the SupMCU Reference Manual for more information). All modules share an I2C bus with the main flight computer that runs Linux. This I2C Bus is used to send commands to and receive telemetry from the SupMCU modules. The MCU service provides an abstracted interface for:

- requesting telemetry from SupMCU modules
- sending commands to SupMCU modules
- updating SupMCU module firmware (optional)
- switching OS boot targets (optional)

It is also responsible for servicing the EPSM's watchdog timer via a periodic command.

Getting Started

- by default the MCU service will start on boot, binding to port `8150`
- navigate to http://192.168.1.70:8150 for an interactive graphql playground interface
- run `mcu-service --help` to see command-line usage of the service

Graphql Examples

```
# check for signs of life
query alive {alive}

# check what modules are available on the bus
query modules {
  modules{name, address, mcu, responseDelay}
}

# get all module definitions
query module_definitions {
  modules {
    name,
    address,
    simulatable,
    mcu,
    responseDelay,
    telemetry {
      name,
      format {format},
      length,
      idx,
      telemetryType,
    },
    commands {name, idx}
  }
}

# set the EPSM's LED to be solid
mutation led {
  sendCommands(
    module: {name: "EPSM"},
    commands:[
      "SUP:LED ON",
    ]
  ) {command, ok}
}

# set the EPSM's LED back to blinky
mutation led {
  sendCommands(
    module: {name: "EPSM"},
    commands:[
      "SUP:LED FLASH",
    ]
  ) {command, ok}
}

# reset the EPSM (will cut power to the bus!)
mutation epsm_reset {
  sendCommands(
    module: {name: "EPSM"},
    commands:[
      "SUP:RES NOW",
    ]
  ) {command, ok}
}

# query how many seconds are left until the EPSM watchdog times out
query epsm_wdt_seconds_left {
    telemetry(
    module: {name:"EPSM"},
    names: ["wdt_seconds_left_s"]
  )
}

# set the EPSM watchdog period to 180 seconds
mutation set_eps_wdt_timeout {
    sendCommands(
    module: {name: "EPSM"},
    commands:[
      "EPSM:NVM UNLOCK,12345",
      "EPSM:NVM WDT, 180",
      "EPSM:NVM WRITE,1",
      "EPSM:WDT KICK"
    ]
  ) {command, ok}
}

# query some EPSM telemetry
query some_eps_tlm {
    telemetry(
    module: {name:"EPSM"},
    names: [
      "elapsed_time_s",
      "mcu_load",
      "supmcu_mcu_id",
      "5v_converter_data",
      "fgpa_version_number",
      "sns_vusb_mv",
      "firmware_version"
    ]
  )
}

# query all EPSM telemetry
query epsm_telemetry {
    telemetry(module: {name: "EPSM"})
}

# query some GPS telemetry
query some_gps_tlm {
  telemetry(
    module: {address: 81},
    names: [
      "scpi_cmds_processed",
      "supmcu_mcu_id",
      "status_pv",
      "combined_telemetry",
      "firmware_version",
    ],
  )
}

# query all GPS telemetry
query gpsrm_telemetry {
    telemetry(module: {name: "GPS"})
}

# set the response delay for telemetry from a particular module
mutation set_epsm_response_delay {
  responseDelay(
    module: {name: "EPSM"},
    delay: 0.08,
  ) {ok}
}

# set boot target to EMMC (BBB only!)
mutation boot_emmc {
  bootTarget(bootTarget: EMMC)
}

# set boot target to uSD (BBB only!)
mutation boot_usd {
  bootTarget(bootTarget: USD)
}

# get current boot target (BBB only!)
query boot_target {
  bootTarget
}

# Update GPSRM firmware (optional feature!)
mutation update_gps {
  updateFirmware(
    hexfile: "/home/root/hexes/fw/512/GPSRM1_512MC206_RevD_fw-1.3.5a-2.0.0a.hex"
    module: {name: "GPS"},
  ) {ok}
}
```