National Aeronautics and
Space Administration

Benefits of using Electronic Data Sheets (EDS) with coreFlight Systems (cFS) – A Project Example

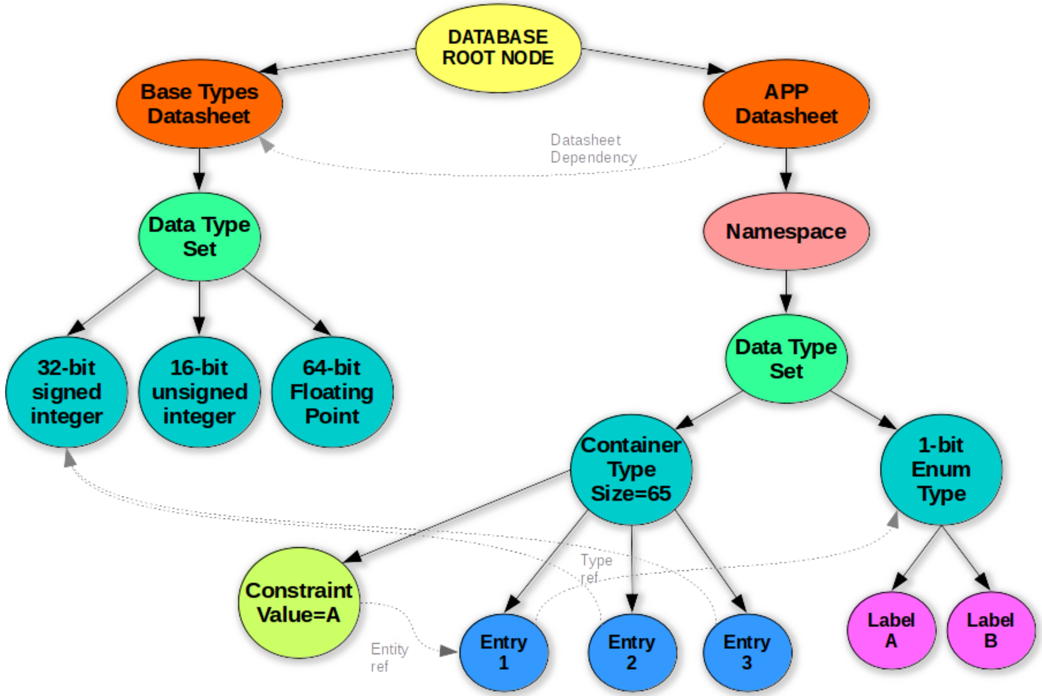
Mathew McCaskey (HX5, NASA GRC) | Regenerative Fuel Cell (RFC) Project
Space Technology Mission Directorate Game Changing Development Program

Outline

- **cFS with EDS support overview**
  - EdsLib tool/database
  - Interfacing EdsLib with cFS (CFE_MissionLib database)
- **The Regenerative Fuel Cell (RFC) Project overview**
- **RFC software approach**
  - Operational Program and Hardware Simulator
    - Usage of cFS with EDS support
  - Operator Interface
    - EdsLib/CFE_MissionLib Python bindings
- **cFS-EDS-GroundStation**
  - Telemetry System
  - Telecommand System
- **Demo**

cFS With EDS Support Overview

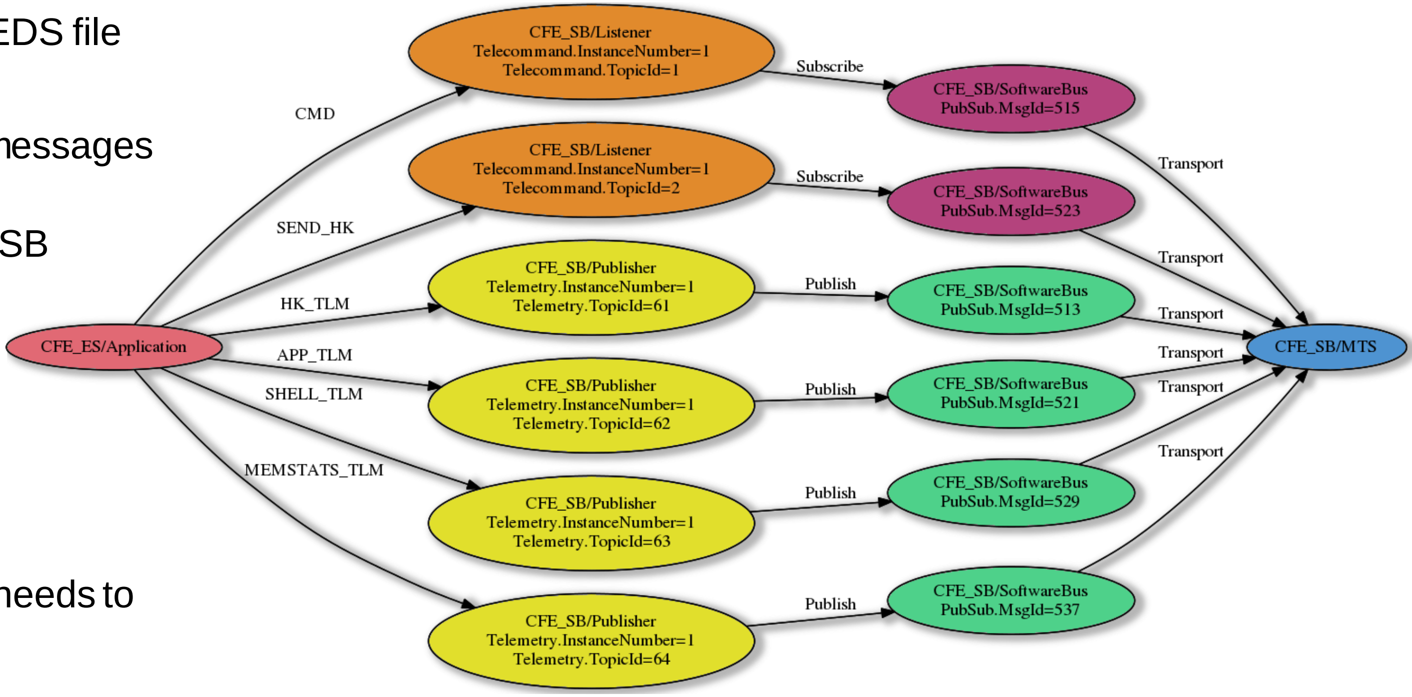
- **Spacecraft Onboard Interface Services (SOIS) EDS XML specifications are defined by CCSDS 867.0-B-1 (Blue Book): https://public.ccsds.org/Pubs/876x0b1.pdf**
  - Specifications for defining constants, data structures, interfaces, constraints, etc. across multiple packages/files
- **EdsLib: https://github.com/nasa/edslib**
  - Contains an implementation of a tool and runtime library for embedded software to create and interpret data structures defined using the XML specifications.
  - Parses the information in the XML files within a given project into a Document Object Model (DOM) tree
    - Creates C header files that contain the data structures defined in the EDS files
    - Generates C database with metadata of the DOM tree
      - Base names, types, structure sizes, number of sub-elements, sub-element information, labels, constraints, etc.
      - Information for packing / unpacking data structures.
  - Runtime library contains API for retrieving EDS metadata and converting between packed and native data structures
  - Bindings for Python, Lua, JSON allow the manipulation of EDS objects within scripts.
  - Application agnostic



Example DOM tree

cFS With EDS Support Overview

- **Additional tools and libraries were developed to interface EdsLib with cFS**
  - The interface specifications of EDS are used to define the telemetry and telecommand interfaces of the cFS Software Bus (SB)
    - Topic IDs for telemetry/telecommand are managed in an EDS file
    - Constraints are used to manage command codes
    - Dispatch tables are generated for applications receiving messages from the SB
    - Alongside the EdsLib database, a C database containing SB interface metadata is also generated (CFE_MissionLib)
      - Interface ID, topic ID, dispatch table, associated EDS object for the SB message
  - Runtime library to get/set header information from telemetry/telecommand messages
    - Conversion between MsgID, TopicID, ApID
    - Customizable for any type of desired header (the header needs to be defined in the EDS files)
  - Tool developed to read a Lua script and generate a binary configuration file that can be read in a cFS instance at runtime
  - https://github.com/jphickey/cfe-eds-framework

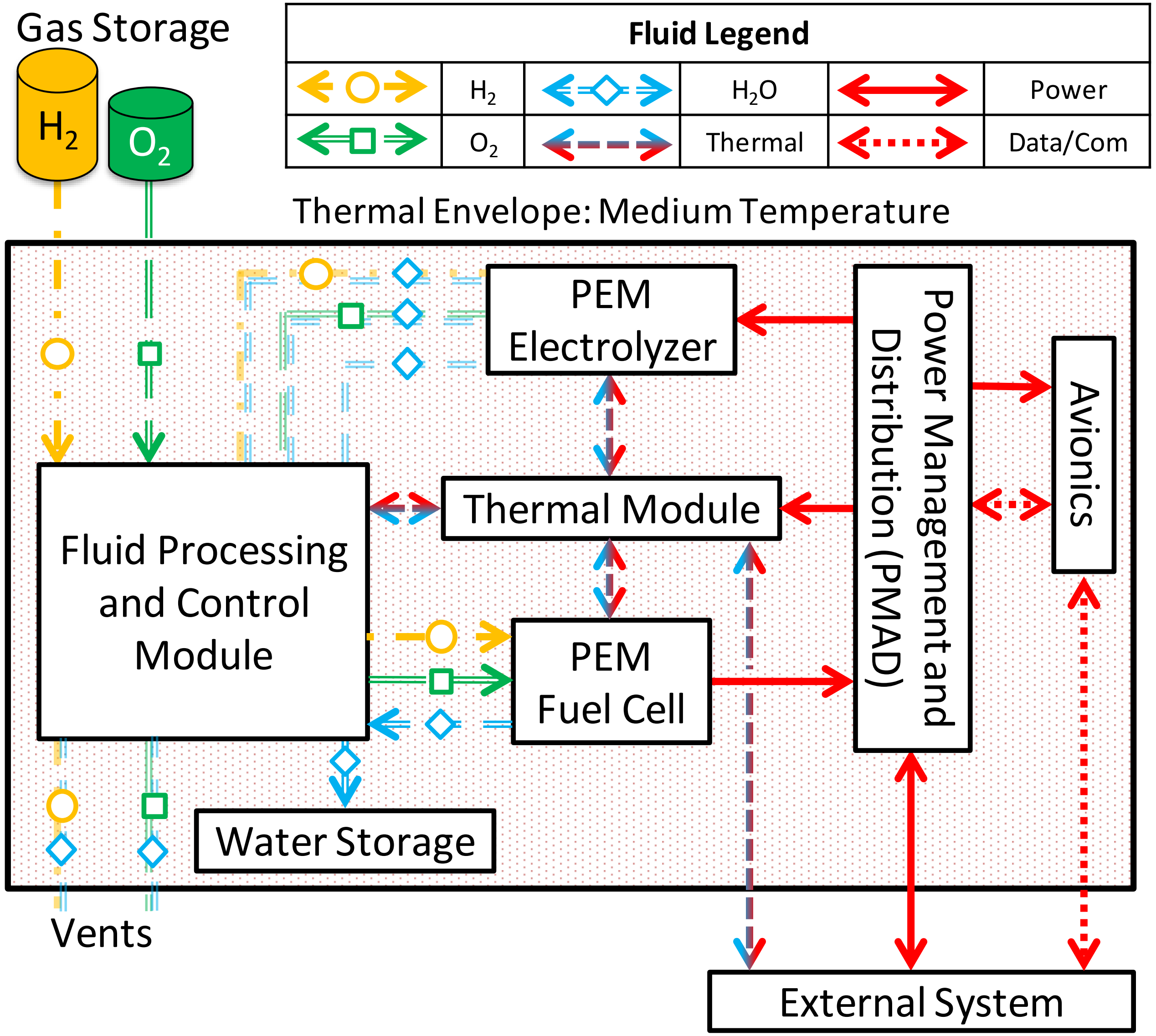


CFE_SB/Listener Telecommand.InstanceNumber=1 Telecommand.TopicId=1
Subscribe
CFE_SB/SoftwareBus PubSub.MsgId=515
CMD
CFE_SB/Listener Telecommand.InstanceNumber=1 Telecommand.TopicId=2
Subscribe
CFE_SB/SoftwareBus PubSub.MsgId=523
Transport
SEND_HK
CFE_SB/Publisher Telemetry.InstanceNumber=1 Telemetry.TopicId=61
Publish
CFE_SB/SoftwareBus PubSub.MsgId=513
Transport
HK_TLM
Transport
CFE_ES/Application
CFE_SB/MTS
APP_TLM
Transport
SHELL_TLM
CFE_SB/Publisher Telemetry.InstanceNumber=1 Telemetry.TopicId=62
Publish
CFE_SB/SoftwareBus PubSub.MsgId=521
Transport
MEMSTATS_TLM
Transport
CFE_SB/Publisher Telemetry.InstanceNumber=1 Telemetry.TopicId=63
Publish
CFE_SB/SoftwareBus PubSub.MsgId=529
CFE_SB/Publisher Telemetry.InstanceNumber=1 Telemetry.TopicId=64
Publish
CFE_SB/SoftwareBus PubSub.MsgId=537

Example Interface Diagram

The Regenerative Fuel Cell (RFC) Project Overview

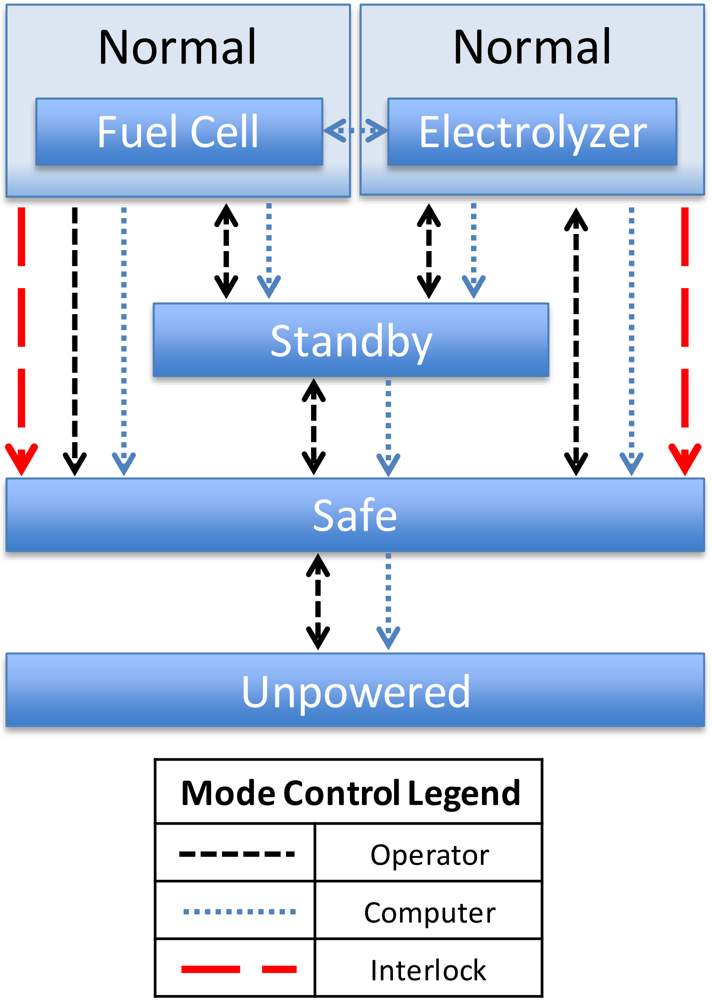
- **A Regenerative Fuel Cell is an energy storage system that utilizes hydrogen and oxygen gases to store energy**
  - Fuel Cell converts gas reactants to electricity and product water
  - Electrolyzer uses electricity (e.g., from a photovoltaic array) to convert water back to gaseous reactants
- **Primary goal of the project is to demonstrate an RFC system within a TVAC chamber for several lunar day/night cycles**
  - At the lunar equator, both the day and night last about 2 weeks
  - For such long durations similarly powered battery systems become prohibitively heavy



RFC Software Approach

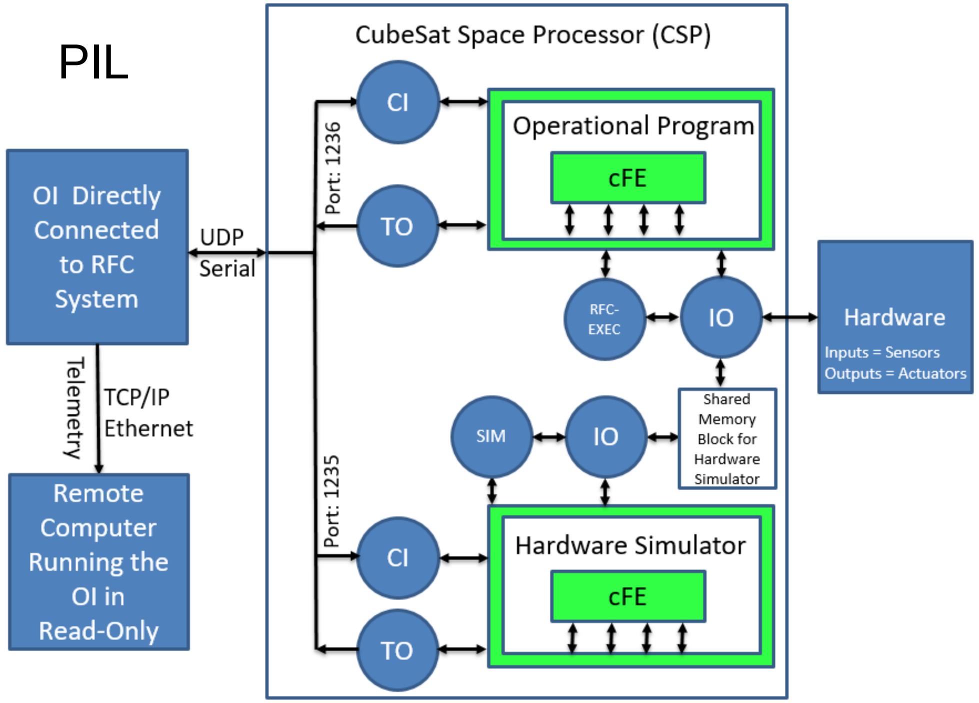
- **The RFC software is responsible for handling the monitoring and control of the RFC system**
  - Manages the state machine as the RFC system transitions through different operational modes
  - Turns on actuators and effectors during state transitions and fault actions
  - Monitors sensors to check if they are within operational limits
    - Caution, Warning, Alarm system is set up to quantify the severity of a fault
  - Perform predefined fault actions based on the sensor and severity of the fault
  - Calculate parameters related to the overall health of the RFC system (e.g., overall system charge)
  - Store operational data in internal data files and send telemetry to a remote operator interface

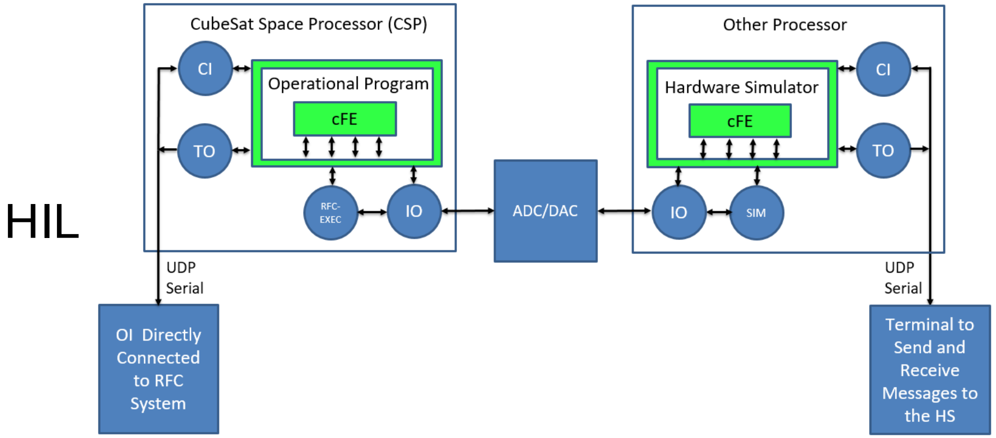
Operational Modes



RFC Software Approach

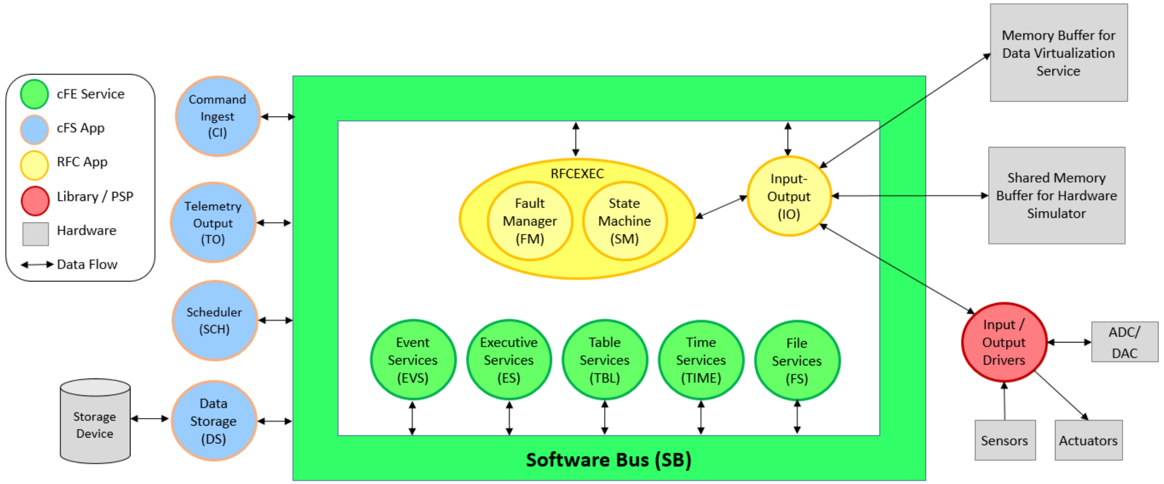
- **Developing three programs:**
- **Operational Program (OP)**
  - Resides on a CubeSat Space Processor (CSP)
  - Manages state machine, fault monitoring, fault actions, telemetry, and data storage
- **Hardware Simulator (HS)**
  - Tool developed to simulate sensor values based on the current state of the actuators/effectors
  - Allows for processor-in-the-loop (PIL) and hardware-in-the-loop (HIL) testing scenarios
- **Operator Interface (OI)**
  - Python based GUI that receives telemetry from the OP, displays it in useful formats for an operator.
  - Download data files stored on the CSP board.



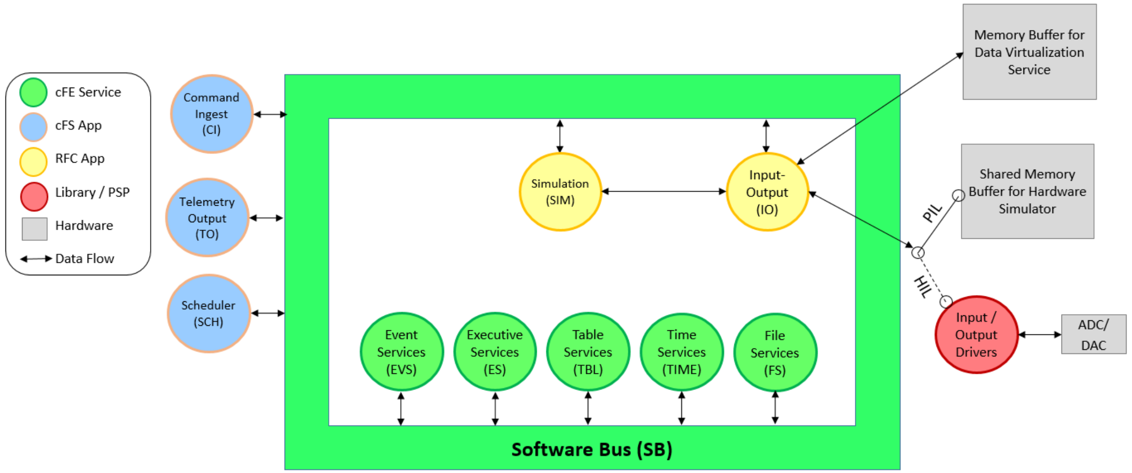

HIL
CubeSat Space Processor (CSP)
CI
TO
Operational Program
cFE
RFC-EXEC
IO
ADC/DAC
Other Processor
Hardware Simulator
cFE
CI
TO
IO
SIM
UDP
Serial
OI Directly Connected to RFC System
UDP
Serial
Terminal to Send and Receive Messages to the HS

Using cFS with EDS Support

- **Both the OP and HS will be based in core-flight with Electronic Data Sheets (EDS) support**
  - cFS has a history of being successfully used on flight projects.
    - The RFC project may eventually transition to a flight project.
  - Before RFC, several GRC projects with similar requirements used cFS w/EDS support
    - Starting point for the RFC code base
  - EDS files become the single source of truth
  - EdsLib simplifies the data communication between different computer architectures
    - Laptop to/from CSP board
  - Bindings allows using EDS objects in scripts
    - Operator Interface (Python)
    - Human readable scripts (Lua) to generate run time configuration files.
    - Functional testing environments (Lua)
  - Community of developers to ask for help.



OP

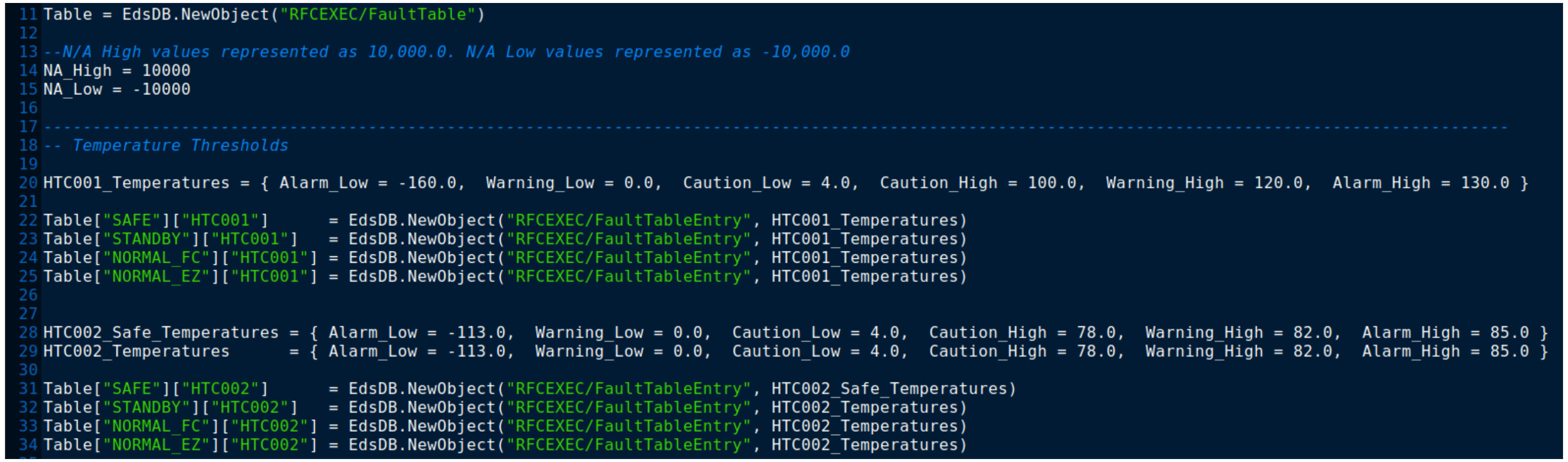


HS

Using cFS with EDS Support

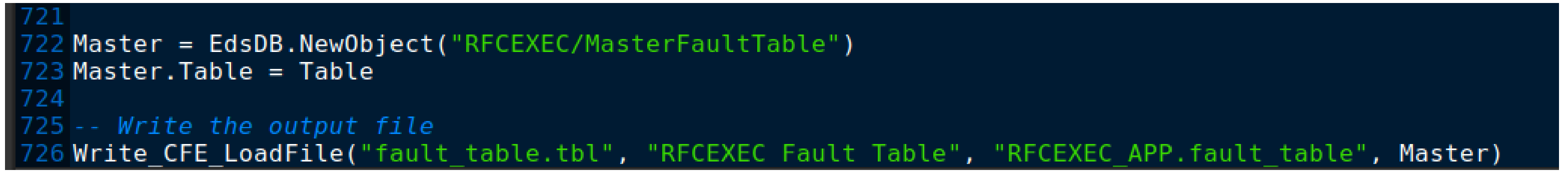
- **Configuration Files**
  - Input/Output App
    - Channel setup
      - type, driver, board/ subchannel location, etc.
    - Multiple files for different setups
      - PIL, HIL, RFC system
  - RFCEXEC App
    - Fault Table
    - Fault Action Table
    - Output Checking Table
  - Data Storage App
    - Message ID management
    - Message ID filtering
    - File management

op_fault_table.lua



```lua
Table = EdsDB.NewObject("RFCEXEC/FaultTable")

--N/A High values represented as 10,000.0. N/A Low values represented as -10,000.0
NA_High = 10000
NA_Low = -10000

-------------------------------------------------------------------------------------------------------------------------------------------------
-- Temperature Thresholds

HTC001_Temperatures = { Alarm_Low = -160.0,  Warning_Low = 0.0,  Caution_Low = 4.0,  Caution_High = 100.0,  Warning_High = 120.0,  Alarm_High = 130.0 }

Table["SAFE"]["HTC001"]      = EdsDB.NewObject("RFCEXEC/FaultTableEntry", HTC001_Temperatures)
Table["STANDBY"]["HTC001"]   = EdsDB.NewObject("RFCEXEC/FaultTableEntry", HTC001_Temperatures)
Table["NORMAL_FC"]["HTC001"] = EdsDB.NewObject("RFCEXEC/FaultTableEntry", HTC001_Temperatures)
Table["NORMAL_EZ"]["HTC001"] = EdsDB.NewObject("RFCEXEC/FaultTableEntry", HTC001_Temperatures)


HTC002_Safe_Temperatures = { Alarm_Low = -113.0,  Warning_Low = 0.0,  Caution_Low = 4.0,  Caution_High = 78.0,  Warning_High = 82.0,  Alarm_High = 85.0 }
HTC002_Temperatures      = { Alarm_Low = -113.0,  Warning_Low = 0.0,  Caution_Low = 4.0,  Caution_High = 78.0,  Warning_High = 82.0,  Alarm_High = 85.0 }

Table["SAFE"]["HTC002"]      = EdsDB.NewObject("RFCEXEC/FaultTableEntry", HTC002_Safe_Temperatures)
Table["STANDBY"]["HTC002"]   = EdsDB.NewObject("RFCEXEC/FaultTableEntry", HTC002_Temperatures)
Table["NORMAL_FC"]["HTC002"] = EdsDB.NewObject("RFCEXEC/FaultTableEntry", HTC002_Temperatures)
Table["NORMAL_EZ"]["HTC002"] = EdsDB.NewObject("RFCEXEC/FaultTableEntry", HTC002_Temperatures)
```

...

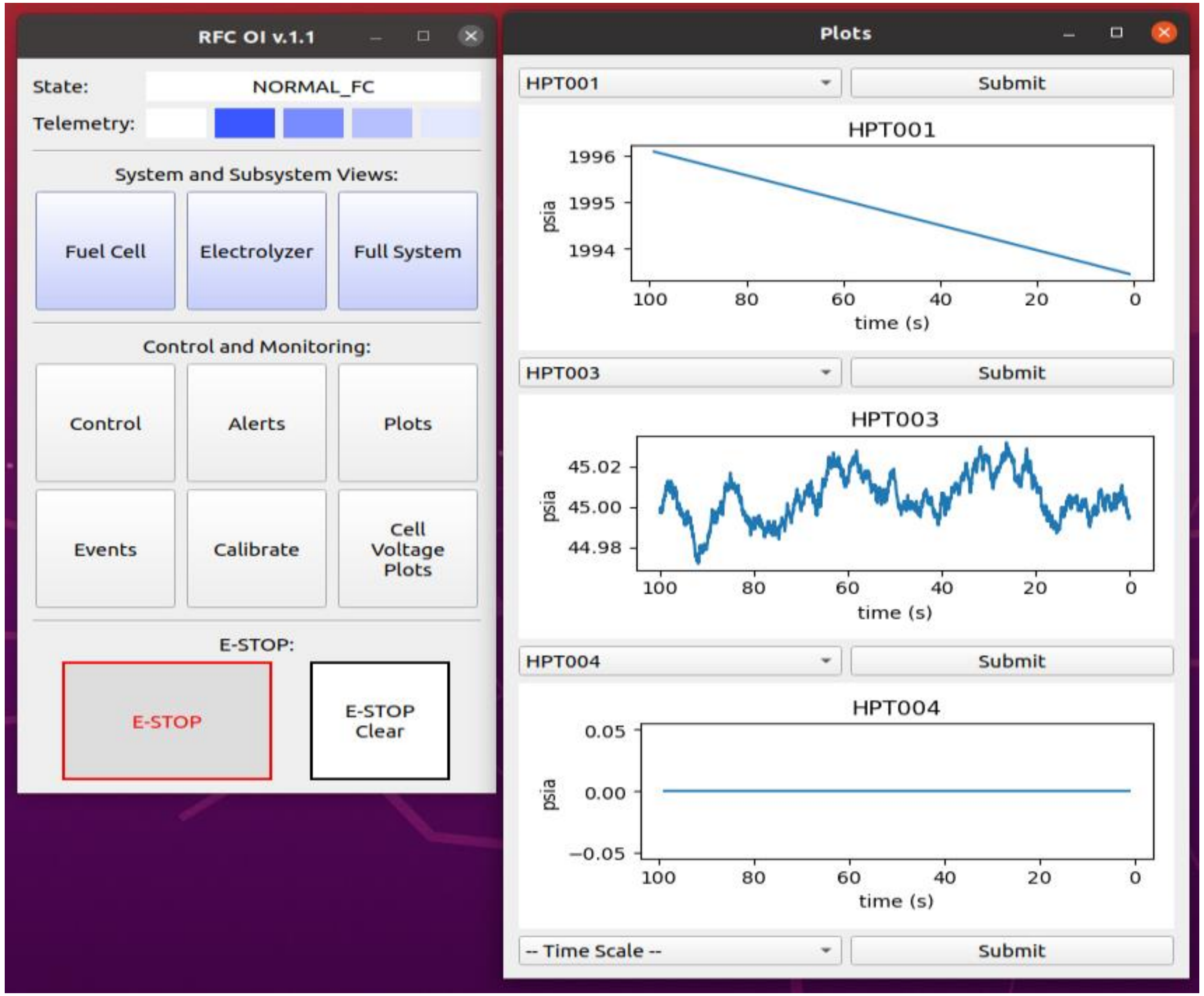


```lua

Master = EdsDB.NewObject("RFCEXEC/MasterFaultTable")
Master.Table = Table

-- Write the output file
Write_CFE_LoadFile("fault_table.tbl", "RFCEXEC Fault Table", "RFCEXEC_APP.fault_table", Master)
```

Generates the file /cf/fault_table.tbl that contains the fault table in a packed binary format with appropriate CFE_FS and CFE_TBL headers

Using cFS with EDS Support

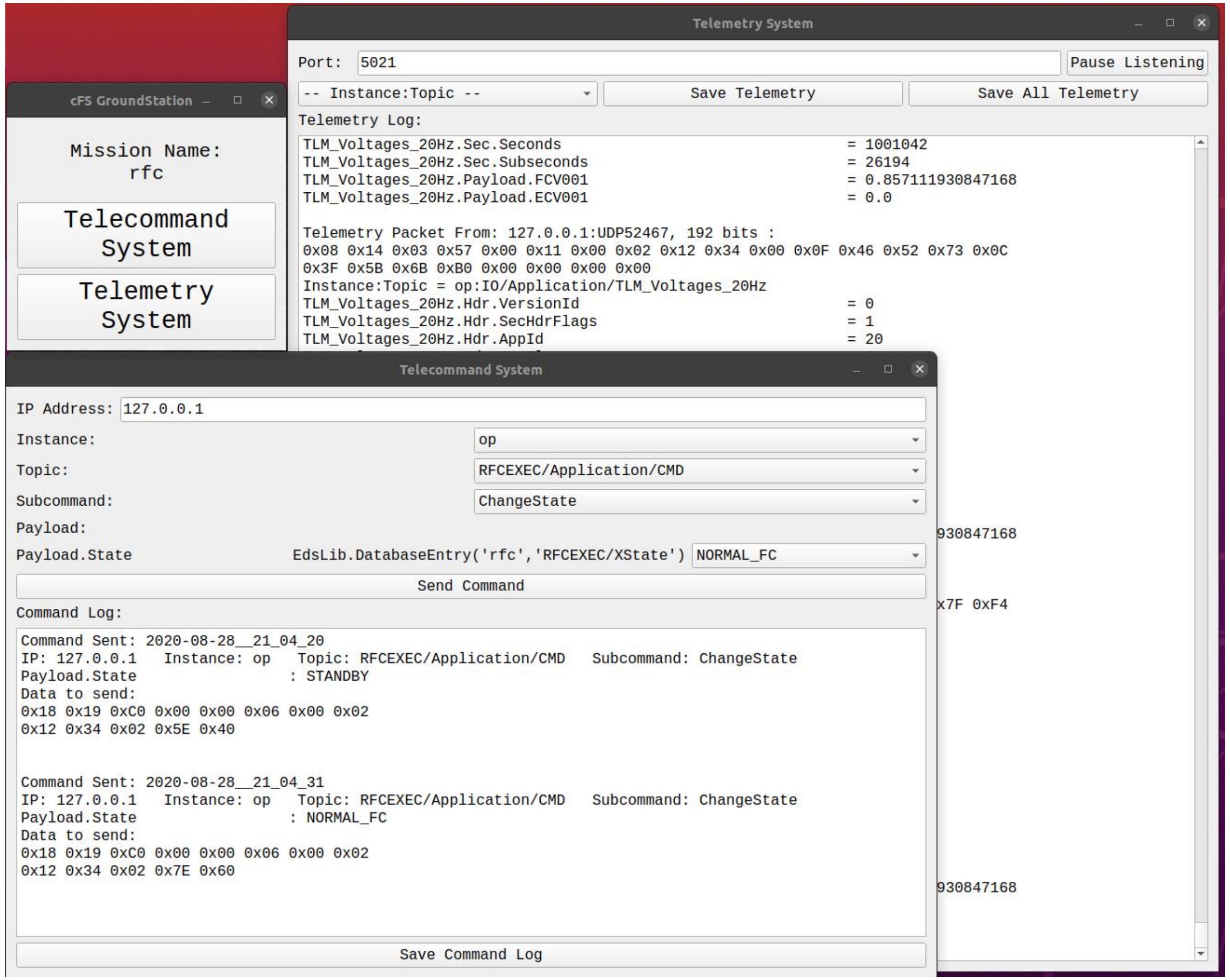
- **Python Bindings**
  - Create and use EDS objects within python
  - The RFC project developed additional bindings
    - Iterators for containers and enumerations to extract entry labels
    - Iterators for cFS instances, topics, and subcommands
    - Methods to extract instance and topic information from telemetry SB message
    - Methods to set Publish/Subscribe parameters for telecommand SB message
- **Operator Interface**
  - Using the bindings we developed a python-based GUI
    - Send Commands
    - Receive/Decode Telemetry
    - Display telemetry in useful formats for an RFC operator



State: NORMAL_FC

cFS-EDS-GroundStation Demo

- **In the process of developing the OI we created a generic python-based GUI that interfaces with any instance of cFS with EDS support**
  - Useful tool for quick command/telemetry checking
  - Telemetry System:
    - Automatically decodes telemetry messages, saves the raw messages in internal arrays, and displays the information in a telemetry log
    - Listening to telemetry messages can be paused and un-paused
    - Telemetry messages can be saved to time stamped binary files on a type-by-type basis or all at once
      - Messages are labeled by a “Instance:Topic” identifier
  - Telecommand system:
    - Dropdown menus for instance, topic, and subcommand (if available)
    - If any chosen topic or subcommand contains a payload, entry fields will be created
      - Dropdown menus for Enumeration labels
      - Text entries otherwise
    - Payload entries are checked, then the command message is packed and sent
- **Python files are configured with the mission name during the cFS build process**
  - Everything is read from EdsLib/CFE_MissionLib databases
- https://github.com/nasa/cFS-EDS-GroundStation



Mission Name:
rfc

Port: 5021

Telemetry Log:

```
TLM_Voltages_20Hz.Sec.Seconds                                = 1001042
TLM_Voltages_20Hz.Sec.Subseconds                             = 26194
TLM_Voltages_20Hz.Payload.FCV001                             = 0.857111930847168
TLM_Voltages_20Hz.Payload.ECV001                             = 0.0

Telemetry Packet From: 127.0.0.1:UDP52467, 192 bits :
0x08 0x14 0x03 0x57 0x00 0x11 0x00 0x02 0x12 0x34 0x00 0x0F 0x46 0x52 0x73 0x0C
0x3F 0x5B 0x6B 0xB0 0x00 0x00 0x00 0x00
Instance:Topic = op:IO/Application/TLM_Voltages_20Hz
TLM_Voltages_20Hz.Hdr.VersionId                              = 0
TLM_Voltages_20Hz.Hdr.SecHdrFlags                            = 1
TLM_Voltages_20Hz.Hdr.AppId                                  = 20
```

IP Address: 127.0.0.1

Instance: op

Topic: RFCEXEC/Application/CMD

Subcommand: ChangeState

Payload:

Payload.State EdsLib.DatabaseEntry('rfc','RFCEXEC/XState') NORMAL_FC

Command Log:

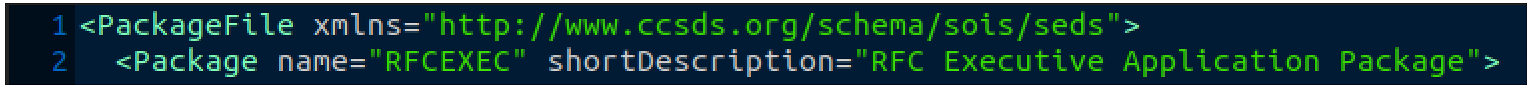
```
Command Sent: 2020-08-28__21_04_20
IP: 127.0.0.1   Instance: op   Topic: RFCEXEC/Application/CMD   Subcommand: ChangeState
Payload.State                    : STANDBY
Data to send:
0x18 0x19 0xC0 0x00 0x00 0x06 0x00 0x02
0x12 0x34 0x02 0x5E 0x40


Command Sent: 2020-08-28__21_04_31
IP: 127.0.0.1   Instance: op   Topic: RFCEXEC/Application/CMD   Subcommand: ChangeState
Payload.State                    : NORMAL_FC
Data to send:
0x18 0x19 0xC0 0x00 0x00 0x06 0x00 0x02
0x12 0x34 0x02 0x7E 0x60
```

Thank you!

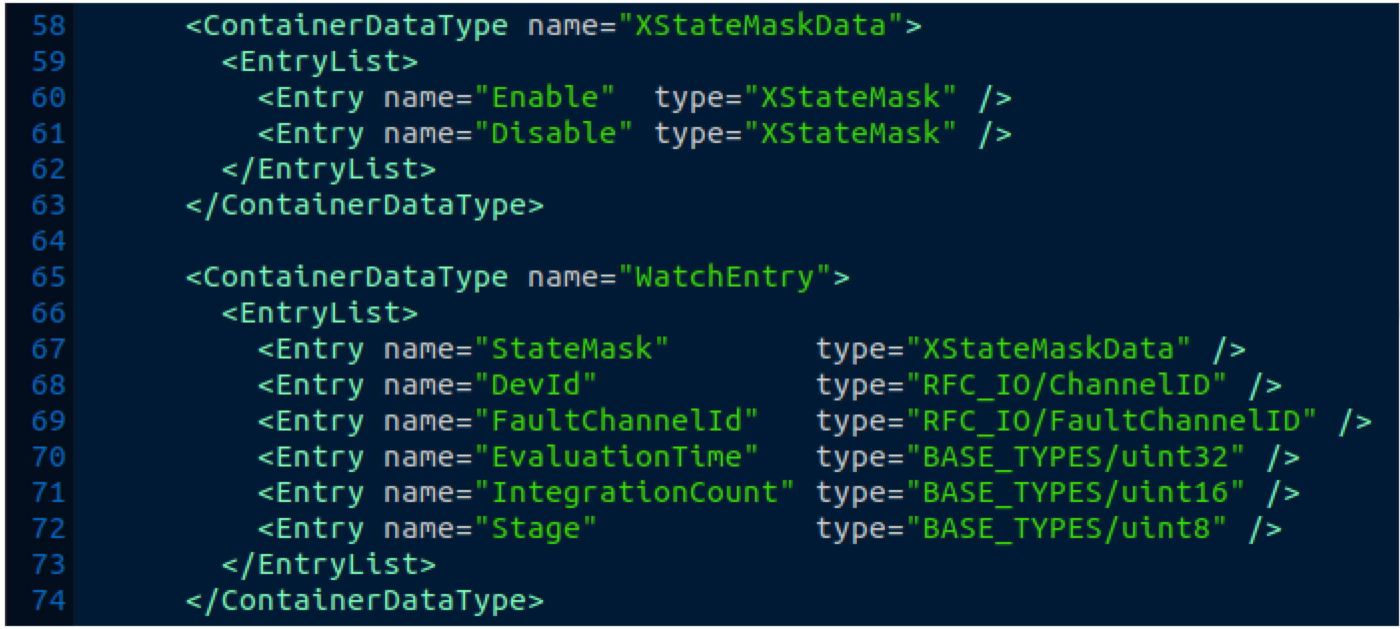
Backup Slides

Using EDS files

- **Example EDS file**
  - Snippet defines a container (C struct)
  - Application components are referenced with a consistent naming format: “Package/Entry”
    - Components defined within the same package only needs “Entry” for the type.



```
<PackageFile xmlns="http://www.ccsds.org/schema/sois/seds">
  <Package name="RFCEXEC" shortDescription="RFC Executive Application Package">
```

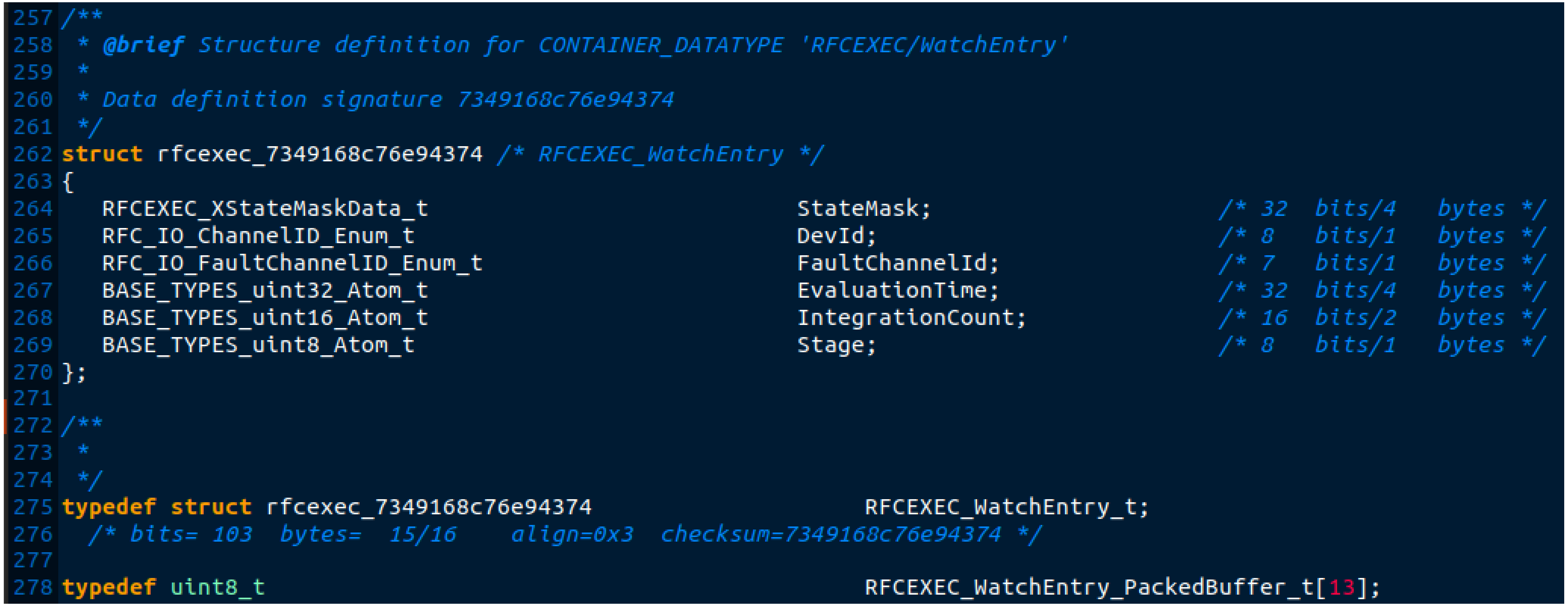


```
      <ContainerDataType name="XStateMaskData">
        <EntryList>
          <Entry name="Enable"  type="XStateMask" />
          <Entry name="Disable" type="XStateMask" />
        </EntryList>
      </ContainerDataType>

      <ContainerDataType name="WatchEntry">
        <EntryList>
          <Entry name="StateMask"        type="XStateMaskData" />
          <Entry name="DevId"            type="RFC_IO/ChannelID" />
          <Entry name="FaultChannelId"   type="RFC_IO/FaultChannelID" />
          <Entry name="EvaluationTime"   type="BASE_TYPES/uint32" />
          <Entry name="IntegrationCount" type="BASE_TYPES/uint16" />
          <Entry name="Stage"            type="BASE_TYPES/uint8" />
        </EntryList>
      </ContainerDataType>
```

rfcexec.xml

Generated Header File

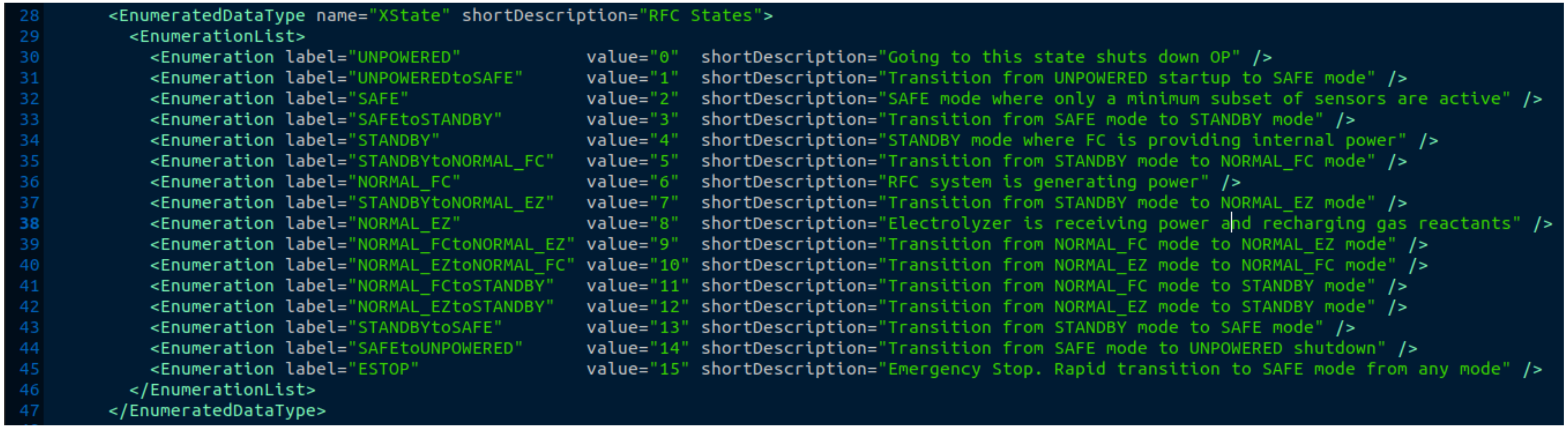
- **Generated type definition header file: rfcexec_eds_typedefs.h**
  - A structure is defined containing all the sub-elements identified in the EDS file
  - A buffer array is defined that can hold the packed structure



```
/**
 * @brief Structure definition for CONTAINER_DATATYPE 'RFCEXEC/WatchEntry'
 *
 * Data definition signature 7349168c76e94374
 */
struct rfcexec_7349168c76e94374 /* RFCEXEC_WatchEntry */
{
   RFCEXEC_XStateMaskData_t                           StateMask;                     /* 32  bits/4   bytes */
   RFC_IO_ChannelID_Enum_t                            DevId;                         /* 8   bits/1   bytes */
   RFC_IO_FaultChannelID_Enum_t                       FaultChannelId;                /* 7   bits/1   bytes */
   BASE_TYPES_uint32_Atom_t                           EvaluationTime;                /* 32  bits/4   bytes */
   BASE_TYPES_uint16_Atom_t                           IntegrationCount;              /* 16  bits/2   bytes */
   BASE_TYPES_uint8_Atom_t                            Stage;                         /* 8   bits/1   bytes */
};

/**
 *
 */
typedef struct rfcexec_7349168c76e94374                    RFCEXEC_WatchEntry_t;
  /* bits= 103  bytes=  15/16    align=0x3  checksum=7349168c76e94374 */

typedef uint8_t                                            RFCEXEC_WatchEntry_PackedBuffer_t[13];
```

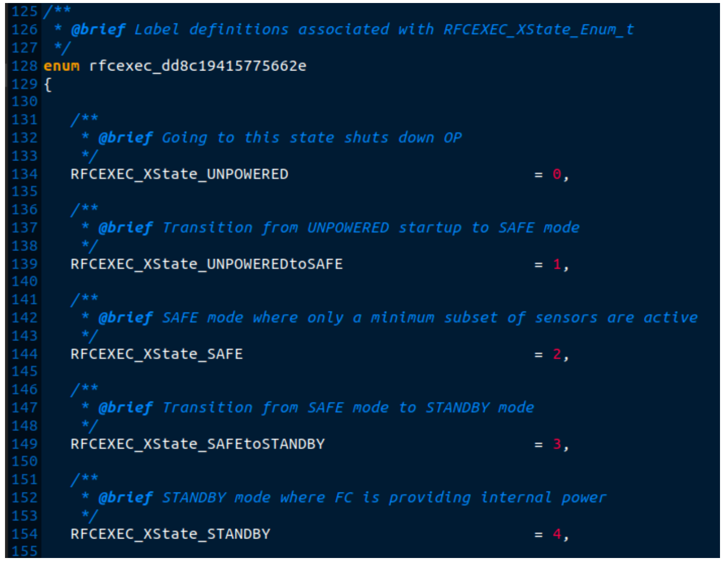
Enumerations

- Similar setup for Enumerations:

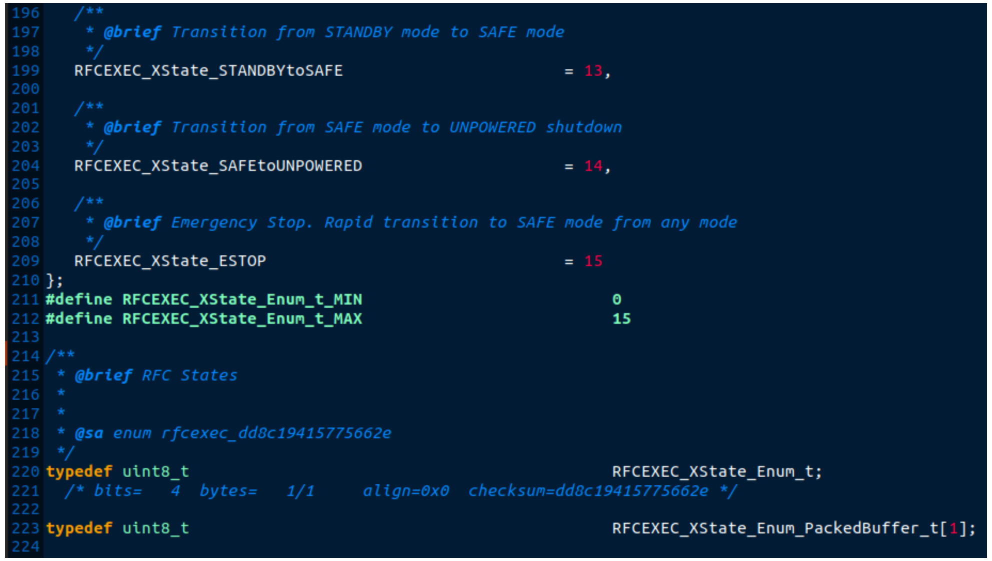


```xml
    <EnumeratedDataType name="XState" shortDescription="RFC States">
      <EnumerationList>
        <Enumeration label="UNPOWERED"            value="0"  shortDescription="Going to this state shuts down OP" />
        <Enumeration label="UNPOWEREDtoSAFE"      value="1"  shortDescription="Transition from UNPOWERED startup to SAFE mode" />
        <Enumeration label="SAFE"                 value="2"  shortDescription="SAFE mode where only a minimum subset of sensors are active" />
        <Enumeration label="SAFEtoSTANDBY"        value="3"  shortDescription="Transition from SAFE mode to STANDBY mode" />
        <Enumeration label="STANDBY"              value="4"  shortDescription="STANDBY mode where FC is providing internal power" />
        <Enumeration label="STANDBYtoNORMAL_FC"   value="5"  shortDescription="Transition from STANDBY mode to NORMAL_FC mode" />
        <Enumeration label="NORMAL_FC"            value="6"  shortDescription="RFC system is generating power" />
        <Enumeration label="STANDBYtoNORMAL_EZ"   value="7"  shortDescription="Transition from STANDBY mode to NORMAL_EZ mode" />
        <Enumeration label="NORMAL_EZ"            value="8"  shortDescription="Electrolyzer is receiving power and recharging gas reactants" />
        <Enumeration label="NORMAL_FCtoNORMAL_EZ" value="9"  shortDescription="Transition from NORMAL_FC mode to NORMAL_EZ mode" />
        <Enumeration label="NORMAL_EZtoNORMAL_FC" value="10" shortDescription="Transition from NORMAL_EZ mode to NORMAL_FC mode" />
        <Enumeration label="NORMAL_FCtoSTANDBY"   value="11" shortDescription="Transition from NORMAL_FC mode to STANDBY mode" />
        <Enumeration label="NORMAL_EZtoSTANDBY"   value="12" shortDescription="Transition from NORMAL_EZ mode to STANDBY mode" />
        <Enumeration label="STANDBYtoSAFE"        value="13" shortDescription="Transition from STANDBY mode to SAFE mode" />
        <Enumeration label="SAFEtoUNPOWERED"      value="14" shortDescription="Transition from SAFE mode to UNPOWERED shutdown" />
        <Enumeration label="ESTOP"                value="15" shortDescription="Emergency Stop. Rapid transition to SAFE mode from any mode" />
      </EnumerationList>
    </EnumeratedDataType>
```

rfcexec.xml



```c
/**
 * @brief Label definitions associated with RFCEXEC_XState_Enum_t
 */
enum rfcexec_dd8c19415775662e
{

   /**
    * @brief Going to this state shuts down OP
    */
   RFCEXEC_XState_UNPOWERED                                 = 0,

   /**
    * @brief Transition from UNPOWERED startup to SAFE mode
    */
   RFCEXEC_XState_UNPOWEREDtoSAFE                           = 1,

   /**
    * @brief SAFE mode where only a minimum subset of sensors are active
    */
   RFCEXEC_XState_SAFE                                      = 2,

   /**
    * @brief Transition from SAFE mode to STANDBY mode
    */
   RFCEXEC_XState_SAFEtoSTANDBY                             = 3,

   /**
    * @brief STANDBY mode where FC is providing internal power
    */
   RFCEXEC_XState_STANDBY                                   = 4,

```

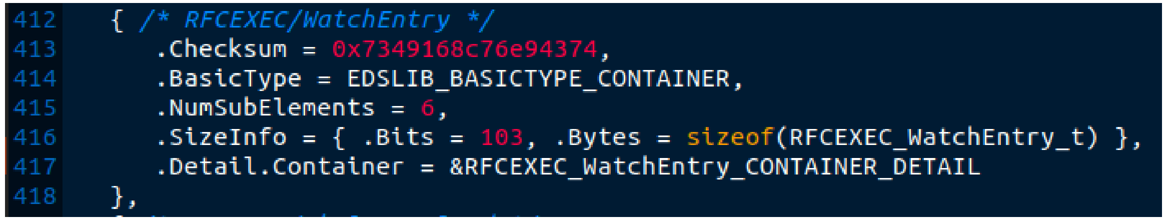


```c
   /**
    * @brief Transition from STANDBY mode to SAFE mode
    */
   RFCEXEC_XState_STANDBYtoSAFE                             = 13,

   /**
    * @brief Transition from SAFE mode to UNPOWERED shutdown
    */
   RFCEXEC_XState_SAFEtoUNPOWERED                           = 14,

   /**
    * @brief Emergency Stop. Rapid transition to SAFE mode from any mode
    */
   RFCEXEC_XState_ESTOP                                     = 15
};
#define RFCEXEC_XState_Enum_t_MIN                                   0
#define RFCEXEC_XState_Enum_t_MAX                                   15

/**
 * @brief RFC States
 *
 *
 * @sa enum rfcexec_dd8c19415775662e
 */
typedef uint8_t                                             RFCEXEC_XState_Enum_t;
  /* bits=   4  bytes=   1/1     align=0x0  checksum=dd8c19415775662e */

typedef uint8_t                                             RFCEXEC_XState_Enum_PackedBuffer_t[1];

```

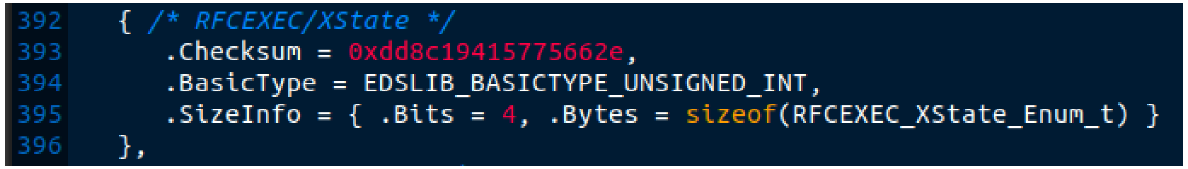
rfcexec_eds_typedefs.h

EdsLib Database

- **The EdsLib database is automatically generated from the EDS files**
  - Base names, types, structure sizes, number of sub-elements, sub-element information, labels, etc.
  - Information for packing / unpacking data structures.
- **EdsLib is a library that combines the EDS Database with API functions that read through the datatype and display information.**

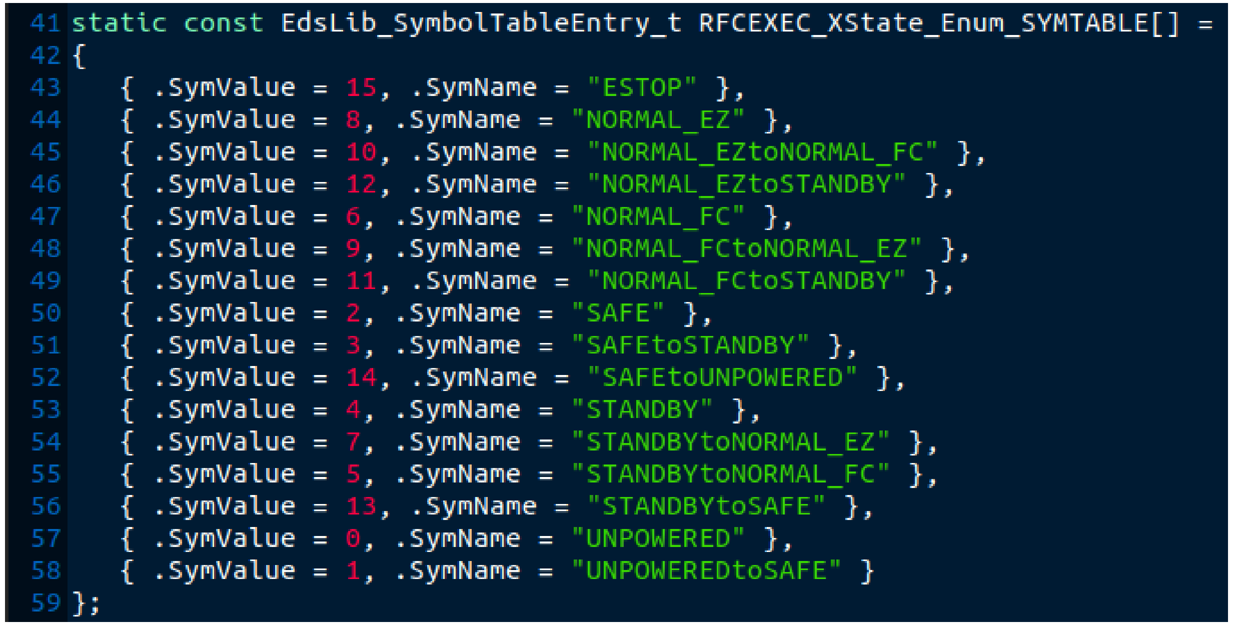


```
    { /* RFCEXEC/WatchEntry */
        .Checksum = 0x7349168c76e94374,
        .BasicType = EDSLIB_BASICTYPE_CONTAINER,
        .NumSubElements = 6,
        .SizeInfo = { .Bits = 103, .Bytes = sizeof(RFCEXEC_WatchEntry_t) },
        .Detail.Container = &RFCEXEC_WatchEntry_CONTAINER_DETAIL
    },
```



```
    { /* RFCEXEC/XState */
        .Checksum = 0xdd8c19415775662e,
        .BasicType = EDSLIB_BASICTYPE_UNSIGNED_INT,
        .SizeInfo = { .Bits = 4, .Bytes = sizeof(RFCEXEC_XState_Enum_t) }
    },
```

rfcexec_eds_datatypedb_impl.c

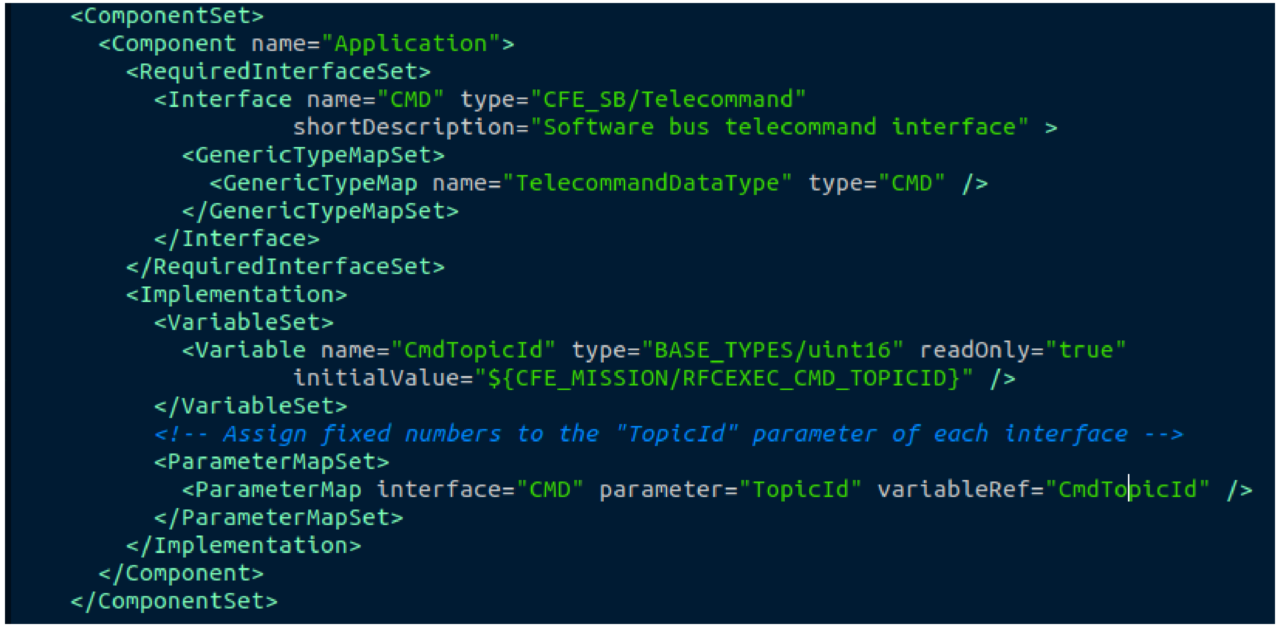


```
static const EdsLib_SymbolTableEntry_t RFCEXEC_XState_Enum_SYMTABLE[] =
{
   { .SymValue = 15, .SymName = "ESTOP" },
   { .SymValue = 8, .SymName = "NORMAL_EZ" },
   { .SymValue = 10, .SymName = "NORMAL_EZtoNORMAL_FC" },
   { .SymValue = 12, .SymName = "NORMAL_EZtoSTANDBY" },
   { .SymValue = 6, .SymName = "NORMAL_FC" },
   { .SymValue = 9, .SymName = "NORMAL_FCtoNORMAL_EZ" },
   { .SymValue = 11, .SymName = "NORMAL_FCtoSTANDBY" },
   { .SymValue = 2, .SymName = "SAFE" },
   { .SymValue = 3, .SymName = "SAFEtoSTANDBY" },
   { .SymValue = 14, .SymName = "SAFEtoUNPOWERED" },
   { .SymValue = 4, .SymName = "STANDBY" },
   { .SymValue = 7, .SymName = "STANDBYtoNORMAL_EZ" },
   { .SymValue = 5, .SymName = "STANDBYtoNORMAL_FC" },
   { .SymValue = 13, .SymName = "STANDBYtoSAFE" },
   { .SymValue = 0, .SymName = "UNPOWERED" },
   { .SymValue = 1, .SymName = "UNPOWEREDtoSAFE" }
};
```

rfcexec_eds_displaydb_impl.c

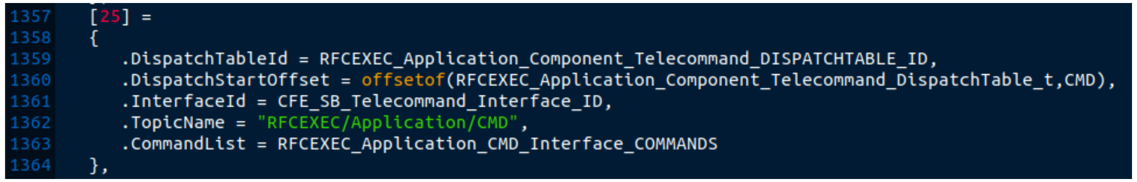
CFE_MissionLib

- **CFE_MissionLib is a library that handles the interface between EDS and cFS (specifically CFE_SB)**
  - Interface database: Telemetry/Telecommand topics with their associated message types (from EdsLib)
  - Contains and API functions to read database information
  - Customizable to the mission based on the CCSDS message header types used.



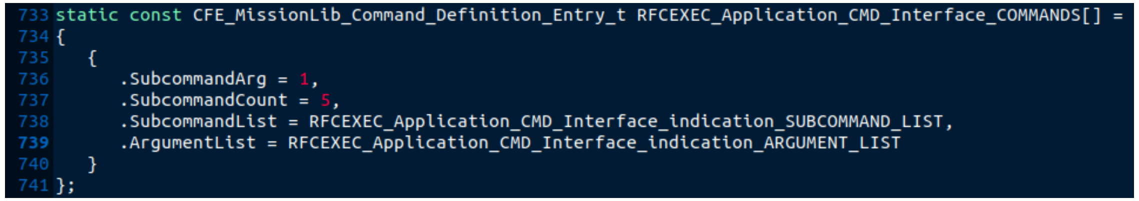
```xml
<ComponentSet>
  <Component name="Application">
    <RequiredInterfaceSet>
      <Interface name="CMD" type="CFE_SB/Telecommand"
                 shortDescription="Software bus telecommand interface" >
        <GenericTypeMapSet>
          <GenericTypeMap name="TelecommandDataType" type="CMD" />
        </GenericTypeMapSet>
      </Interface>
    </RequiredInterfaceSet>
    <Implementation>
      <VariableSet>
        <Variable name="CmdTopicId" type="BASE_TYPES/uint16" readOnly="true"
                  initialValue="${CFE_MISSION/RFCEXEC_CMD_TOPICID}" />
      </VariableSet>
      <!-- Assign fixed numbers to the "TopicId" parameter of each interface -->
      <ParameterMapSet>
        <ParameterMap interface="CMD" parameter="TopicId" variableRef="CmdTopicId" />
      </ParameterMapSet>
    </Implementation>
  </Component>
</ComponentSet>
```

rfcexec.xml



```c
    [25] =
    {
        .DispatchTableId = RFCEXEC_Application_Component_Telecommand_DISPATCHTABLE_ID,
        .DispatchStartOffset = offsetof(RFCEXEC_Application_Component_Telecommand_DispatchTable_t,CMD),
        .InterfaceId = CFE_SB_Telecommand_Interface_ID,
        .TopicName = "RFCEXEC/Application/CMD",
        .CommandList = RFCEXEC_Application_CMD_Interface_COMMANDS
    },
```

Topic Listing



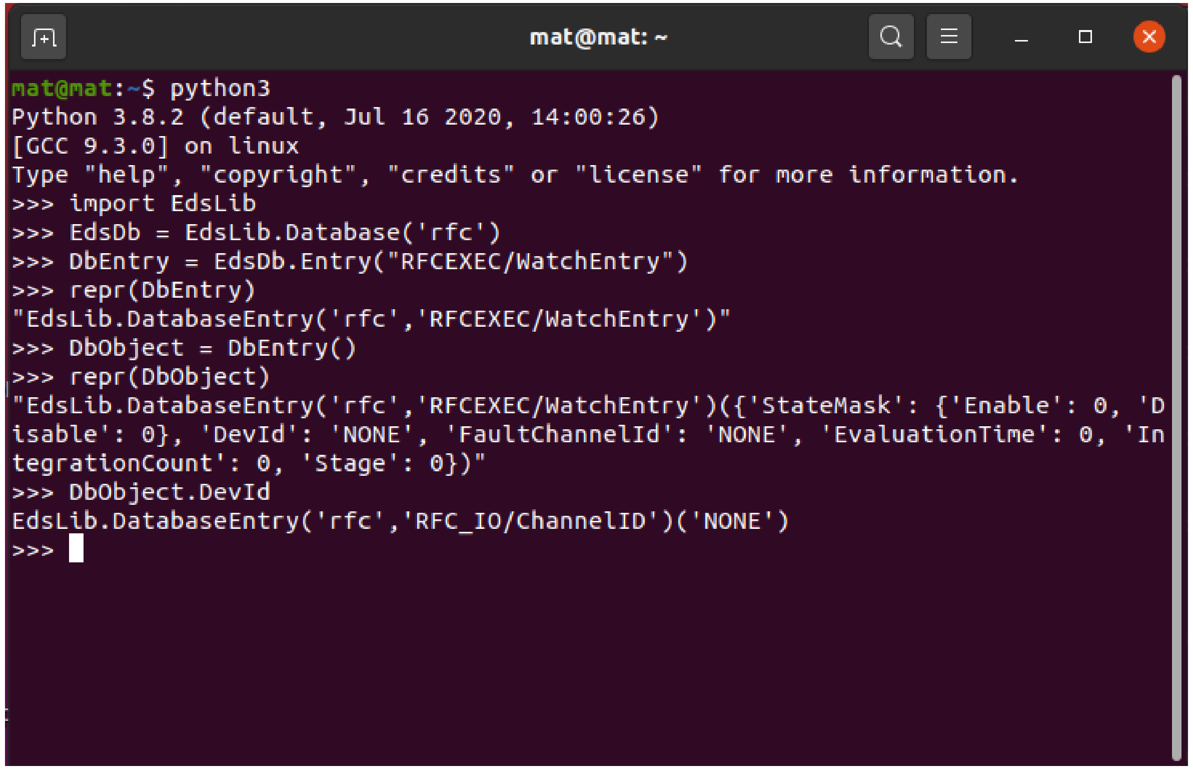
```c
static const CFE_MissionLib_Command_Definition_Entry_t RFCEXEC_Application_CMD_Interface_COMMANDS[] =
{
    {
        .SubcommandArg = 1,
        .SubcommandCount = 5,
        .SubcommandList = RFCEXEC_Application_CMD_Interface_indication_SUBCOMMAND_LIST,
        .ArgumentList = RFCEXEC_Application_CMD_Interface_indication_ARGUMENT_LIST
    }
};
```

Subcommand Information

rfc_eds_interfacedb_impl.c

Python Bindings

- **The Operator Interface CSCI needed a way to access the information in EdsLib and CFE_MissionLib from python**
  - Python bindings were created for EdsLib and CFE_MissionLib to create EDS objects in python

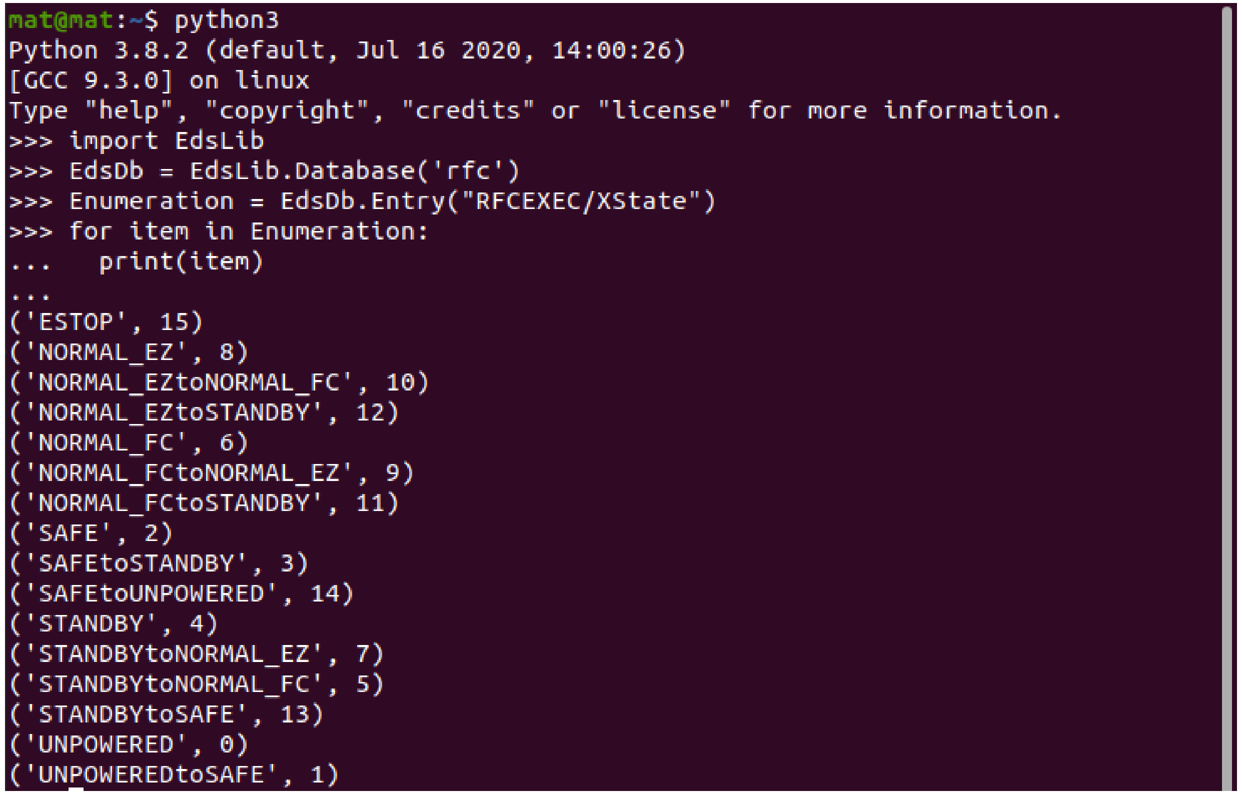


```
mat@mat:~$ python3
Python 3.8.2 (default, Jul 16 2020, 14:00:26)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import EdsLib
>>> EdsDb = EdsLib.Database('rfc')
>>> DbEntry = EdsDb.Entry("RFCEXEC/WatchEntry")
>>> repr(DbEntry)
"EdsLib.DatabaseEntry('rfc','RFCEXEC/WatchEntry')"
>>> DbObject = DbEntry()
>>> repr(DbObject)
"EdsLib.DatabaseEntry('rfc','RFCEXEC/WatchEntry')({'StateMask': {'Enable': 0, 'D
isable': 0}, 'DevId': 'NONE', 'FaultChannelId': 'NONE', 'EvaluationTime': 0, 'In
tegrationCount': 0, 'Stage': 0})"
>>> DbObject.DevId
EdsLib.DatabaseEntry('rfc','RFC_IO/ChannelID')('NONE')
>>> 
```

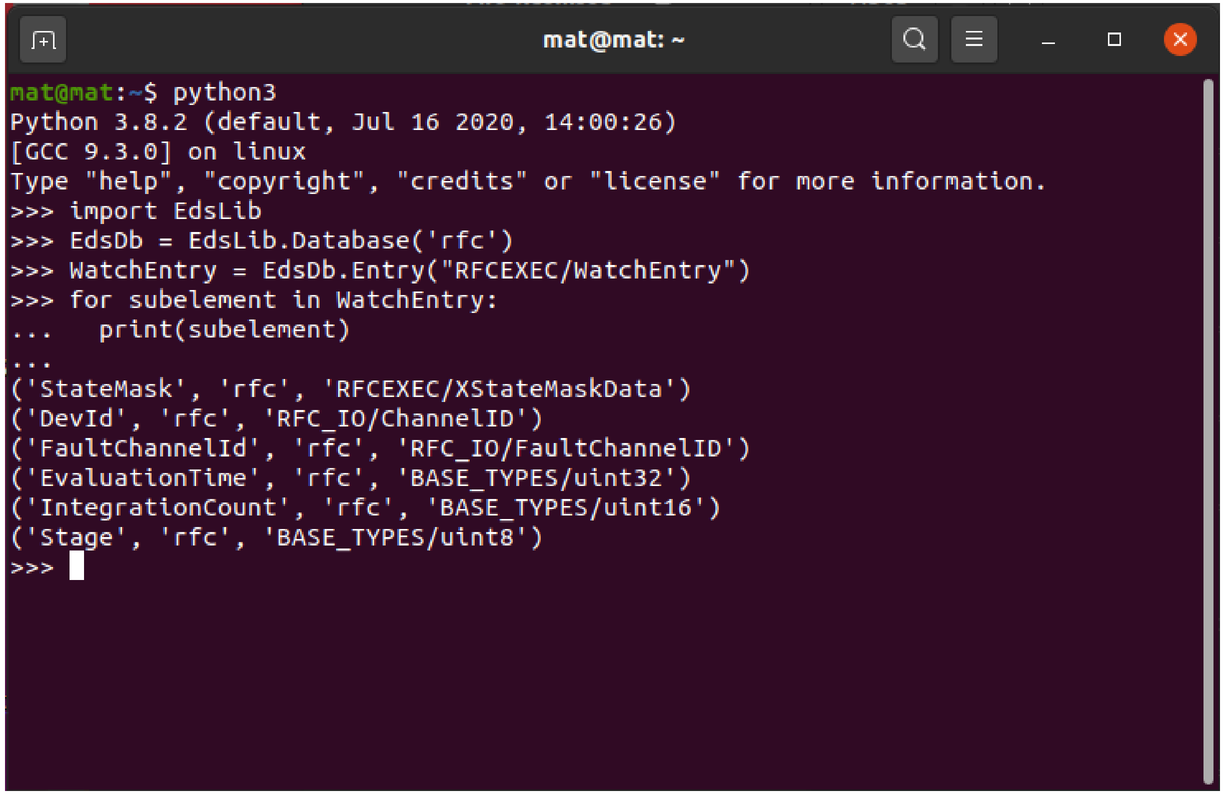
- EdsDb: EDS Database referenced by mission name
- DbEntry: Function to create an EDS object in python
  - Referenced by the same naming convention as EDS
- DbObject: EDS Object that can be used in Python
  - Structs are treated as python dictionaries

New EdsLib Python Bindings

- **Iterators:**
  - EdsDb entries for Enumerations will loop over all the label/value pairs
    - Create a Python dictionary with the label/value pairs
  - EdsDb entries for structures will loop over all the sub-elements
    - Gives information to create each sub-element in python



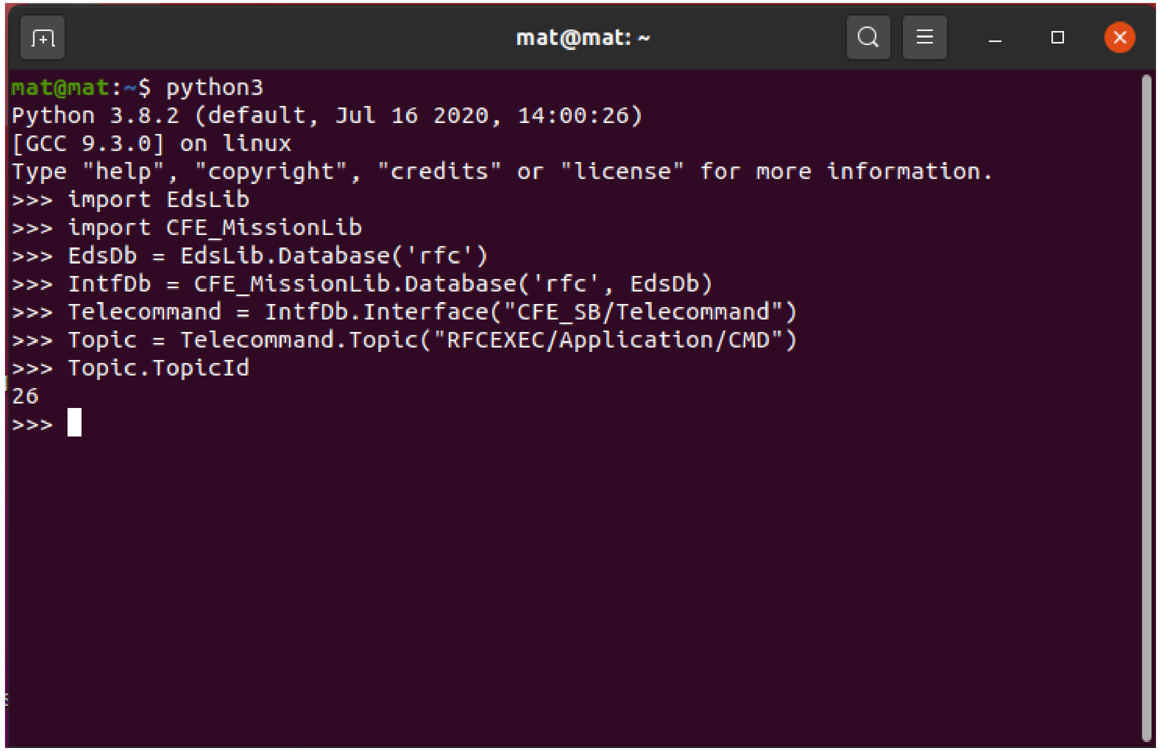
```
mat@mat:~$ python3
Python 3.8.2 (default, Jul 16 2020, 14:00:26)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import EdsLib
>>> EdsDb = EdsLib.Database('rfc')
>>> Enumeration = EdsDb.Entry("RFCEXEC/XState")
>>> for item in Enumeration:
...   print(item)
...
('ESTOP', 15)
('NORMAL_EZ', 8)
('NORMAL_EZtoNORMAL_FC', 10)
('NORMAL_EZtoSTANDBY', 12)
('NORMAL_FC', 6)
('NORMAL_FCtoNORMAL_EZ', 9)
('NORMAL_FCtoSTANDBY', 11)
('SAFE', 2)
('SAFEtoSTANDBY', 3)
('SAFEtoUNPOWERED', 14)
('STANDBY', 4)
('STANDBYtoNORMAL_EZ', 7)
('STANDBYtoNORMAL_FC', 5)
('STANDBYtoSAFE', 13)
('UNPOWERED', 0)
('UNPOWEREDtoSAFE', 1)
```



```
mat@mat:~$ python3
Python 3.8.2 (default, Jul 16 2020, 14:00:26)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import EdsLib
>>> EdsDb = EdsLib.Database('rfc')
>>> WatchEntry = EdsDb.Entry("RFCEXEC/WatchEntry")
>>> for subelement in WatchEntry:
...   print(subelement)
...
('StateMask', 'rfc', 'RFCEXEC/XStateMaskData')
('DevId', 'rfc', 'RFC_IO/ChannelID')
('FaultChannelId', 'rfc', 'RFC_IO/FaultChannelID')
('EvaluationTime', 'rfc', 'BASE_TYPES/uint32')
('IntegrationCount', 'rfc', 'BASE_TYPES/uint16')
('Stage', 'rfc', 'BASE_TYPES/uint8')
>>> 
```

CFE_MissionLib Python Bindings

- **EDS/cFS interface objects that can be created within python**
  - Interface database object which contains a pointer to the database itself
  - Interface object: CFE_SB/Telemetry and CFE_SB/Telecommand
  - Topic object: RFCEXEC/Application/CMD
    - The Topic ID is an accessible member of the Topic python object

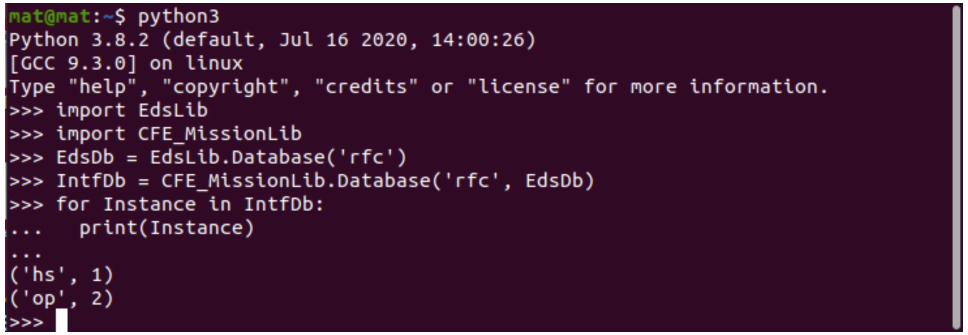


```
mat@mat:~$ python3
Python 3.8.2 (default, Jul 16 2020, 14:00:26)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import EdsLib
>>> import CFE_MissionLib
>>> EdsDb = EdsLib.Database('rfc')
>>> IntfDb = CFE_MissionLib.Database('rfc', EdsDb)
>>> Telecommand = IntfDb.Interface("CFE_SB/Telecommand")
>>> Topic = Telecommand.Topic("RFCEXEC/Application/CMD")
>>> Topic.TopicId
26
>>> 
```

Note: In order to use CFE_MissionLib we must use EdsLib

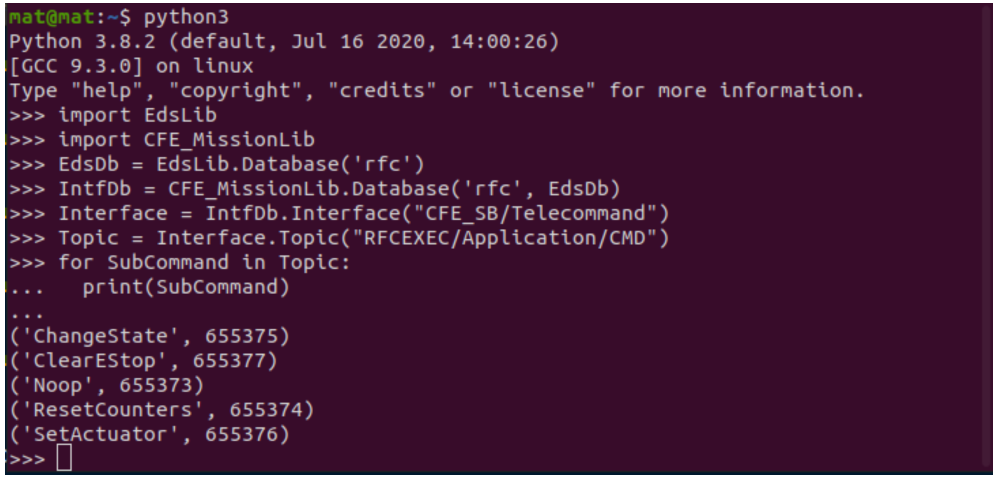
CFE_MissionLib Python Bindings

- **Iterators:**
  - Interface Database Object: iterates over the cFS instance names
  - Interface: iterates over the topics for that interface
  - Topic: iterators over the subcommands (if available)
    - Gives the numeric identifier of the EDS command object (EdsId)



```
mat@mat:~$ python3
Python 3.8.2 (default, Jul 16 2020, 14:00:26)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import EdsLib
>>> import CFE_MissionLib
>>> EdsDb = EdsLib.Database('rfc')
>>> IntfDb = CFE_MissionLib.Database('rfc', EdsDb)
>>> for Instance in IntfDb:
...     print(Instance)
...
('hs', 1)
('op', 2)
>>> 
```

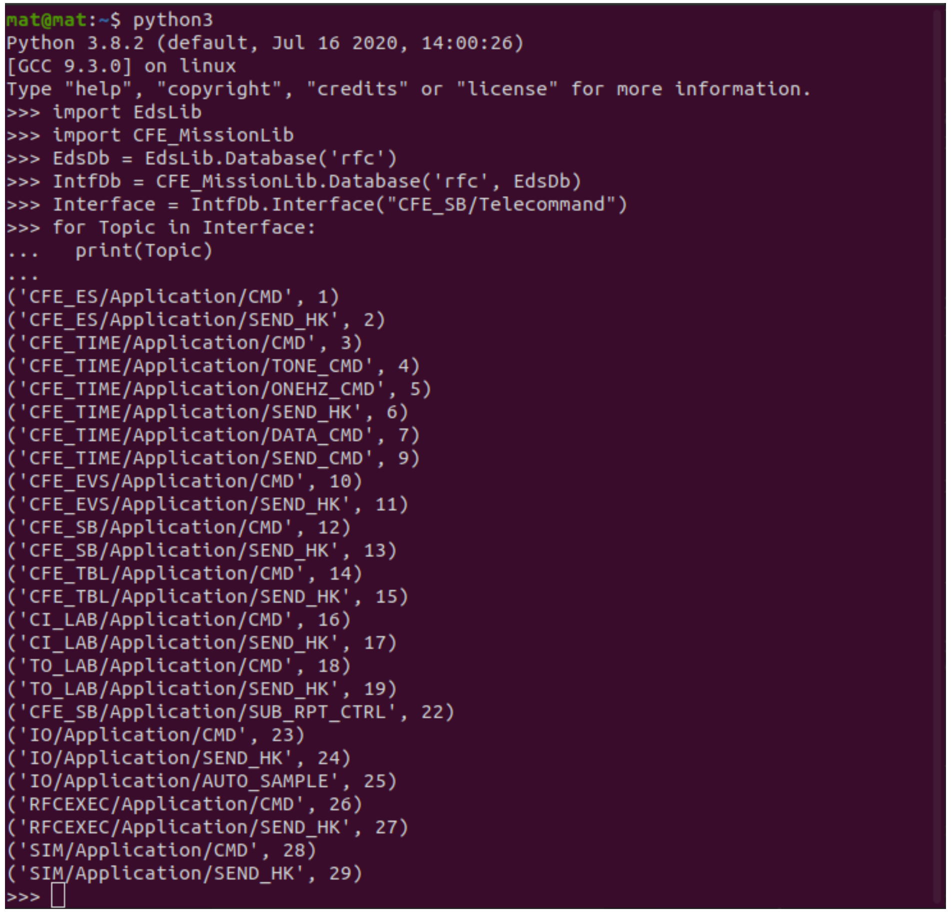
Database Iterator



```
mat@mat:~$ python3
Python 3.8.2 (default, Jul 16 2020, 14:00:26)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import EdsLib
>>> import CFE_MissionLib
>>> EdsDb = EdsLib.Database('rfc')
>>> IntfDb = CFE_MissionLib.Database('rfc', EdsDb)
>>> Interface = IntfDb.Interface("CFE_SB/Telecommand")
>>> Topic = Interface.Topic("RFCEXEC/Application/CMD")
>>> for SubCommand in Topic:
...     print(SubCommand)
...
('ChangeState', 655375)
('ClearEStop', 655377)
('Noop', 655373)
('ResetCounters', 655374)
('SetActuator', 655376)
>>> 
```

Topic Iterator

Interface Iterator



```
mat@mat:~$ python3
Python 3.8.2 (default, Jul 16 2020, 14:00:26)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import EdsLib
>>> import CFE_MissionLib
>>> EdsDb = EdsLib.Database('rfc')
>>> IntfDb = CFE_MissionLib.Database('rfc', EdsDb)
>>> Interface = IntfDb.Interface("CFE_SB/Telecommand")
>>> for Topic in Interface:
...     print(Topic)
...
('CFE_ES/Application/CMD', 1)
('CFE_ES/Application/SEND_HK', 2)
('CFE_TIME/Application/CMD', 3)
('CFE_TIME/Application/TONE_CMD', 4)
('CFE_TIME/Application/ONEHZ_CMD', 5)
('CFE_TIME/Application/SEND_HK', 6)
('CFE_TIME/Application/DATA_CMD', 7)
('CFE_TIME/Application/SEND_CMD', 9)
('CFE_EVS/Application/CMD', 10)
('CFE_EVS/Application/SEND_HK', 11)
('CFE_SB/Application/CMD', 12)
('CFE_SB/Application/SEND_HK', 13)
('CFE_TBL/Application/CMD', 14)
('CFE_TBL/Application/SEND_HK', 15)
('CI_LAB/Application/CMD', 16)
('CI_LAB/Application/SEND_HK', 17)
('TO_LAB/Application/CMD', 18)
('TO_LAB/Application/SEND_HK', 19)
('CFE_SB/Application/SUB_RPT_CTRL', 22)
('IO/Application/CMD', 23)
('IO/Application/SEND_HK', 24)
('IO/Application/AUTO_SAMPLE', 25)
('RFCEXEC/Application/CMD', 26)
('RFCEXEC/Application/SEND_HK', 27)
('SIM/Application/CMD', 28)
('SIM/Application/SEND_HK', 29)
>>> 
```

CFE_MissionLib Python Bindings Methods

- **Decode a generic telemetry message**
  - Each telemetry packet is based off of a CCSDS_SpacePacket_t header structure
  - Partially decode just the header portion of the incoming message
  - From the header information, the Topic ID can be extracted
  - Call the CFE_MissionLib API functions to return the EdsId of the associated Topic ID
    - This EDS Object is the full telemetry packet structure of the incoming message
  - With the EDS object known the full message is decoded into a python object
- **Set Publish/Subscribe Parameters for a command message**
  - Input the Instance ID, Topic ID, and the Python object associated with the command packet
  - Calls the CFE_MissionLib API functions to take the input information and fill in the appropriate header values in the packet
    - Default: SecHeaderFlags, Apid, SubsystemID are filled in
    - This is customizable based on the types of message headers used in the mission
  - Once these parameters are set the command message can be packed and sent to its destination.