

The RFC Project – An Example of core Flight Systems (cFS) with Electronic Data Sheets (EDS) Support

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



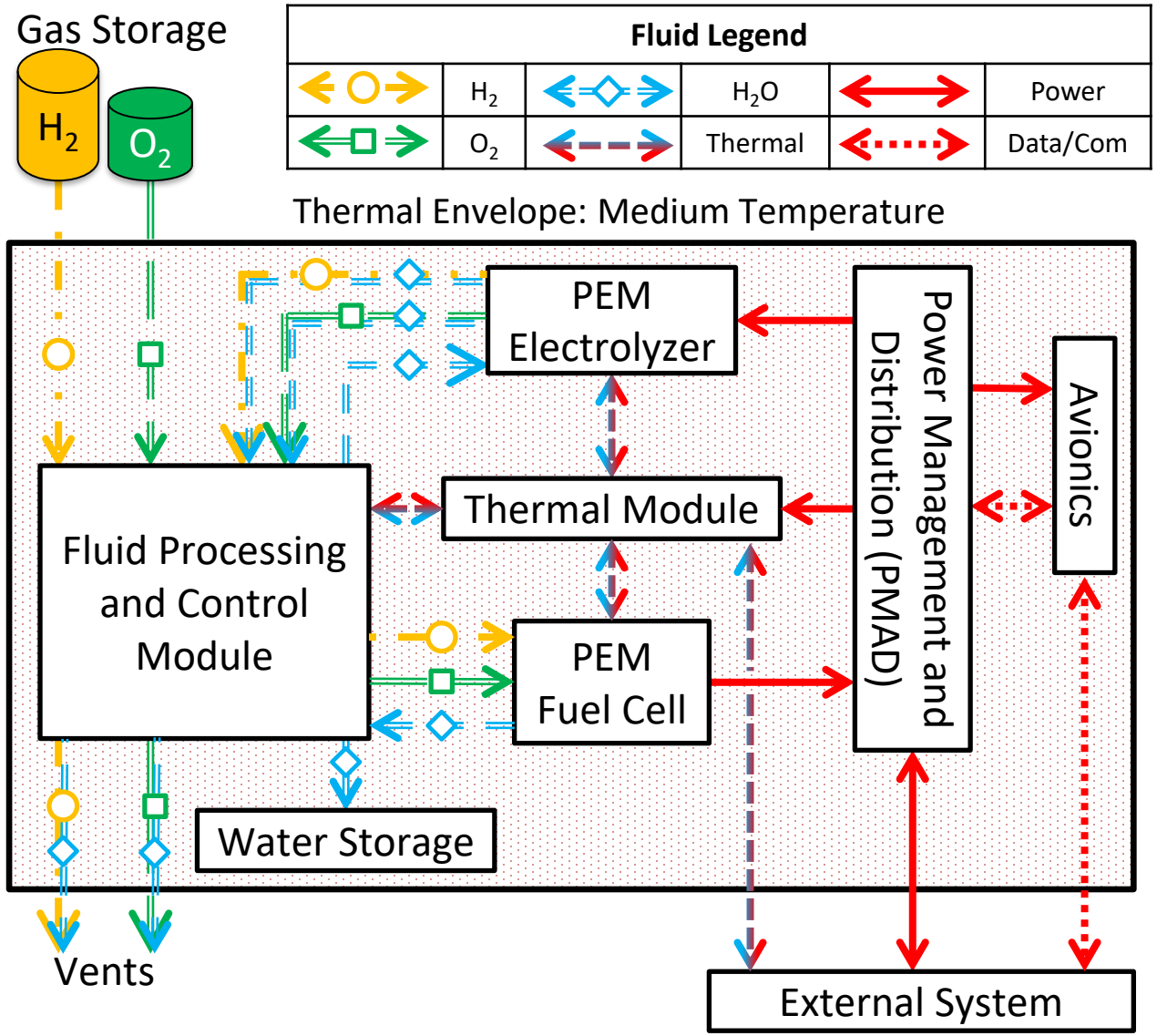
Outline



- **The Regenerative Fuel Cell (RFC) Project overview**
  - RFC Software Requirements
  - Selecting cFS with Electronic Data Sheets (EDS) support
- **cFS with EDS support overview**
  - EdsLib tool/database
  - Interfacing EdsLib with cFS (CFE_MissionLib database)
- **RFC software approach**
  - Operational Program and Hardware Simulator
    - Uses cFS with EDS support
  - Operator Interface
    - EdsLib/CFE_MissionLib Python bindings
- **Configuration and Verification**
- **Initial RFC Results**

The Regenerative Fuel Cell (RFC) Project Overview

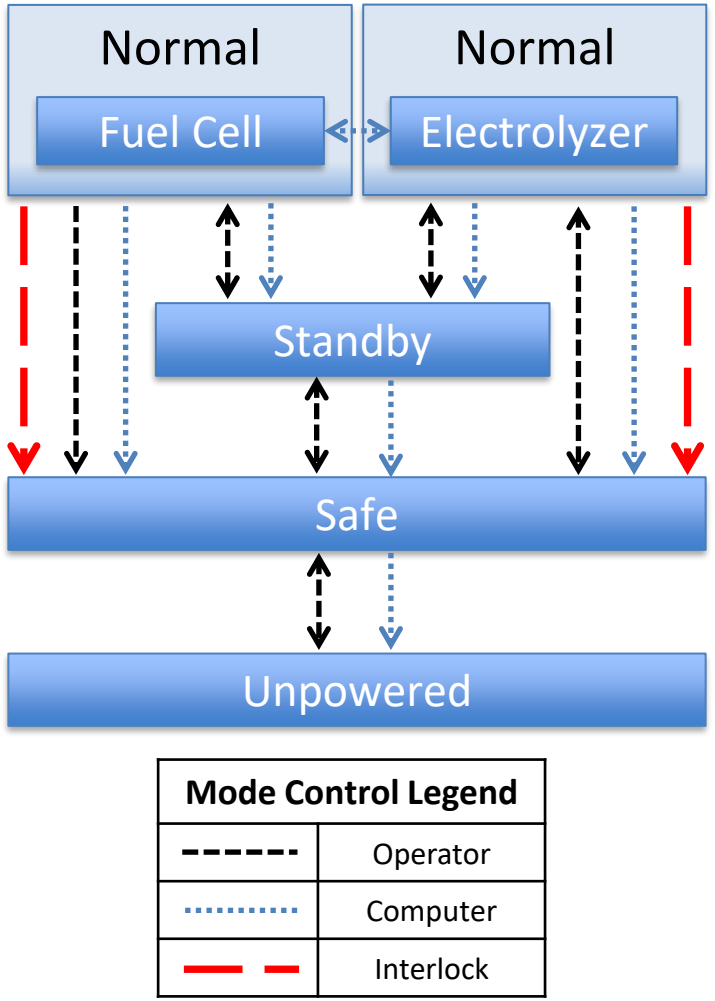
- **A Regenerative Fuel Cell is an energy storage system that utilizes hydrogen and oxygen gases to store energy**
  - Fuel Cell converts gas reactants to electricity and product water
  - Electrolyzer uses electricity (e.g., from a photovoltaic array) to convert water back to gaseous reactants
- **Primary 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 Requirements

- **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
  - Store operational data in internal data files and send telemetry to a remote operator interface
- **Monitors sensors to check if they are within operational limits**
  - Caution, Warning, Alarm system is set up to quantify the severity of a fault
  - Critical sensors (e.g., cell voltages) need response times of 100 ms, which would be too fast for the cFS Health and Safety App.
  - 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)

Operational Modes



Selecting cFS with EDS Support

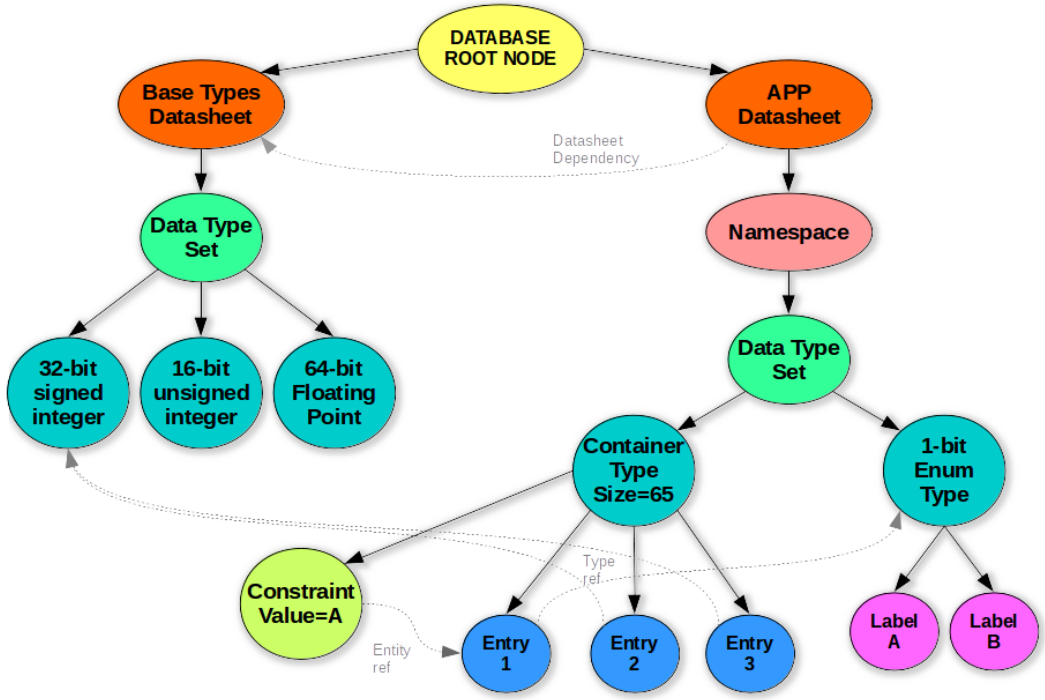
- **Initially, the goal of the RFC project was to develop an energy storage system to be tested in a TVAC environment (TRL 6) with the potential to transition to a flight project.**
  - Software development started early in the project (2019)
  - cFS has a history of being successfully used on flight projects.
- **Prior to RFC, several GRC projects with similar requirements were used as starting points for RFC software**
  - AMPS power module
    - Monitoring/Control software for a fuel cell stack
  - EVA Caution Warning System
    - Similar fault monitoring system and state machine
    - Used cFS with EDS support
- **Benefits to using cFS with EDS support:**
  - EDS files become the single source of truth for object definitions alleviating conflicts in structure definitions
  - EDS Libraries simplify the data communication between different computer architectures
    - Cubesat Space Processor (CSP) to/from ground computer
    - Common definitions made it easier to simulate hardware responses for testing and validation.
  - Community of developers that could provide assistance.



AMPS 1kW Fuel Cell stack powering SCARAB rover

cFS With EDS Support Overview

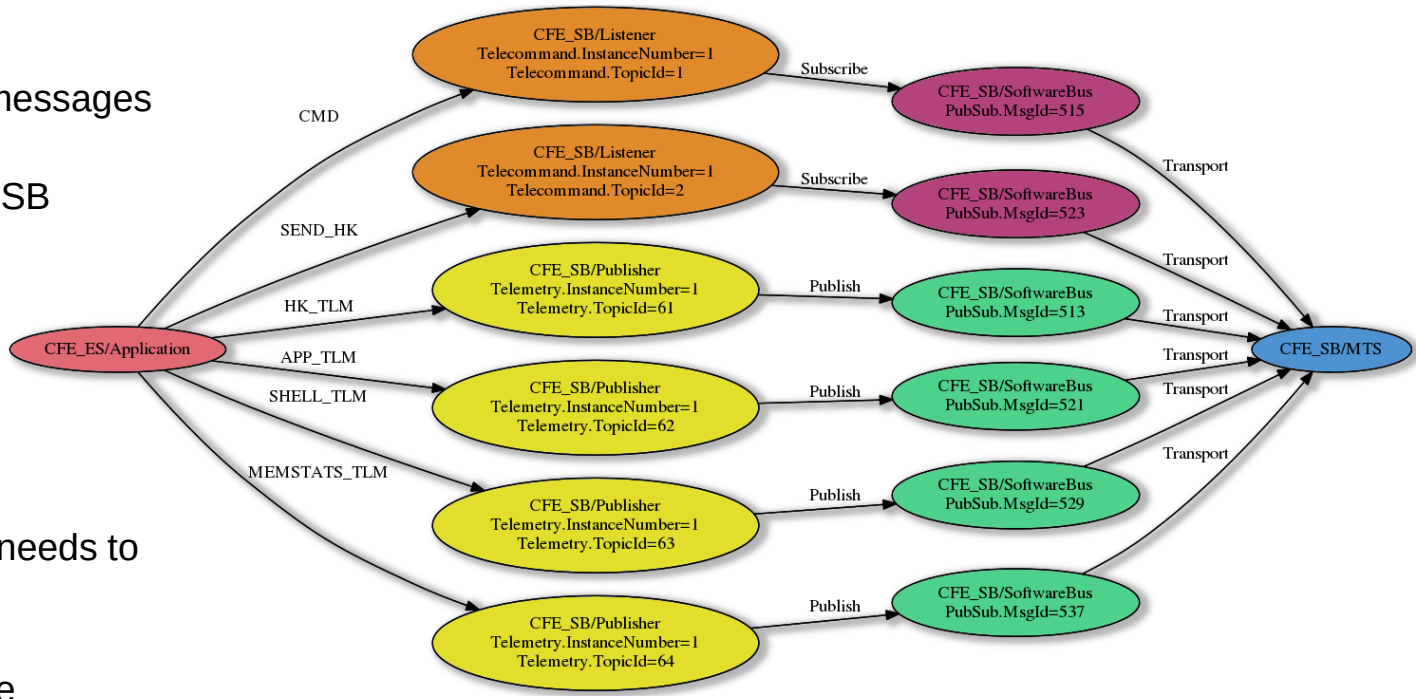
- **Spacecraft Onboard Interface Services (SOIS) EDS XML specifications are defined by CCSDS 867.0-B-1 (Blue Book): https://public.ccsds.org/Pubs/876x0b1.pdf**
  - Specifications for defining constants, data structures, interfaces, constraints, etc. across multiple packages/files
- **EdsLib: https://github.com/nasa/edslib**
  - 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

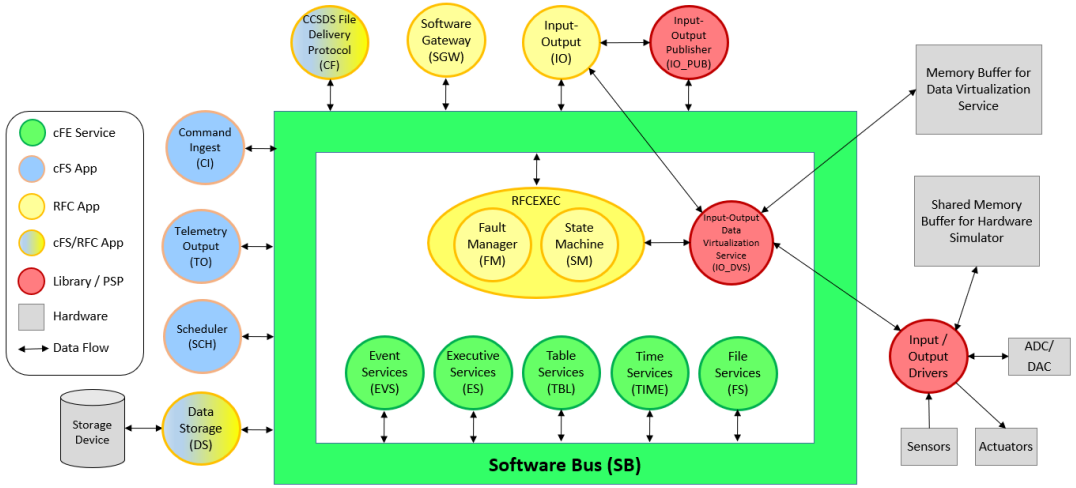
- **Additional tools and libraries were developed to interface EdsLib with cFS**
  - The interface specifications of EDS are used to define the telemetry and telecommand interfaces of the cFS Software Bus (SB)
    - Topic IDs for telemetry/telecommand are managed in an EDS file
    - Constraints are 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
  - Previous talk at the 2022 Flight Software Workshop showing usage and benefits of cFS with EDS Support
    - https://www.youtube.com/watch?v=36wuoQbOc9w



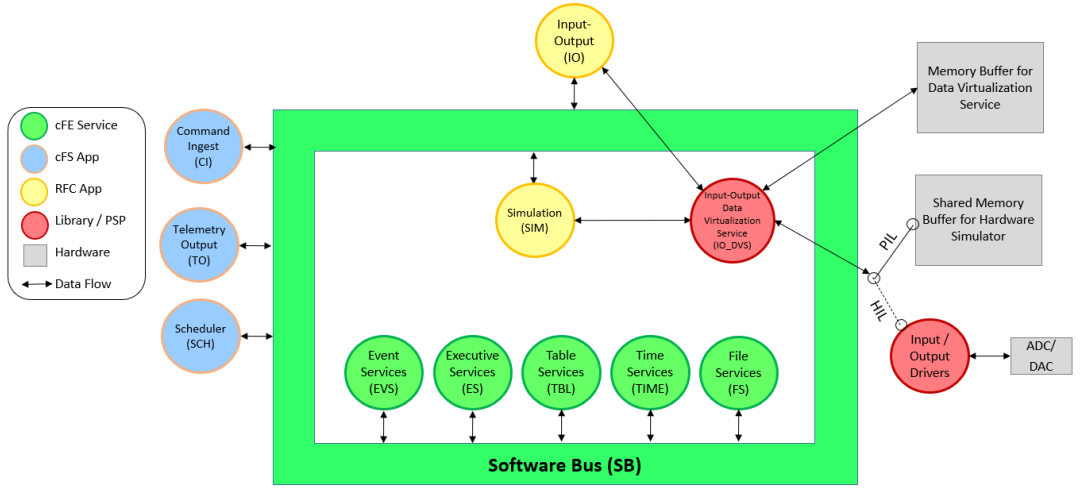
Example Interface Diagram

RFC Software Approach

- **Operational Program (OP)**
  - Resides on a CubeSat Space Processor (CSP)
  - Manages state machine, fault monitoring, fault actions, telemetry, and data storage
  - Converted DS and CF Apps from the greater cFS community to be compatible with EDS
- **Hardware Simulator (HS)**
  - Tool developed to provide a low fidelity simulation of 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.
  - Gives an operator the ability to send commands to the OP.
  - Download data files stored on the CSP.
  - Additional python script to convert packed EDS data log files in CVS format for further processing

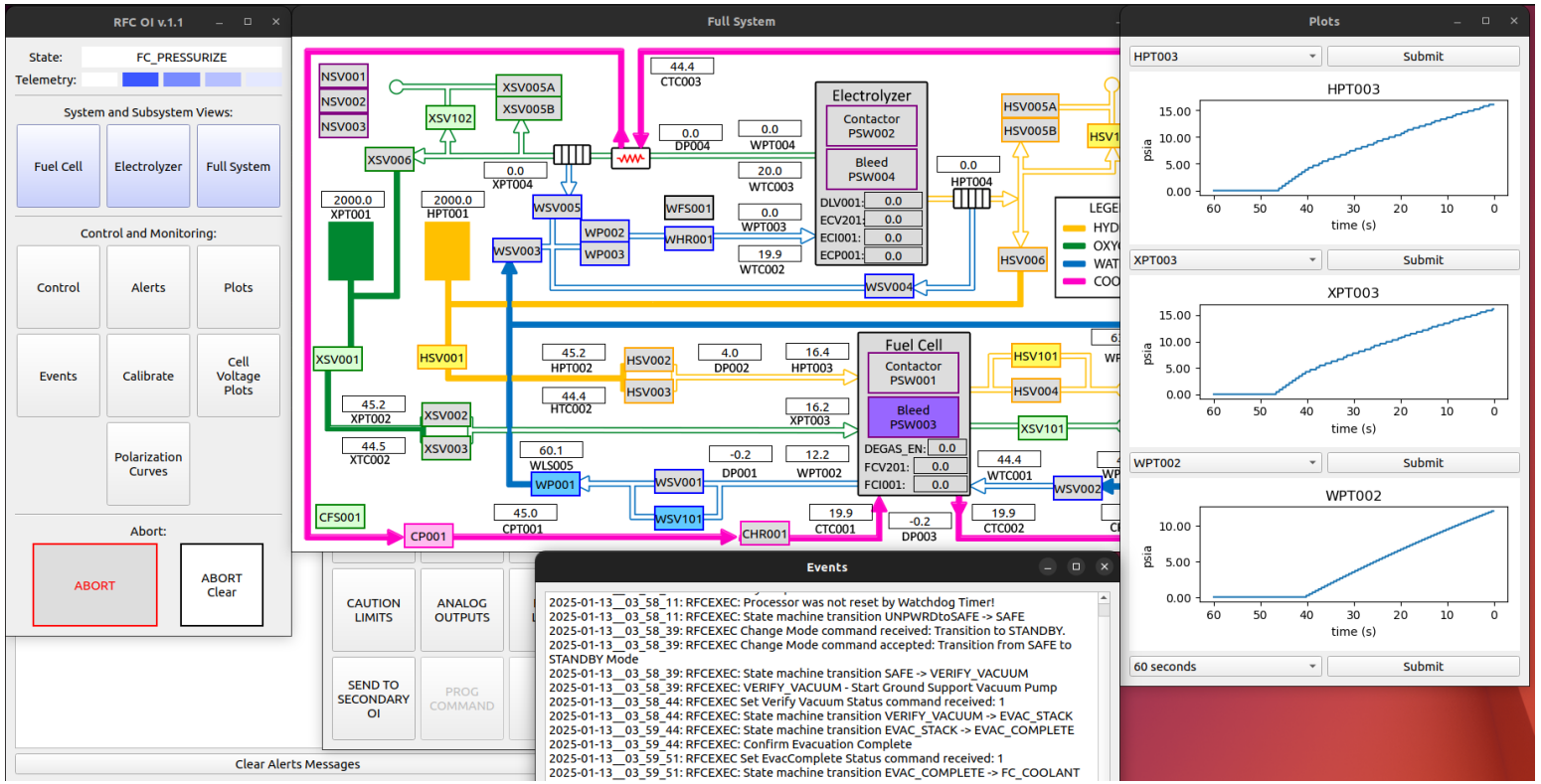


OP



HS

Operator Interface



- Telemetry and Command objects are defined in EDS files and can be created as objects in python
  - EdsLib API to pack/unpack SB messages
- The RFC project developed additional Python bindings:
  - Iterators for containers, enumerations, cFS instances, topics, subcommands to extract labels and values
  - Methods to extract instance and topic information from telemetry SB messages
  - Methods to set Publish/Subscribe parameters for telecommand SB messages

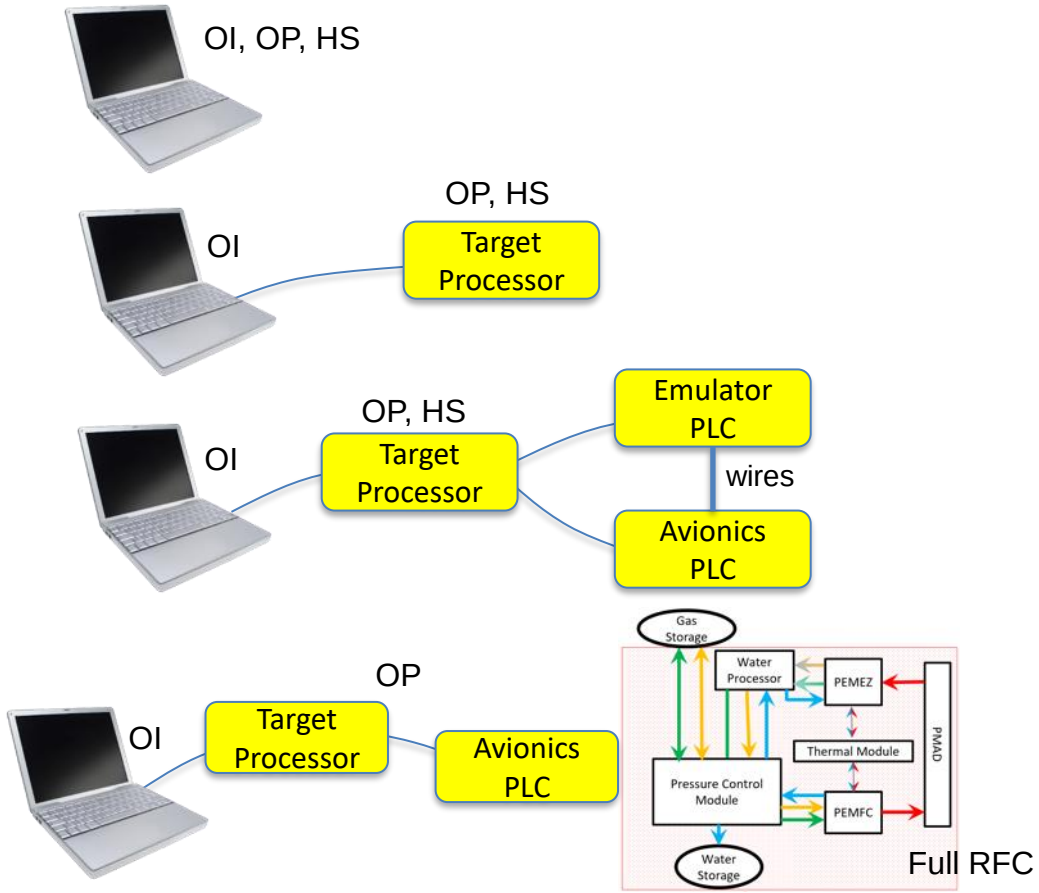
RFC Project Software Development Configurations

All-in-one: OI communicates locally with the OP and HS running on the same computer

Processor-in-the-Loop: Remote Computer (OI) communicates with the target process running both the OP and HS

Hardware-in-the-Loop: OI communicates with the OP running on the target processor. OP drives Avionics PLC while HS drives the Emulator PLC to send physical signals.

Full Hardware System: OI communicates with the OP running on the target processor with inputs/output from the full test bed hardware



- Configurations can be switched by selecting a LUA script with the appropriate hardware definitions
- Allowed software development to make progress while hardware development was slowed during CoVID

RFC Software Verification

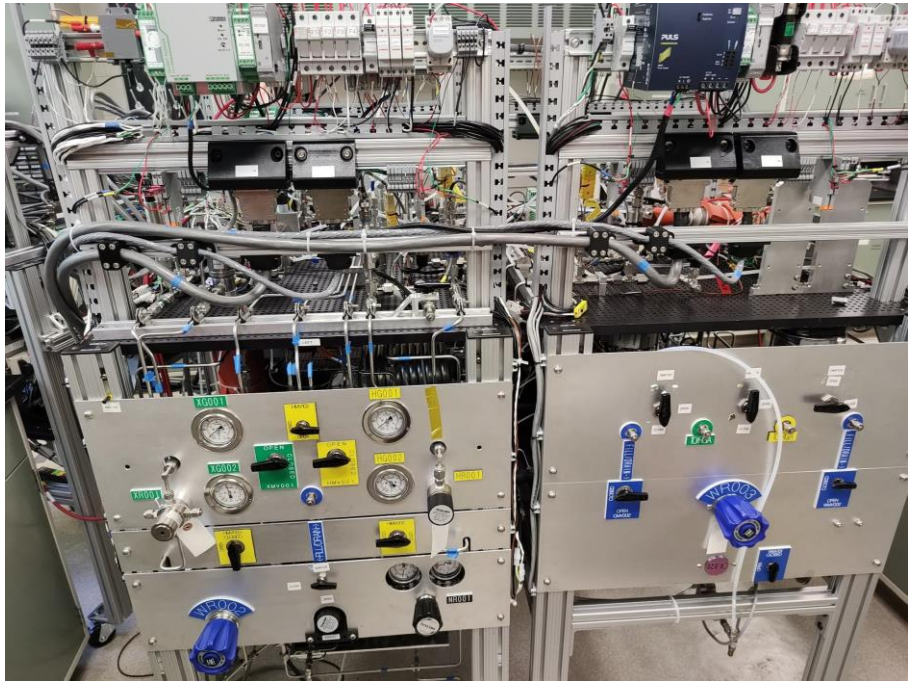
- **Functional Testing Framework**
  - Simple python script that listens for telemetry messages sent by the OP and executes a sequence of commands
  - Allows automation of functional tests for both development and formal verification.
- **Example: Fault Test**
  - Over 100 channels, 4 fault modes, 6 fault types per mode
  - Read in the fault table configuration file
  - Generate a command list for all actionable faults by looping over all channels, fault modes, and fault types:
    - Set up state machine to appropriate state
    - Send fault simulation to the HS
    - Wait for the appropriate event message to arrive for that fault
  - If channels/faults are changed, the functional test itself does not need to be updated.

Electrical Block Diagram: RFC-WRK-DOC-0846, RFC-WRK-DRW-0107				Safe						Safe					
Hardware Tag	Data Scan Rate, Hz	Data Save Rate, Hz	Persistence Time, s	Alarm - Low	Warning - Low	Caution - Low	Caution - High	Warning - High	Alarm - High	Alarm - Low	Warning - Low	Caution - Low	Caution - High	Warning - High	Alarm - High
HTC002	1	0.016	120	-113	0	4	85	88	90	7	3	1	1	3	7
XTC002	1	0.016	120	-113	0	4	85	88	90	7	3	1	1	3	7
WTC001	1	0.016	120	1	2	4	85	88	90	7	3	1	1	3	7
WTC002	1	1	10	1	2	4	85	88	90	7	3	1	1	3	7
WTC003	1	1	10	1	2	4	85	88	90	7	3	1	1	3	7
DT001	1	0.016	120	-10	-8	-6	6	8	10	7	3	1	1	3	7
WTC004	1	0.016	120	1	2	4	85	88	90	7	3	1	1	3	7
WTC005	1	0.016	120	1	2	4	85	88	90	7	3	1	1	3	7
WTC006	1	0.016	120	1	2	4	160	180	200	0	0	0	1	3	7
CTC001	1	1	1	4	7	10	73	77	80	7	3	1	1	3	7
CTC002	1	1	1	4	7	10	72	75	80	7	3	1	1	3	7
DT002	1	0.016	10	N/A	N/A	N/A	N/A	N/A	N/A	0	0	0	0	0	0
CTC003	1	0.016	120	1	2	4	85	88	90	7	3	1	1	3	7
CTC004	1	0.016	120	1	2	4	85	88	90	7	3	1	1	3	7
CTC005	1	0.016	120	1	2	4	90	95	100	7	3	1	1	1	7
CTS005	1	0.016	N/A	0	N/A	N/A	N/A	N/A	N/A	0	0	0	0	0	0
ATC001	1	0.016	120	0	4	7	85	95	100	7	1	1	1	1	7
PTC001	1	0.016	120	N/A	N/A	N/A	N/A	N/A	N/A	0	0	0	0	0	0
PTC002	1	0.016	120	N/A	N/A	N/A	N/A	N/A	N/A	0	0	0	0	0	0
PTC003	1	0.016	120	N/A	N/A	N/A	N/A	N/A	N/A	0	0	0	0	0	0

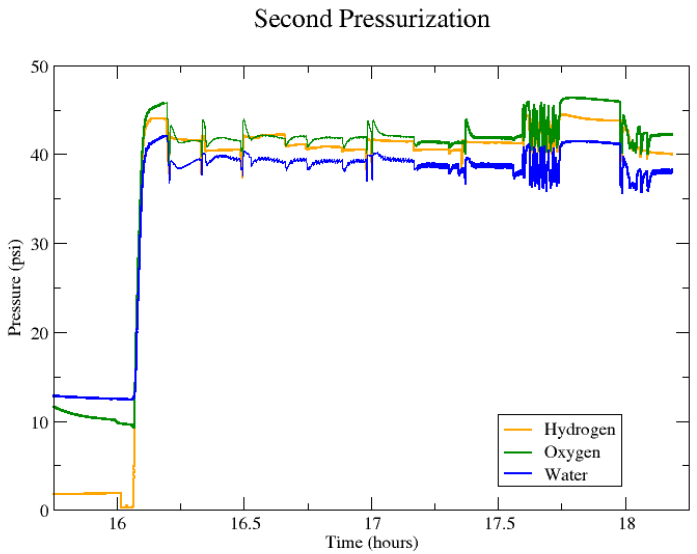
Fault Table Snippet

With the functional tests covering fault monitoring and command checking, formal testing was completed in 2 days

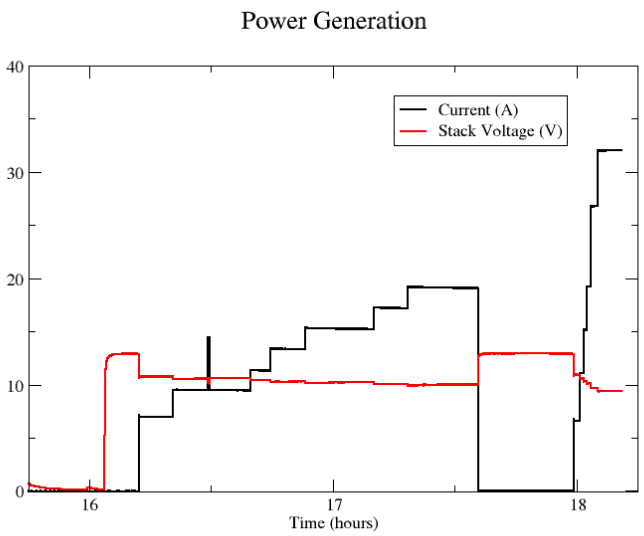
Initial RFC Results



Fuel Cell and Electrolyzer Carts



Pressurization of the stack showing several control algorithms in action



Power generation of the fuel cell stack

- **With software verification complete the RFC system was able to be run with reactant gasses**
  - Generated over 300 W of power
- **Next steps include testing the Electrolyzer stack and transitions between Fuel Cell and Electrolyzer operation**
  - Goal is for at least one unattended day/night cycle operation.





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.

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

rfcexec.xml

```
58       <ContainerDataType name="XStateMaskData">
59         <EntryList>
60           <Entry name="Enable"  type="XStateMask" />
61           <Entry name="Disable" type="XStateMask" />
62         </EntryList>
63       </ContainerDataType>
64
65       <ContainerDataType name="WatchEntry">
66         <EntryList>
67           <Entry name="StateMask"        type="XStateMaskData" />
68           <Entry name="DevId"            type="RFC_IO/ChannelID" />
69           <Entry name="FaultChannelId"   type="RFC_IO/FaultChannelID" />
70           <Entry name="EvaluationTime"   type="BASE_TYPES/uint32" />
71           <Entry name="IntegrationCount" type="BASE_TYPES/uint16" />
72           <Entry name="Stage"            type="BASE_TYPES/uint8" />
73         </EntryList>
74       </ContainerDataType>
```

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

```
257 /**
258  * @brief Structure definition for CONTAINER_DATATYPE 'RFCEXEC/WatchEntry'
259  *
260  * Data definition signature 7349168c76e94374
261  */
262 struct rfcexec_7349168c76e94374 /* RFCEXEC_WatchEntry */
263 {
264    RFCEXEC_XStateMaskData_t                 StateMask;              /* 32  bits/4   bytes */
265    RFC_IO_ChannelID_Enum_t                  DevId;                  /* 8   bits/1   bytes */
266    RFC_IO_FaultChannelID_Enum_t             FaultChannelId;         /* 7   bits/1   bytes */
267    BASE_TYPES_uint32_Atom_t                 EvaluationTime;         /* 32  bits/4   bytes */
268    BASE_TYPES_uint16_Atom_t                 IntegrationCount;       /* 16  bits/2   bytes */
269    BASE_TYPES_uint8_Atom_t                  Stage;                  /* 8   bits/1   bytes */
270 };
271
272 /**
273  *
274  */
275 typedef struct rfcexec_7349168c76e94374                 RFCEXEC_WatchEntry_t;
276   /* bits= 103  bytes=  15/16    align=0x3  checksum=7349168c76e94374 */
277
278 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,

   /**
    * @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.**

```
412   { /* RFCEXEC/WatchEntry */
413      .Checksum = 0x7349168c76e94374,
414      .BasicType = EDSLIB_BASICTYPE_CONTAINER,
415      .NumSubElements = 6,
416      .SizeInfo = { .Bits = 103, .Bytes = sizeof(RFCEXEC_WatchEntry_t) },
417      .Detail.Container = &RFCEXEC_WatchEntry_CONTAINER_DETAIL
418   },
```

```
392   { /* RFCEXEC/XState */
393      .Checksum = 0xdd8c19415775662e,
394      .BasicType = EDSLIB_BASICTYPE_UNSIGNED_INT,
395      .SizeInfo = { .Bits = 4, .Bytes = sizeof(RFCEXEC_XState_Enum_t) }
396   },
```

rfcexec_eds_datatypedb_impl.c

```
41 static const EdsLib_SymbolTableEntry_t RFCEXEC_XState_Enum_SYMTABLE[] =
42 {
43    { .SymValue = 15, .SymName = "ESTOP" },
44    { .SymValue = 8, .SymName = "NORMAL_EZ" },
45    { .SymValue = 10, .SymName = "NORMAL_EZtoNORMAL_FC" },
46    { .SymValue = 12, .SymName = "NORMAL_EZtoSTANDBY" },
47    { .SymValue = 6, .SymName = "NORMAL_FC" },
48    { .SymValue = 9, .SymName = "NORMAL_FCtoNORMAL_EZ" },
49    { .SymValue = 11, .SymName = "NORMAL_FCtoSTANDBY" },
50    { .SymValue = 2, .SymName = "SAFE" },
51    { .SymValue = 3, .SymName = "SAFEtoSTANDBY" },
52    { .SymValue = 14, .SymName = "SAFEtoUNPOWERED" },
53    { .SymValue = 4, .SymName = "STANDBY" },
54    { .SymValue = 7, .SymName = "STANDBYtoNORMAL_EZ" },
55    { .SymValue = 5, .SymName = "STANDBYtoNORMAL_FC" },
56    { .SymValue = 13, .SymName = "STANDBYtoSAFE" },
57    { .SymValue = 0, .SymName = "UNPOWERED" },
58    { .SymValue = 1, .SymName = "UNPOWEREDtoSAFE" }
59 };
```

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.

```
<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

```
1357    [25] =
1358    {
1359        .DispatchTableId = RFCEXEC_Application_Component_Telecommand_DISPATCHTABLE_ID,
1360        .DispatchStartOffset = offsetof(RFCEXEC_Application_Component_Telecommand_DispatchTable_t,CMD),
1361        .InterfaceId = CFE_SB_Telecommand_Interface_ID,
1362        .TopicName = "RFCEXEC/Application/CMD",
1363        .CommandList = RFCEXEC_Application_CMD_Interface_COMMANDS
1364    },
```

Topic Listing

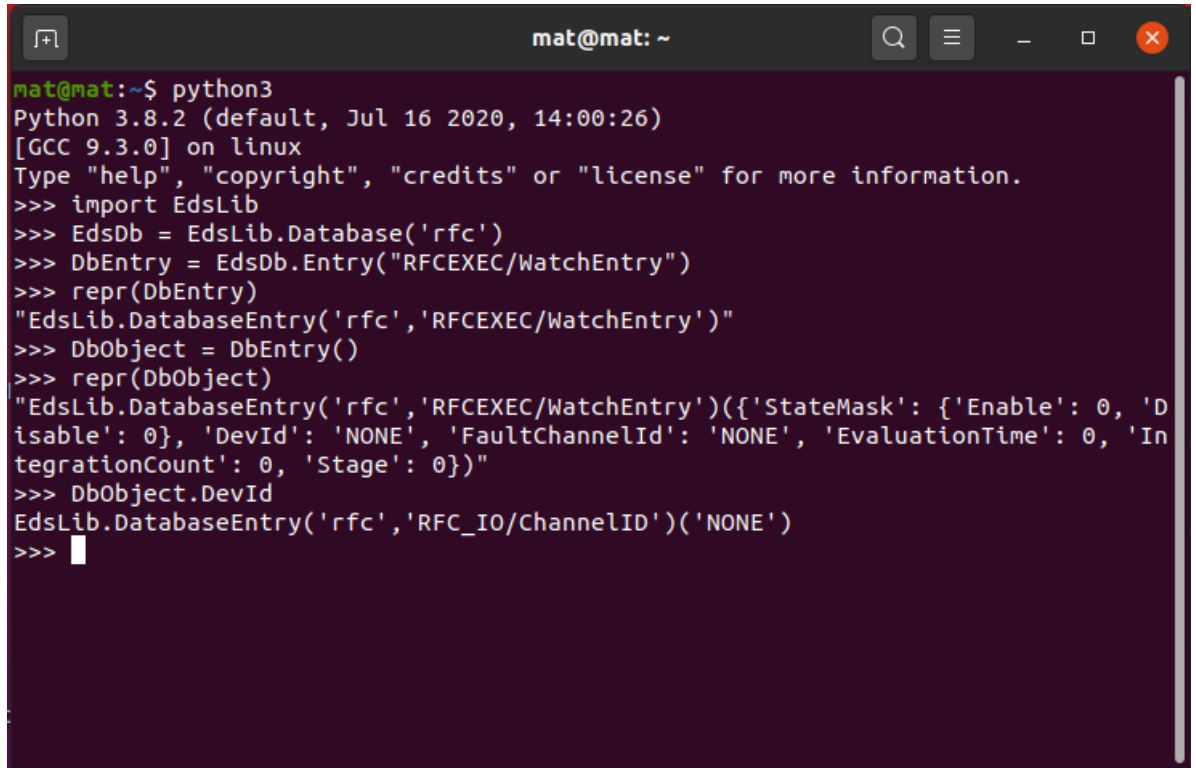
```
733 static const CFE_MissionLib_Command_Definition_Entry_t RFCEXEC_Application_CMD_Interface_COMMANDS[] =
734 {
735     {
736         .SubcommandArg = 1,
737         .SubcommandCount = 5,
738         .SubcommandList = RFCEXEC_Application_CMD_Interface_indication_SUBCOMMAND_LIST,
739         .ArgumentList = RFCEXEC_Application_CMD_Interface_indication_ARGUMENT_LIST
740     }
741 };
```

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



- 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

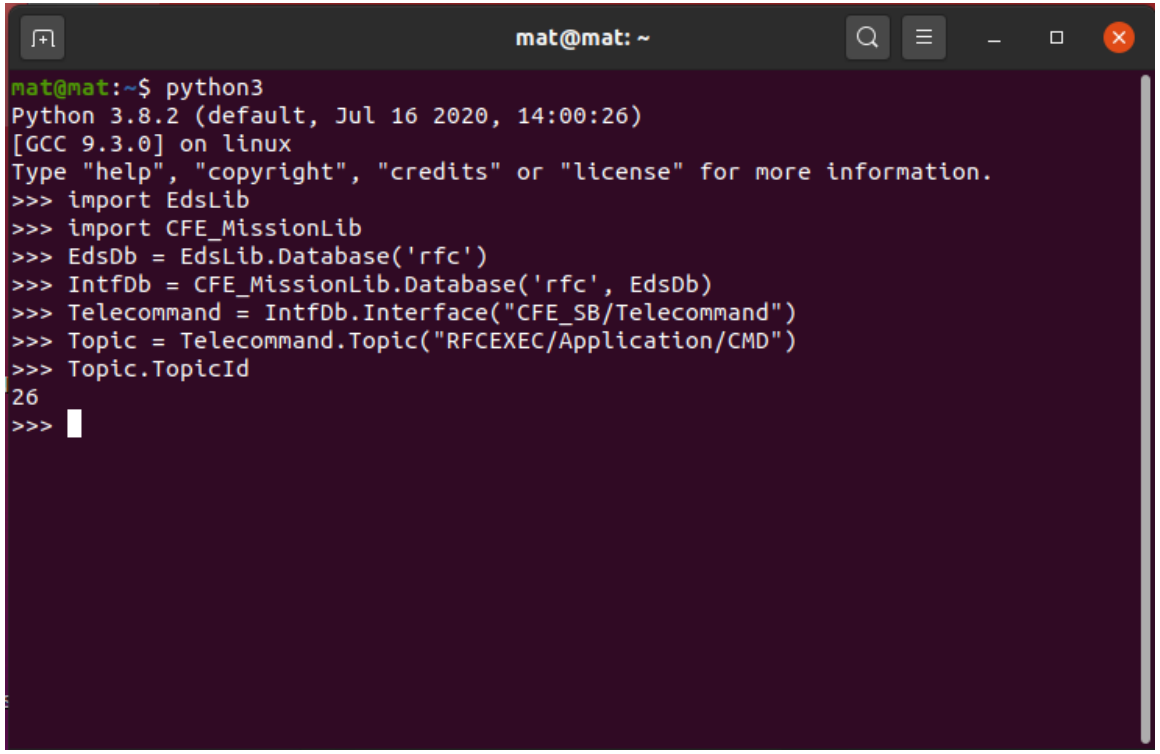
- **Iterators:**
  - EdsDb entries for Enumerations will loop over all the label/value pairs
    - Create a Python dictionary with the label/value pairs
  - EdsDb entries for structures will loop over all the sub-elements
    - Gives information to create each sub-element in python

```
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: ~
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



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 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.

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

```
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)
```

...

```
721
722 Master = EdsDB.NewObject("RFCEXEC/MasterFaultTable")
723 Master.Table = Table
724
725 -- Write the output file
726 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