|  |  |
| --- | --- |
|  |  |
| National Aeronautics and  Space Administration |  |

John H. Glenn Research Center

21000 Brookpark Road

Cleveland, OH 44135

cFS-EDS-GroundStation Users Manual

Change History

|  |  |  |
| --- | --- | --- |
| **Revision** | **Effective Date** | **Description** |
| *1.0* | *10/6/2020* | *First Release* |
| *1.1* | *9/10/2021* | *Update installation instructions* |
|  |  |  |
|  |  |  |
|  |  |  |

Table of Contents

[1.0 Prerequisites: 4](#_Toc52971115)

[2.0 Installation: 4](#_Toc52971116)

[3.0 Execution: 5](#_Toc52971117)

[3.1 Telecommand System: 5](#_Toc52971118)

[3.2 Telemetry System 7](#_Toc52971119)

[4.0 Utility Scripts 8](#_Toc52971120)

[4.1 cmd\_util 8](#_Toc52971121)

[4.2 tlm\_decode 9](#_Toc52971122)

[4.3 convert\_tlm\_file 9](#_Toc52971123)

[Appendix A: EDS Python Bindings 9](#_Toc52971124)

[A.1: EDSLIB Python Bindings 9](#_Toc52971125)

[A.2: CFE\_MissionLib Python Bindings 11](#_Toc52971126)

[Appendix B: Acronyms 13](#_Toc52971127)

List of Figures

[Figure 1. Main cFS-EDS-GroundStation window 5](#_Toc52971128)

[Figure 2. Telecommand System 6](#_Toc52971129)

[Figure 3. Telemetry System (Startup) 7](#_Toc52971130)

[Figure 4. Telemetry System (Running) 8](#_Toc52971131)

[Figure 5. EDSLIB Python Bindings Example 10](#_Toc52971132)

[Figure 6. EDSLIB Iterators 11](#_Toc52971133)

[Figure 7. CFE\_MissionLib Python Bindings Example 12](#_Toc52971134)

[Figure 8. CFE\_MissionLib Instance Iterator 12](#_Toc52971135)

[Figure 9. CFE\_MissionLib Topic Iterator 13](#_Toc52971136)

[Figure 10. CFE\_MissionLib Subcommand Iterator 13](#_Toc52971137)

# Prerequisites:

The cFS-EDS-GroundStation software is based in Python3 and uses the PyQt5 framework for the graphical user interface. Testing has taken place on Ubuntu “LTS” distributions. The following development packages must be installed on the host. Note these are names of Debian/Ubuntu packages; other Linux distributions should provide a similar set, but the package names may vary.

* python3-dev
* python3-pyqt5

# Installation:

The cFS-EDS-GroundStation software is designed to be integrated with the cfe-eds-framework repository which can be found at: <https://github.com/jphickey/cfe-eds-framework>. Downloading the cFS-EDS-GroundStation software to <cfs-home>/tools, the directory can be added to the cfs cmake build process via cfe/cmake/mission\_build.cmake

# Include all the EDS libraries and tools which are built for the host system

include\_directories(${MISSION\_BINARY\_DIR}/inc)

add\_subdirectory(${MISSION\_SOURCE\_DIR}/tools/eds/edslib eds/edslib)

add\_subdirectory(${MISSION\_SOURCE\_DIR}/tools/eds/tool eds/tool)

add\_subdirectory(${MISSION\_SOURCE\_DIR}/tools/eds/cfecfs eds/cfecfs)

**add\_subdirectory(${MISSION\_SOURCE\_DIR}/tools/cFS-EDS-GroundStation eds/cFS-EDS-GroundStation)**

There are three cmake variables that need to be turned on to run the cFS-EDS-GroundStation:

* EDSLIB\_PYTHON\_BUILD\_STANDALONE\_MODULE: This tells the cFS build process to compile the standalone EdsLib python library that can be imported into a python3 instance.
* CFE\_MISSIONLIB\_PYTHON\_BUILD\_STANDALONE\_MODULE: This tells the cFS build process to compile the standalone CFE\_MissionLib python library that can be imported into a python3 instance.
* CONFIGURE\_CFS\_EDS\_GROUNDSTATION: This takes the mission name, defined in “targets.cmake” in the mission definition folder, and inserts it into the cFS-EDS-GroundStation software files. The files are then copied into the build directory under “<build\_path>/exe/host/cFS-EDS-GroundStation/”

The cmake variables can be set using the PREP\_OPTS variable in the cFS project Makefile

PREP\_OPTS := -DEDSLIB\_PYTHON\_BUILD\_STANDALONE\_MODULE=ON

-DCFE\_MISSIONLIB\_PYTHON\_BUILD\_STANDALONE\_MODULE=ON

-DCONFIGURE\_CFS\_EDS\_GROUNDSTATION=ON

Alternatively, once the prep step has been called, the variables can be manually changed in the “<build\_path>/CMakeCache.txt” file.

EDSLIB\_PYTHON\_BUILD\_STANDALONE\_MODULE:BOOL=ON

CFE\_MISSIONLIB\_PYTHON\_BUILD\_STANDALONE\_MODULE:BOOL=ON

CONFIGURE\_CFS\_EDS\_GROUNDSTATION:BOOL=ON

With the cmake variables set, follow the normal procedure to build cFS:

$> make SIMULATION=native O=<build\_path> prep

$> make

$> make install

When the build process is complete, several modules will be created in the build directory. The EDS and CFE\_MissionLib databases will be written to the “<build\_path>/exe/lib/” folder under the files <mission\_name>\_eds\_db.so” and “<mission\_name>\_eds\_interfacedb.so” respectively.

The python modules will be written to “<build\_path>/exe/lib/EdsLib.so” and “<build\_path>/exe/lib/python/CFE\_MissionLib.so”. Since all of these files will need to be either imported or linked to a python instance the “<build\_path>/exe/lib” and “<build\_path>/exe/lib/python” folders need to be added to LD\_LIBRARY\_PATH and PYTHONPATH environment variables respectively. For example in a “.bashrc” file the following lines can be added.

export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:<full\_build\_path>/exe/lib

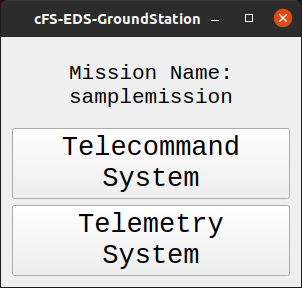
export PYTHONPATH=$PYTHONPATH:<full\_build\_path>/exe/lib/python

# Execution:

After the cFS build process is complete, the cFS-EDS-GroundStation software can be found in the “<build\_path>/exe/host/cFS-EDS-GroundStation/” folder. The software can be run using the following command within that folder.

$> python3 cFS-EDS-GroundStation.py

This opens the main window of the cFS-EDS-GroundStation software, shown in Figure 1.

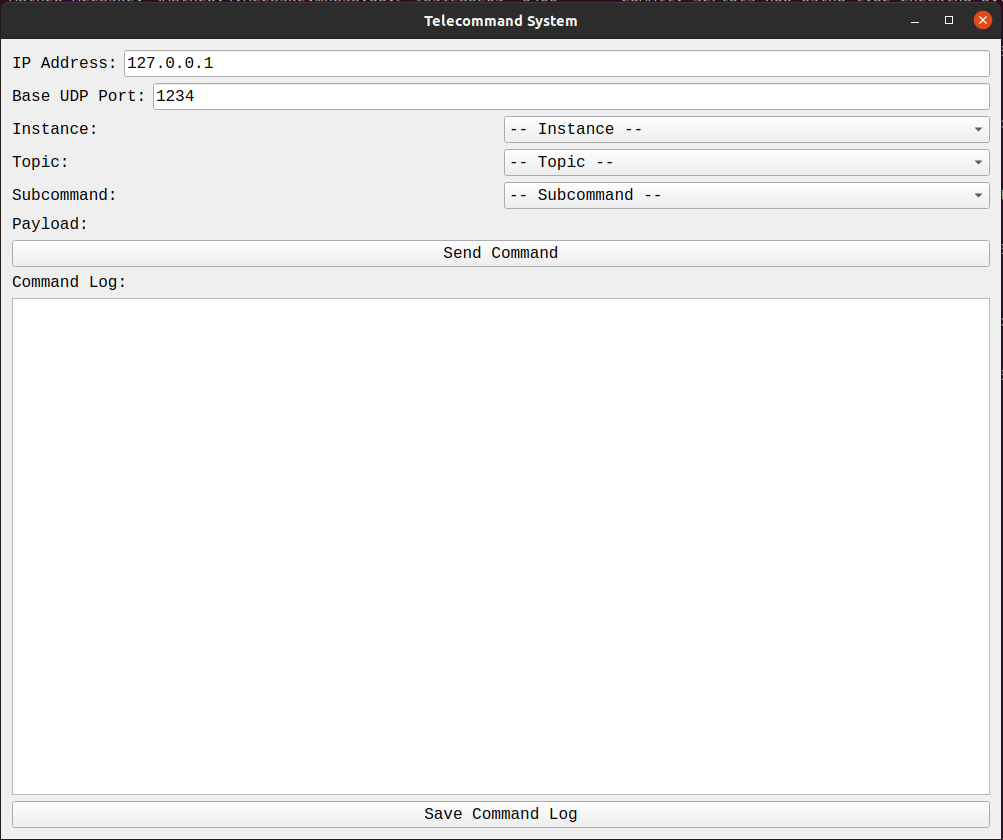


1. Main cFS-EDS-GroundStation window

The mission name is already configured through the cFS build process. The two buttons on this window opens the Telecommand and Telemetry Systems.

## Telecommand System:

The Telecommand System allows a user to send commands to an instance of core flight and is shown in Figure 2.



1. Telecommand System

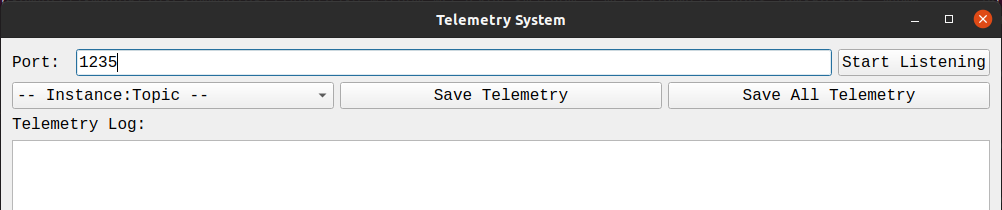
The interface has a series of inputs to determine which command to send and where. An outline of the user inputs is as follows:

* IP Address: The IP address to send the command. By default, this is set to the home IP address (127.0.0.1)
* Base UDP Port: The base port used to send the command. By default, this is set to 1234.  
  Port = Base UDP Port + Instance ID - 1
* Instance: The destination cFS instance can be selected by a dropdown menu containing a list of potential cFS instances.
* Topic: The telecommand topic can be selected by a dropdown menu that contains a list of all possible telecommand topics.
* Subcommand: If a Topic is selected that has subcommands associated with it, a list of subcommands is populated and can be selected by a dropdown menu.
* Payload: If the selected Topic or Subcommand has payload values associated with the command message, entries will appear for each payload value. Enumerations labels are populated in a dropdown menu while numerical and string payload values are entered with a text window.

Once everything is entered, the “Send Command” button can be clicked. If there are any numerical payload values, they are checked to assure that a number was entered. An EDS command is then created and sent to the specified cFS instance. A summary of the command is displayed in the command log. The “Save Command Log” button writes all the sent commands to a time-stamped file in the output folder.

## Telemetry System

The Telemetry System listens for messages sent from the Telemetry Output application of a cFS instance. Figure 3 shows the Telemetry System when it is initially started.



1. Telemetry System (Startup)

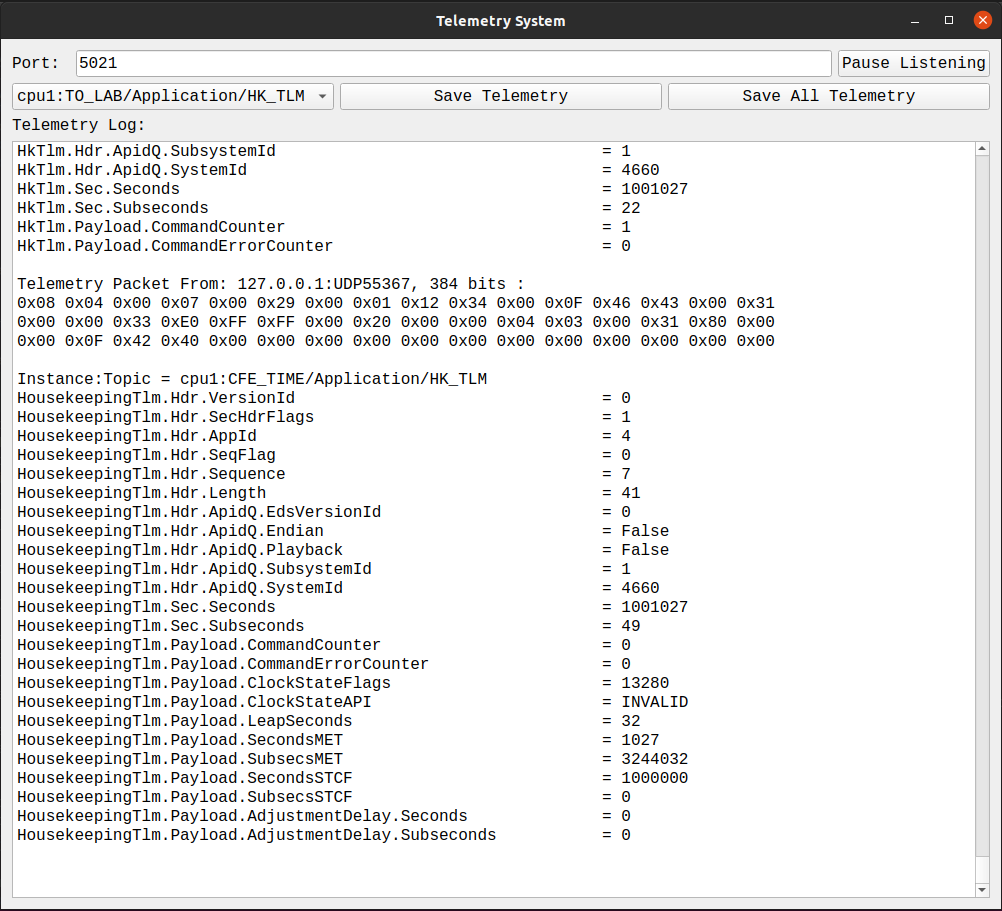
First, the desired port can be specified by the user. By default, the Telemetry Output applications send messages to port 1235. The “Start Listening” button can then be clicked which starts the continuous process of listening for incoming messages.

Note: By default, cFS instances do not have telemetry enabled. It can be enabled by sending the following command with the Telecommand System:

* Instance = cpu1
* Topic = TO\_LAB/Application/CMD
* Subcommand = EnableOutput
* Dest\_IP = 127.0.0.1

As telemetry messages arrive, they are automatically decoded into EDS objects within Python.

These objects can be parsed, and their contents displayed in the Telemetry Log. Figure 4 shows the Telemetry System while message listening has started.



1. Telemetry System (Running)

The header portion of the telemetry message contains information about the cFS instance the message originated from and the topic associated with each message. Therefore, each message can be sorted by this “<instance>:<topic>” pair and stored in an internal array. The telemetry messages can further be saved to a time-stamped binary file either by specifying an “<instance>:<topic>” pair and clicking the “Save Telemetry” button or simply clicking the “Save All Telemetry” button. All saved telemetry files are written to the output folder

The telemetry listening can be temporarily paused after it has been started by clicking the “Pause Listening” button in the upper right corner of the Telemetry System. The label on the button changes to “Resume Listening” and can be clicked again to resume telemetry listening.

# Utility Scripts

Along with the GUI, there are three python scripts that provide additional functionality. These scripts are found in the “<build\_path>/exe/host/cFS-EDS-GroundStation/utils” folder.

## cmd\_util

The cmd\_util python script essentially performs the same function as the Telecommand System in the cFS-EDS-GroundStation software. Instead of a graphical user interface, the script gathers the information needed to create and send a command message through a series of user prompts: cFS instance, topic, subcommand (if necessary), destination IP (with 127.0.0.1 set as default), and destination port base (with 1234 set as default). This script can be run using the following command in the utils folder

$> python3 cmd\_util.py

## tlm\_decode

The tlm\_decode python script performs the telemetry listening/decoding function of the Telemetry System. The script can be run using the following command in the utils folder

$> python3 tlm\_decode (--port=1235)

The port used to listen for telemetry messages can be specified with the port option. Similarly to the cFS-EDS-GroundStation, the default port is set to 1235. As messages arrive, they are decoded into an EDS object within python. The objects are parsed and displayed to the screen in a similar manner to the telemetry log in the Telemetry System.

## convert\_tlm\_file

This script converts the binary telemetry files written from the Telemetry System into a CSV file. The script can be run with the following command

$> python3 convert\_tlm\_file.py –-file=<filename>

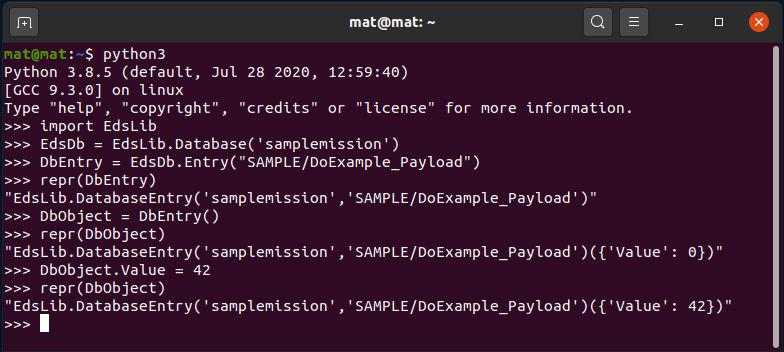
where <filename> is the telemetry binary file one wishes to convert. The script opens the binary file and reads through the saved messages. For each message, it is automatically decoded into an EDS object in Python, and the components of the message are written in an unpacked format to a new csv file with the same base name as the telemetry binary file. These csv files can then be opened in any Excel type program for further processing, plotting, etc.

1. EDS Python Bindings

The cFS-EDS-GroundStation software utilizes two sets of python bindings to run, EDSLIB and CFE\_MissionLib. EDSLIB contains the generic structure information defined in all of the EDS files throughout the mission. CFE\_MissionLib contains the information needed to interface EDSLIB with cFS, as in general EDSLIB is application agnostic.

* 1. EDSLIB Python Bindings

The EDSLIB python bindings allow a user to create EDS objects within python. Figure 5 shows a few examples of how these objects are created.

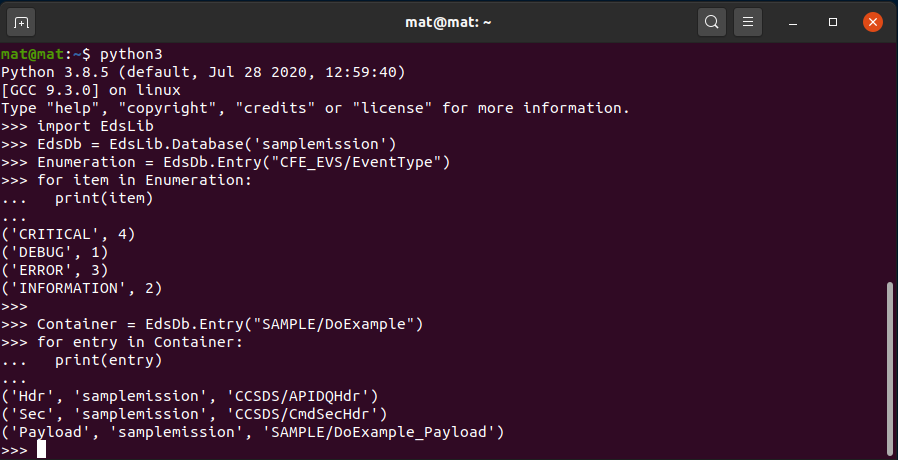


1. EDSLIB Python Bindings Example

Going line by line in the terminal window:

* Import the EdsLib python module into the python instance
* Create the EDS database object. Note the argument is the mission name which can be seen in the main cFS-EDS-GroundStation window (Figure 1)
* Create an EDS database entry. The argument follows the “<Package>/<Object>” structure which is the common way each EDS object can be identified. EDS database entries are essentially functions that can be called to create an EDS object.
* The representation of the EDS database entry shows the mission name and the EDS “<Package>/<Object>” identifier.
* Calling the EDS database entry function creates an EDS object. With no argument the object is initialized to all zeros, but it can be initialized with an appropriate object as the argument.
* The representation of the EDS object now shows the data structure. EDS containers are specifically treated as Python dictionaries.
* Since this example is a container, we can set the payload value referencing the specified entry in the dictionary.
* Now the representation of the EDS object shows the updated value in the data structure.

There are a few iterators that are used within the cFS-EDS-GroundStation software that are quite handy. Figure 6 shows two examples of these iterators.



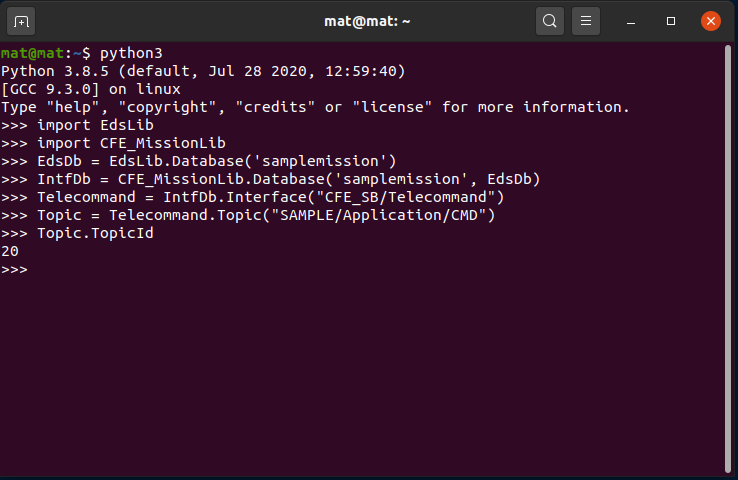
1. EDSLIB Iterators

For EDS enumeration objects, the iterator loops over all the label/value pairs. This example is simply printing them out, but the cFS-EDS-GroundStation software uses them to populate dropdown menus for enumeration payload values in the Telecommand System.

For EDS entries associated with container objects (this example is using the DoExample command from the Sample App), the iterator loops over all the sub-objects and obtains the information needed to create each within Python. There is a base label, the mission name, and the “<Package>/<Object>” identifier. This is how the Telecommand System can determine if a particular command has a payload structure and what entry fields are necessary for the user to fill. The Telemetry System also uses these iterators to display the entire contents of a telemetry message in the telemetry log.

* 1. CFE\_MissionLib Python Bindings

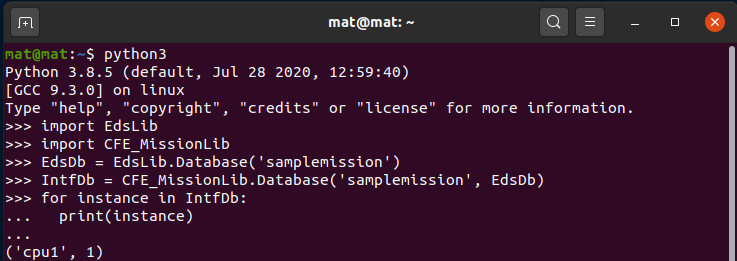
The CFE\_MissionLib Python bindings allow a user to create the interface objects that link EDS with cFS. Figure 7 shows a few examples of how these interface objects are created.



1. CFE\_MissionLib Python Bindings Example

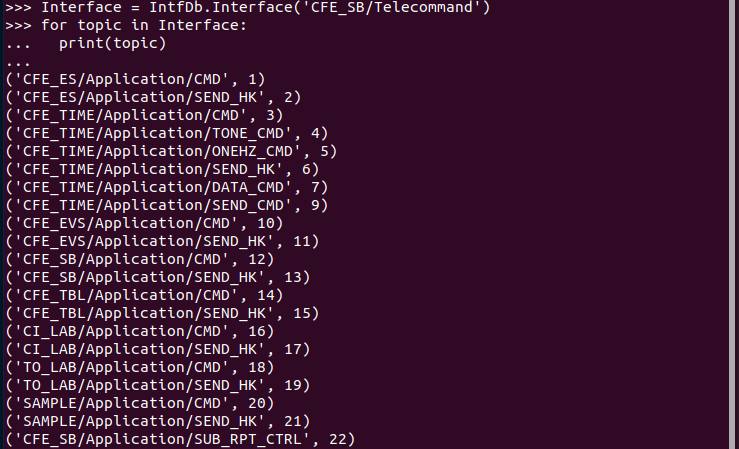
CFE\_MissionLib is dependent on EdsLib, so the EDS database needs to be created first so it can be used as an argument to create the interface database. Once created, an interface object can be created from the interface database. Typically, users will only use “CFE\_SB/Telecommand” or “CFE\_SB/Telemetry” as the arguments to create a telecommand or telemetry interface respectively. With an interface object, a topic object can be created using the “<Package>/Application/<TopicName>” identifier defined in the EDS files. Each Topic has a “TopicId” member which gives the ID associated with that topic defined in the mission configuration EDS files (<mission>\_defs/eds/cfe-topicids.xml).

There are several iterators which can be used in the CFE\_MissionLib Python bindings. Iterating over the interface database object will show the cFS instance name/ID pairs that the library can connect to (Figure 8).



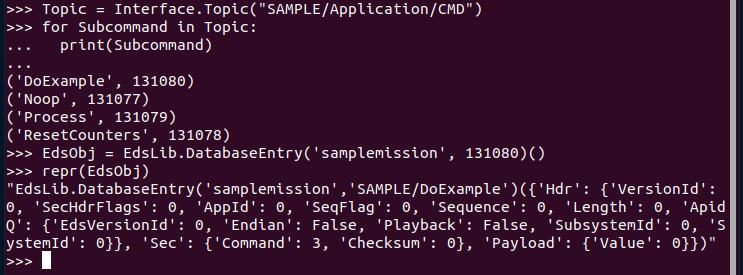
1. CFE\_MissionLib Instance Iterator

Iterating over an interface object will list all of the Topics associated with that instance as well as the corresponding TopicIDs (Figure 9)



1. CFE\_MissionLib Topic Iterator

Finally, iterating over a topic that has subcommands will list the subcommand name as well as the numerical EdsId associated with the object. This is the link between CFE\_MissionLib and EdsLib as we can use this number to create the Eds object associated with the subcommand (Figure 10).



1. CFE\_MissionLib Subcommand Iterator
2. Acronyms

|  |  |
| --- | --- |
| Acronym | Definition |
| App | Application |
| cFS | Core Flight System |
| CSV | Comma Separated Value |
| EDS | Electronic Data Sheets |
| GUI | Graphical User Interface |
| IP | Internet Protocol |
| LTS | Long Term Service |